Página 3

IT Revolution Press, LLC

25 NW 23rd Pl, Suite 6314

Portland, OR 97210

Primera edición

Impreso en los Estados Unidos de América

10 9 8 7 6 5 4 3 2 1

Diseño de portada por Strauber Design Studio

Ilustración de portada por eboy

Diseño de libro por Mammoth Collective

Diseño de libro electrónico por Carpeta digital

ISBN de impresión: 978-1942788003

Ebook – ISBN ISBN: 978-1-942788-07-2

Ebook – ISBN ISBN: 978-1-942788-08-9

Número de control de la Biblioteca del Congreso: 2016951904

Nota del editor a los lectores: muchas de las ideas, citas y paráfrasis atribuido a diferentes pensadores y líderes de la industria aquí se extraen de conversaciones informales, correspondencia, entrevistas, conferencias mesas redondas y otras formas de comunicación oral que tuvieron lugar durante los últimos seis años durante el desarrollo y la escritura de este libro. A pesar de que los autores y el editor han hecho todo lo posible para garantizar que la información en este libro era correcta al momento de la publicación, los autores y el editor no asume y por la presente renuncia a cualquier responsabilidad ante cualquier parte por cualquier pérdida, daño o interrupción causada por errores u omisiones, ya sea

Prefacio

¡Ajá!

El viaje para completar *el Manual de DevOps* tiene
ha sido largo, comenzó con Skype trabajando semanalmente
convocatorias entre los coautores en febrero de 2011, con
La visión de crear una guía prescriptiva que
servir como compañero del libro aún no terminado *The*
Proyecto Phoenix: una novela sobre TI, DevOps y
Ayudando a su empresa a ganar .

Más de cinco años después, con más de dos mil horas.
de trabajo, *el manual DevOps* finalmente está aquí.
Completar este libro ha sido extremadamente largo
proceso, aunque uno que ha sido altamente gratificante
y lleno de aprendizaje increíble, con un alcance que es mucho
más amplio de lo que imaginamos originalmente. Durante el
proyecto, todos los coautores compartieron la creencia de que DevOps es
genuinamente importante, formado en un "ajá" personal
momento mucho antes en cada una de nuestras carreras profesionales,
con lo cual sospecho que muchos de nuestros lectores resonarán.

Gene Kim
He tenido el privilegio de estudiar de alto rendimiento
organizaciones tecnológicas desde 1999, y una de las
Los primeros hallazgos fueron que abarcan los límites.

Página 7

entre los diferentes grupos funcionales de TI
Operaciones, seguridad de la información y desarrollo
Fue crítico para el éxito. Pero aún recuerdo el primero
vez que vi la magnitud de la espiral descendente que
resultaría cuando estas funciones trabajaran hacia
objetivos opuestos

Era 2006, y tuve la oportunidad de pasar un
semana con el grupo que manejó la TI tercerizada
Operaciones de un gran servicio de reserva de aerolíneas. Ellos
describió las consecuencias aguas abajo de su
grandes versiones anuales de software: cada versión sería

causar inmenso caos e interrupción para el
outsourcer, así como clientes; habría SLA
(acuerdo de nivel de servicio) sanciones, debido a la
interrupciones que afectan al cliente; habría despidos
del personal más talentoso y experimentado, debido a
el déficit de ganancias resultante; habría mucho
trabajo no planificado y lucha contra incendios para que el
el personal restante no podía trabajar en el creciente
pedidos atrasados de solicitudes de servicio procedentes de clientes; el
contrato se mantendría unido por la heroicidad de
gerencia intermedia; y todos sintieron que el
el contrato estaría condenado a ser presentado para volver a ofertar en
tres años.

La sensación de desesperanza y futilidad que resultó
creó para mí los inicios de una cruzada moral.
El desarrollo parecía ser visto siempre como
estratégico, pero las operaciones de TI fueron vistas como tácticas,
a menudo delegado o subcontratado por completo, solo para

Página 8

volver en cinco años en peor estado de lo que era primero
entregado.

Durante muchos años, muchos de nosotros supimos que debe haber
una mejor manera. Recuerdo haber visto las conversaciones saliendo
de la Conferencia Velocity 2009, que describe increíbles
resultados habilitados por arquitectura, técnica
prácticas y normas culturales que ahora conocemos como

DevOps. Estaba tan emocionado, porque claramente apuntaba a la mejor manera que todos habíamos estado buscando. Y ayudar a difundir esa palabra fue uno de mis personales Motivaciones para el coautor del *Proyecto Phoenix* . Tú puedo imaginar lo increíblemente gratificante que era ver la comunidad en general reacciona a ese libro, describiendo cómo les ayudó a lograr su propio "ajá" momentos

Jez Humble

Mi momento "ajá" de DevOps fue en un arranque en 2000 —Mi primer trabajo después de graduarme. Por un tiempo estuve Uno de los dos técnicos. Yo hice todo: redes, programación, soporte, sistemas administración. Implementamos software para producción por FTP directamente desde nuestras estaciones de trabajo.

Luego, en 2004, conseguí un trabajo en ThoughtWorks, un consultoría donde mi primer concierto estaba trabajando en un proyecto que involucra a unas setenta personas. Estaba en un equipo de ocho ingenieros cuyo trabajo a tiempo completo era desplegar nuestro software en una producción ambiente. Al principio, fue realmente estresante. Pero en unos meses pasamos del manual

Página 9

implementaciones que tardaron dos semanas en automatizarse despliegue que tomó una hora, donde pudimos rodar hacia adelante y hacia atrás en milisegundos usando el azul

patrón de implementación verde durante el negocio normal
horas

Ese proyecto inspiró muchas de las ideas tanto en el
Libro de *entrega continua* (Addison-Wesley, 2000)
y éste. Mucho de lo que me impulsa a mí y a los demás
trabajar en este espacio es saber que, sea lo que sea
sus limitaciones, siempre podemos hacerlo mejor, y el
deseo de ayudar a las personas en su viaje.

Patrick Debois

Para mí, fue una colección de momentos. En 2007 estaba
trabajando en un proyecto de migración del centro de datos con algunos
Equipos ágiles. Estaba celoso de que tuvieran tanto
productividad: capaz de hacer tanto en tan poco
hora.

Para mi próxima tarea, comencé a experimentar con
Kanban en operaciones y vio cómo la dinámica de
El equipo cambió. Más tarde, en el Agile Toronto 2008
conferencia presenté mi documento IEEE sobre esto, pero yo
sentía que no resonó ampliamente en la comunidad ágil.
Comenzamos un grupo de administración del sistema ágil, pero
Pasé por alto el lado humano de las cosas.

Después de ver la Conferencia Velocity 2009
presentación "10 implementaciones por día" por John Allspaw
y Paul Hammond, estaba convencido de que otros estaban
pensando de manera similar. Entonces decidí organizar el

primeros DevOpsDays, acuñando accidentalmente el término DevOps.

La energía en el evento fue única y contagiosa. Cuando la gente comenzó a agradecerme porque cambió su vida para mejor, entendí el impacto. yo No he dejado de promocionar DevOps desde entonces.

John Willis

En 2008, acababa de vender un negocio de consultoría que enfocado en prácticas de operaciones de TI heredadas a gran escala alrededor de la gestión de configuración y monitoreo (Tivoli) cuando conocí a Luke Kanies (el fundador de Laboratorios de marionetas). Luke estaba haciendo una presentación sobre Marioneta en una conferencia de código abierto de O'Reilly sobre gestión de configuración (CM).

Al principio estaba pasando el rato en la parte de atrás de la habitación matando el tiempo y pensando: "¿Qué podría este veinte- de un año me cuenta sobre la gestión de la configuración? Después de todo, literalmente había estado trabajando toda mi vida en Algunas de las empresas más grandes del mundo, ayudando ellos arquitecto CM y otra gestión de operaciones soluciones Sin embargo, unos cinco minutos en su sesión, pasé a la primera fila y me di cuenta todo lo que había estado haciendo durante los últimos veinte años estaba mal. Luke estaba describiendo lo que ahora llamo segunda generación CM.

Después de su sesión tuve la oportunidad de sentarme y tomar un café con él Estaba totalmente vendido por lo que ahora llame a infraestructura como código. Sin embargo, mientras nosotros se reunió para tomar un café, Luke comenzó a ir aún más lejos

explicando sus ideas. Comenzó a decirme que creía
que las operaciones iban a tener que comenzar a comportarse
como los desarrolladores de software. Iban a tener que
mantener sus configuraciones en control de fuente y adoptar
Patrones de entrega de CI / CD para su flujo de trabajo. Siendo el
antigua persona de operaciones de TI en ese momento, creo que respondí
para él con algo como: "Esa idea se va a hundir
como Led Zeppelin con la gente de Ops ". (Estaba claramente
incorrecto.)

Luego, aproximadamente un año después en 2009 en otro O'Reilly
conferencia, Velocity, vi a Andrew Clay Shafer dar un
Presentación sobre Infraestructura Ágil. En su
presentación, Andrew mostró esta imagen icónica de un
muro entre desarrolladores y operaciones con una
representación metafórica del trabajo arrojado sobre el
pared. Él acuñó este "muro de confusión". Las ideas
expresó en esa presentación codificó lo que Lucas
Estaba tratando de decirme un año antes. Esa era la luz
bombilla para mi. Más tarde ese año, fui el único estadounidense
Invitado a los DevOpsDays originales en Gante. Por el
cuando ese evento terminó, a esto lo llamamos DevOps
Estaba claramente en mi sangre.

Claramente, los coautores de este libro llegaron a una situación similar.
epifanía, incluso si vinieron desde muy diferentes
direcciones. Pero ahora hay un peso abrumador de
evidencia de que los problemas descritos anteriormente suceden

casi en todas partes, y que las soluciones asociadas
con DevOps son casi universalmente aplicables.

El objetivo de escribir este libro es describir cómo
replicar las transformaciones DevOps de las que hemos sido parte

de o han observado, así como disipar muchos de los mitos
de por qué DevOps no funcionará en ciertas situaciones. Abajo
son algunos de los mitos más comunes que escuchamos
DevOps.

Mito : *DevOps es solo para startups:* mientras que DevOps
las prácticas han sido pioneras en la escala web, Internet
Empresas "unicornio" como Google, Amazon, Netflix,
y Etsy, cada una de estas organizaciones tiene, en algún momento
en su historia, corrían el riesgo de salir del negocio debido a
los problemas asociados con el "caballo" más tradicional
organizaciones: lanzamientos de códigos altamente peligrosos que fueron
propenso a fallas catastróficas, incapacidad para liberar funciones
lo suficientemente rápido como para vencer a la competencia, cumplimiento
preocupaciones, incapacidad para escalar, altos niveles de desconfianza
entre Desarrollo y Operaciones, y así sucesivamente.

Sin embargo, cada una de estas organizaciones pudo
transformar su arquitectura, prácticas técnicas y
cultura para crear los sorprendentes resultados que asociamos
con DevOps Como el Dr. Branden Williams, una información
ejecutivo de seguridad, bromeó: "Que no se hable más de
DevOps unicornios o caballos pero solo purasangres y
caballos que se dirigen a la fábrica de pegamento.

Mito : *DevOps reemplaza a Agile:* principios de DevOps y las prácticas son compatibles con Agile, con muchas observaciones que DevOps es una continuación lógica del viaje ágil que comenzó en 2001. Ágil a menudo sirve como un efectivo habilitador de DevOps, debido a su enfoque en equipos pequeños entregando continuamente código de alta calidad a los clientes.

Página 13

Muchas prácticas de DevOps surgen si continuamos administrando nuestro trabajo más allá del objetivo de "código potencialmente enviable" al final de cada iteración, extendiéndola a tener nuestro código siempre en estado desplegable, con desarrolladores registrarse en el baúl todos los días, y que demostramos nuestra características en entornos de producción.

Mito : *DevOps es incompatible con ITIL:* muchas vistas DevOps como reacción a ITIL o ITSM (Servicio de TI Gestión), que se publicó originalmente en 1989.
ITIL ha influido ampliamente en múltiples generaciones de operaciones profesionales, incluido uno de los coautores, y es un biblioteca de prácticas en constante evolución destinada a codificar el procesos y prácticas que sustentan TI de clase mundial Operaciones, abarcando estrategia de servicio, diseño y apoyo.

Las prácticas de DevOps pueden hacerse compatibles con ITIL proceso. Sin embargo, para soportar los plazos de entrega más cortos y

frecuencias de implementación más altas asociadas con DevOps, muchas áreas de los procesos ITIL se vuelven completamente automatizado, resolviendo muchos problemas asociados con el procesos de configuración y gestión de versiones (por ejemplo, mantener la base de datos de gestión de configuración y bibliotecas de software definitivas actualizadas). Y porqué DevOps requiere una detección y recuperación rápidas cuando ocurren incidentes de servicio, las disciplinas de servicio de ITIL el diseño, el incidente y la gestión de problemas permanecen como relevante como siempre.

Mito : *DevOps es incompatible con la información Seguridad y cumplimiento:* la ausencia de tradicional controles (por ejemplo, segregación de funciones, aprobación de cambios

Página 14

procesos, revisiones manuales de seguridad al final del proyecto) puede consternar la seguridad de la información y profesionales de cumplimiento.

Sin embargo, eso no significa que las organizaciones DevOps No tiene controles efectivos. En lugar de seguridad y las actividades de cumplimiento solo se realizan al final de En el proyecto, los controles están integrados en cada etapa de trabajo diario en el ciclo de vida del desarrollo de software, resultando en una mejor calidad, seguridad y cumplimiento resultados.

Mito : *DevOps significa eliminar las operaciones de TI, o "NoOps":* muchos malinterpretan DevOps como el completo eliminación de la función de operaciones de TI. Sin embargo, esto

Raramente es el caso. Si bien la naturaleza de las operaciones de TI funciona puede cambiar, sigue siendo tan importante como siempre. ESO
Operaciones colabora mucho antes en la vida del software
ciclo con Desarrollo, que continúa trabajando con TI
Operaciones mucho después de que el código se haya implementado en
producción.

En lugar de que las operaciones de TI hagan el trabajo manual que viene
desde tickets de trabajo, permite la productividad del desarrollador
a través de API y plataformas de autoservicio que crean
entornos, probar e implementar código, monitorear y mostrar
telemetría de producción, etc. Al hacer esto, TI
Las operaciones se vuelven más como Desarrollo (al igual que QA
e Infosec), dedicados al desarrollo de productos, donde
el producto es la plataforma que los desarrolladores usan de manera segura,
pruebe, implemente y ejecute su TI de forma rápida y segura
servicios en produccion.

Página 15

Mito : *DevOps es solo "Infraestructura como código" o*
Automatización: mientras se muestran muchos de los patrones de DevOps
en este libro requieren automatización, DevOps también requiere
normas culturales y una arquitectura que permita la
objetivos compartidos que se alcanzarán en todo el valor de TI
corriente. Esto va mucho más allá de la automatización. Como
Christopher Little, un ejecutivo de tecnología y uno de los
los primeros cronistas de DevOps escribieron: "DevOps no es
sobre automatización, así como la astronomía no se trata

telescopios ".

Mito : *DevOps es solo para software de código abierto:*

Aunque muchas historias de éxito de DevOps tienen lugar en organizaciones que usan software como la pila LAMP (Linux, Apache, MySQL, PHP), logrando DevOps Los resultados son independientes de la tecnología utilizada. Se han logrado éxitos con aplicaciones escritas en Microsoft.NET, COBOL y ensamblaje de mainframe código, así como con SAP e incluso sistemas integrados (por ejemplo, firmware HP LaserJet).

¡SEPARA EL AHA! MOMENTO

Cada uno de los autores se ha inspirado en la increíble innovaciones que suceden en la comunidad DevOps y los resultados que están creando: están creando seguros sistemas de trabajo, y permitiendo que pequeños equipos rápidamente y desarrollar y validar independientemente el código que puede ser desplegado de forma segura a los clientes. Dada nuestra creencia de que DevOps es una manifestación de crear aprendizaje dinámico organizaciones que refuerzan continuamente la alta confianza

normas culturales, es inevitable que estas organizaciones continuará innovando y ganando en el mercado.

Es nuestra sincera esperanza de que *el Manual DevOps* voluntad servir como un recurso valioso para muchas personas en diferentes maneras: una guía para planificar y ejecutar DevOps

transformaciones, un conjunto de estudios de caso para investigar y aprender de, una crónica de la historia de DevOps, un medio para crear una coalición que abarque propietarios de productos, Arquitectura, desarrollo, control de calidad, operaciones de TI y Seguridad de la información para lograr objetivos comunes, una forma de Obtenga los niveles más altos de apoyo de liderazgo para DevOps iniciativas, así como un imperativo moral para cambiar el forma en que gestionamos las organizaciones tecnológicas para permitir mejor efectividad y eficiencia, así como permitir un ambiente de trabajo más feliz y más humano, ayudando todos se convierten en aprendices de por vida, esto no solo ayuda todos alcanzan sus metas más altas como seres humanos, pero también ayuda a sus organizaciones a ganar.

Prefacio

En el pasado, muchos campos de la ingeniería han experimentado una especie de evolución notable, continuamente "nivelando" su comprensión de su propio trabajo. Mientras hay currículos universitarios y apoyo profesional organizaciones situadas dentro de disciplinas específicas de ingeniería (civil, mecánica, eléctrica, nuclear, etc.), El hecho es que la sociedad moderna necesita todas las formas de ingeniería reconocer los beneficios y trabajar en un forma multidisciplinaria

Piense en el diseño de un vehículo de alto rendimiento. ¿Dónde termina el trabajo de un ingeniero mecánico y comienza el trabajo de un ingeniero eléctrico? Dónde y cómo y cuándo) debería alguien con dominio conocimiento de la aerodinámica (quien ciertamente tendría opiniones bien formadas sobre la forma, el tamaño y la ubicación de windows) colabora con un experto en pasajero ¿ergonomía? ¿Qué pasa con las influencias químicas del combustible? mezcla y aceite sobre los materiales del motor y transmisión durante la vida útil del vehículo? Existen otras preguntas que podemos hacer sobre el diseño de un automóvil, pero el resultado final es el mismo: éxito en los esfuerzos técnicos modernos requieren absolutamente múltiples perspectivas y experiencia para colaborar.

Para que un campo o disciplina progrese y madure, necesita llegar a un punto donde pueda reflexionar cuidadosamente

en sus orígenes, busque un conjunto diverso de perspectivas sobre esas reflexiones y ubican esa síntesis en un contexto eso es útil para cómo la comunidad imagina el futuro.

Este libro representa tal síntesis y debería ser visto como una colección seminal de perspectivas sobre el (lo haré discutir, aún emergente y evolucionando rápidamente) campo de ingeniería de software y operaciones.

No importa en qué industria se encuentre, o en qué producto o servicio que brinda su organización, esta forma de pensar es primordial y necesario para la supervivencia de cada Líder empresarial y tecnológico.

—John Allspaw, CTO, Etsy

Brooklyn, NY, agosto de 2016

Imagina un mundo donde Dev y Ops

Conviértete en DevOps

Una introducción al manual de DevOps

Imagine un mundo donde los propietarios de productos, desarrollo, control de calidad, operaciones de TI e Infosec trabajar juntos, no solo para ayudarse mutuamente, sino también para garantizar que la organización en general tiene éxito Al trabajar hacia un objetivo común, permiten el flujo rápido de trabajo planificado en producción (por ejemplo, realizar decenas, cientos o incluso miles de implementaciones de código por día), mientras se logra estabilidad, confiabilidad, disponibilidad y seguridad de clase mundial.

En este mundo, los equipos interfuncionales prueban rigurosamente sus hipótesis sobre qué características La mayoría deleita a los usuarios y promueve los objetivos de la organización. No solo les importa implementando funciones de usuario, pero también asegurando activamente que su trabajo fluya sin problemas y frecuentemente a través de todo el flujo de valor sin causar caos e interrupción a TI Operaciones o cualquier otro cliente interno o externo.

Simultáneamente, QA, IT Operations e Infosec siempre están trabajando en formas de reducir fricción para el equipo, creando los sistemas de trabajo que permiten a los desarrolladores ser más productivo y obtener mejores resultados. Al agregar la experiencia de control de calidad, operaciones de TI y Infosec en equipos de entrega y herramientas y plataformas automáticas de autoservicio, los equipos pueden usar esa experiencia en su trabajo diario sin depender de otros equipos.

Esto permite a las organizaciones crear un sistema de trabajo seguro, donde pequeños equipos pueden Desarrolle, pruebe e implemente de forma rápida e independiente el código y el valor de forma rápida, segura, de forma segura y confiable para los clientes. Esto permite a las organizaciones maximizar el desarrollador productividad, permitir el aprendizaje organizacional, crear una alta satisfacción de los empleados y ganar el mercado.

Estos son los resultados que resultan de DevOps. Para la mayoría de nosotros, este no es el mundo que vivir. Más a menudo que no, el sistema en el que trabajamos está roto, lo que resulta en extremadamente pobre resultados que están muy por debajo de nuestro verdadero potencial. En nuestro mundo, desarrollo y TI Las operaciones son adversarias; las pruebas y las actividades de Infosec ocurren solo al final de un proyecto, demasiado tarde para corregir cualquier problema encontrado; y casi cualquier actividad crítica también requiere mucho esfuerzo manual y demasiadas transferencias, dejándonos siempre esperando. No solo

esto contribuye a tiempos de entrega extremadamente largos para hacer cualquier cosa, pero la calidad de nuestro El trabajo, especialmente los despliegues de producción, también es problemático y caótico, lo que resulta en Impactos negativos para nuestros clientes y nuestro negocio.

Como resultado, no cumplimos con nuestros objetivos, y toda la organización está insatisfecha con el rendimiento de TI, lo que resulta en reducciones de presupuesto y empleados frustrados e infelices que sentirse impotente para cambiar el proceso y sus resultados. ¿La solución? Necesitamos cambiar cómo trabajamos; DevOps nos muestra el mejor camino a seguir.

Para comprender mejor el potencial de la revolución DevOps, veamos el Revolución de fabricación de la década de 1980. Al adoptar los principios y prácticas Lean, Las organizaciones de fabricación mejoraron drásticamente la productividad de la planta, el liderazgo del cliente tiempos, calidad del producto y satisfacción del cliente, lo que les permite ganar en el mercado.

Antes de la revolución, los plazos de entrega promedio de pedidos de plantas de fabricación eran de seis semanas, con Menos del 70% de los pedidos se envían a tiempo. Para 2005, con el generalizado

Página 21

implementación de prácticas Lean, los plazos de entrega promedio del producto se redujeron a menos de tres semanas, y más del 95% de los pedidos se enviaron a tiempo. Organizaciones que no implementó las prácticas Lean, perdió participación de mercado y muchos cerraron enteramente.

Del mismo modo, se ha elevado el listón para ofrecer productos y servicios tecnológicos: qué fue lo suficientemente bueno en décadas anteriores no es lo suficientemente bueno ahora. Para cada uno de los últimos cuatro décadas, el costo y el tiempo requeridos para desarrollar e implementar capacidades comerciales estratégicas y las características se han reducido en órdenes de magnitud. Durante las décadas de 1970 y 1980, la mayoría de los nuevos Las características requieren de uno a cinco años para desarrollarse y desplegarse, a menudo cuestan decenas de millones de dolares

En la década de 2000, debido a los avances tecnológicos y la adopción de principios ágiles y prácticas, el tiempo requerido para desarrollar una nueva funcionalidad se había reducido a semanas o meses, pero desplegarse en producción todavía requeriría semanas o meses, a menudo con catástrofes resultados.

Y para 2010, con la introducción de DevOps y la mercantilización interminable de hardware, software y ahora la nube, características (e incluso compañías de inicio completas) podrían crearse en semanas, desplegarse rápidamente en producción en solo horas o minutos, por En estas organizaciones, la implementación finalmente se convirtió en rutina y bajo riesgo. Estas organizaciones son capaces de realizar experimentos para probar ideas de negocios, descubriendo qué ideas crean el mayor valor para los clientes y la organización en su conjunto, que luego son más desarrollado en funciones que se pueden implementar de forma rápida y segura en la producción.

Tabla 1. *La tendencia cada vez más acelerada hacia una entrega de software más rápida, más barata y de bajo riesgo*

(Fuente: Adrian Cockcroft, "Velocity and Volume (or Speed Wins)", presentación en FlowCon, San Francisco, CA, noviembre de 2013.)

Hoy, las organizaciones que adoptan los principios y prácticas de DevOps a menudo implementan cambios cientos o incluso miles de veces por día. En una época donde la ventaja competitiva requiere un tiempo de comercialización rápido y una experimentación incesante, organizaciones que no pueden para replicar estos resultados están destinados a perder en el mercado a más ágil competidores y potencialmente podrían cerrar completamente, como la fabricación organizaciones que no adoptaron los principios Lean.

Página 22

En estos días, independientemente de en qué industria estamos compitiendo, la forma en que adquirimos clientes y entregarles valor depende de la corriente de valor de la tecnología. Pon aún más sucintamente, como declaró Jeffrey Immelt, CEO de General Electric, "Todas las industrias y la empresa que no está llevando el software al núcleo de su negocio se verá afectada ". O como Jeffrey Snover, miembro técnico de Microsoft, dijo: "En épocas económicas anteriores, las empresas valor creado moviendo átomos. Ahora crean valor moviendo bits ".

Es difícil exagerar la magnitud de este problema: afecta a todas las organizaciones, independientemente de la industria en la que operamos, el tamaño de nuestra organización, ya sea que seamos con o sin fines de lucro. Ahora más que nunca, cómo se gestiona y realiza el trabajo tecnológico predice si nuestras organizaciones ganarán en el mercado o incluso sobrevivirán. En muchos casos, tendremos que adoptar principios y prácticas que se vean muy diferentes de aquellos que nos han guiado con éxito en las últimas décadas. VerApéndice 1 .

Ahora que hemos establecido la urgencia del problema que resuelve DevOps, tomemos tiempo para explorar con más detalle la sintomatología del problema, por qué ocurre y por qué, sin una intervención dramática, el problema empeora con el tiempo.

EL PROBLEMA: ALGO EN TU ORGANIZACIÓN

En su lugar, la mayoría de las organizaciones no pueden implementar cambios de producción en minutos u horas. requiriendo semanas o meses. Tampoco pueden implementar cientos o miles de cambios en producción por día; en cambio, luchan por desplegarse mensualmente o incluso trimestralmente. Ni son rutinas de despliegues de producción, en lugar de interrupciones y extinción de incendios crónicos y heroica

En una era donde la ventaja competitiva requiere un tiempo de comercialización rápido, altos niveles de servicio, e incesante experimentación, estas organizaciones se encuentran en una competencia significativa desventaja. Esto se debe en gran parte a su incapacidad para resolver un conflicto central crónico dentro de su organización tecnológica.

EL NÚCLEO, CONFLICTO CRÓNICO

En casi todas las organizaciones de TI, existe un conflicto inherente entre Desarrollo y TI Operaciones que crean una espiral descendente, lo que resulta en un tiempo cada vez más lento para comercializar nuevos productos y características, menor calidad, mayores interrupciones y, lo peor de todo, una creciente cantidad de deuda técnica.

El término "deuda técnica" fue acuñado por primera vez por Ward Cunningham. Análogo a financiero deuda, la deuda técnica describe cómo las decisiones que tomamos conducen a problemas que cada vez más difícil de solucionar con el tiempo, reduciendo continuamente nuestras opciones disponibles en el futuro, incluso cuando se toma con prudencia, seguimos generando intereses.

Un factor que contribuye a esto son los objetivos a menudo competitivos del Desarrollo y TI Operaciones Las organizaciones de TI son responsables de muchas cosas. Entre ellos están los dos siguientes objetivos, que deben perseguirse simultáneamente:

- Responda al panorama competitivo que cambia rápidamente

Página 23

- Proporcionar un servicio estable, confiable y seguro al cliente.

Con frecuencia, el desarrollo se responsabilizará de responder a los cambios en el mercado, desplegar características y cambios en producción lo más rápido posible. Las operaciones de TI Asumir la responsabilidad de proporcionar a los clientes un servicio de TI estable, confiable y seguro, lo que hace que sea difícil o incluso imposible para cualquiera introducir cambios en la producción eso podría poner en peligro la producción. Configurado de esta manera, Desarrollo y Operaciones de TI tener objetivos e incentivos diametralmente opuestos.

Dr. Eliyahu M. Goldratt, uno de los fundadores de la gestión de fabricación. movimiento, llamado este tipo de configuración "el núcleo, conflicto crónico", cuando Las medidas e incentivos organizacionales en diferentes silos impiden el logro de objetivos globales y organizacionales.

Este conflicto crea una espiral descendente tan poderosa que impide el logro de lo deseado
resultados comerciales, tanto dentro como fuera de la organización de TI. Estos conflictos crónicos
a menudo ponen a los trabajadores tecnológicos en situaciones que conducen a una mala calidad de software y servicio,
y malos resultados para el cliente, así como una necesidad diaria de soluciones, extinción de incendios y
heroicidad, ya sea en gestión de productos, desarrollo, control de calidad, operaciones de TI o
Seguridad de información. VerApéndice 2 .

ESPIRAL HACIA ABAJO EN TRES ACTOS

La espiral descendente en TI tiene tres actos que probablemente sean familiares para la mayoría de los profesionales de TI.

El primer acto comienza en las operaciones de TI, donde nuestro objetivo es mantener las aplicaciones y
infraestructura en funcionamiento para que nuestra organización pueda ofrecer valor a los clientes. En nuestro diario
trabajo, muchos de nuestros problemas se deben a aplicaciones e infraestructura que son complejas,
mal documentado e increíblemente frágil. Esta es la deuda técnica y diaria
soluciones con las que vivimos constantemente, siempre prometiendo que arreglaremos el desastre cuando
Ten un poco más de tiempo. Pero ese momento nunca llega.

Es alarmante que nuestros artefactos más frágiles respalden nuestros ingresos más importantes.
sistemas generadores o nuestros proyectos más críticos. En otras palabras, los sistemas más propensos a
El fracaso también es nuestro más importante y está en el epicentro de nuestros cambios más urgentes.
Cuando estos cambios fallan, ponen en peligro nuestras promesas organizacionales más importantes, como
como disponibilidad para los clientes, objetivos de ingresos, seguridad de los datos del cliente, información financiera precisa
informes, y así sucesivamente.

El segundo acto comienza cuando alguien tiene que compensar la última promesa rota:
podría ser un gerente de producto que promete una característica más grande y audaz para deslumbrar a los clientes con o
un ejecutivo de negocios que establece un objetivo de ingresos aún mayor. Entonces, ajeno a lo que
la tecnología puede o no puede hacer, o qué factores llevaron a perder nuestro compromiso anterior, ellos
comprometer a la organización tecnológica a cumplir esta nueva promesa.

Como resultado, el desarrollo tiene la tarea de otro proyecto urgente que inevitablemente requiere
resolviendo nuevos desafíos técnicos y atajos para cumplir con la fecha de lanzamiento prometida,
agregando aún más a nuestra deuda técnica, hecha, por supuesto, con la promesa de que arreglaremos cualquier
problemas resultantes cuando tenemos un poco más de tiempo.

Esto prepara el escenario para el tercer y último acto, donde todo se vuelve un poco más
difícil, poco a poco: todos se ponen un poco más ocupados, el trabajo lleva un poco más de tiempo,

Página 24

las comunicaciones se vuelven un poco más lentas y las colas de trabajo se alargan un poco más. Nuestro trabajo
se vuelve más estrechamente acoplado, acciones más pequeñas causan fallas más grandes, y nos volvemos más
temeroso y menos tolerante de hacer cambios. El trabajo requiere más comunicación,
coordinación y aprobaciones; los equipos deben esperar un poco más por su trabajo dependiente
para terminar y nuestra calidad sigue empeorando. Las ruedas comienzan a moler más lentamente y

requieren más esfuerzo para seguir girando. VerApéndice 3.

Aunque es difícil de ver en el momento, la espiral descendente es obvia cuando uno toma
Un paso atrás. Notamos que las implementaciones de código de producción tardan cada vez más en
completa, pasando de minutos a horas a días a semanas. Y peor, el despliegue
los resultados se han vuelto aún más problemáticos, lo que resulta en un aumento constante
Número de interrupciones que afectan al cliente que requieren más heroicidad y extinción de incendios en
Operaciones, privándolos aún más de su capacidad de pagar deudas técnicas.

Como resultado, nuestros ciclos de entrega de productos continúan avanzando más y más lentamente, menos proyectos
se emprenden, y los que son, son menos ambiciosos. Además, los comentarios sobre
El trabajo de todos se vuelve más lento y más débil, especialmente las señales de retroalimentación de nuestro
clientes. Y, independientemente de lo que intentemos, las cosas parecen empeorar: ya no podemos
para responder rápidamente a nuestro cambiante panorama competitivo, ni podemos proporcionar
Servicio estable y confiable a nuestros clientes. Como resultado, finalmente perdemos en el mercado.

Una y otra vez, aprendemos que cuando falla la TI, toda la organización falla. Como Steven J.
Spear señaló en su libro *The High-Velocity Edge* , si los daños "se desarrollan lentamente como
una enfermedad debilitante "o rápidamente" como un choque de fuego ... la destrucción puede ser igual de completa ".

¿POR QUÉ ESTE ESPIRAL HACIA ABAJO SUCEDE EN TODAS PARTES?

Durante más de una década, los autores de este libro han observado que esta espiral destructiva ocurre en
innumerables organizaciones de todos los tipos y tamaños. Entendemos mejor que nunca por qué esto
se produce una espiral descendente y por qué requiere mitigar los principios de DevOps. Primero como
descrito anteriormente, cada organización de TI tiene dos objetivos opuestos, y segundo, cada
La compañía es una compañía de tecnología, lo sepan o no.

Como Christopher Little, un ejecutivo de software y uno de los primeros cronistas de DevOps,
dijo: "Cada empresa es una empresa de tecnología, independientemente de qué negocio piensan
están dentro. Un banco es solo una compañía de TI con una licencia bancaria ". §

Para convencernos de que este es el caso, tenga en cuenta que la gran mayoría del capital
los proyectos tienen cierta dependencia de TI. Como dice el dicho: "Es prácticamente imposible hacer
cualquier decisión comercial que no resulte en al menos un cambio de TI ".

En el contexto empresarial y financiero, los proyectos son críticos porque sirven como el principal
mecanismo de cambio dentro de las organizaciones. Los proyectos son típicamente lo que la gerencia necesita
aprobar, presupuestar y rendir cuentas; por lo tanto, son el mecanismo que
lograr los objetivos y aspiraciones de la organización, ya sea para crecer o incluso reducirse.¶

Los proyectos generalmente se financian a través de gastos de capital (es decir, fábricas, equipos y
proyectos importantes, y los gastos se capitalizan cuando se espera que la recuperación de la inversión demore años),
de los cuales el 50% ahora está relacionado con la tecnología. Esto es incluso cierto en verticales de la industria de "baja tecnología"
con el gasto histórico más bajo en tecnología, como energía, metal, recursos
extracción, automotriz y construcción. En otras palabras, los líderes empresariales son mucho más

depender de la gestión efectiva de TI para lograr sus objetivos de lo que ellos pensar.** **

LOS COSTOS: HUMANOS Y ECONOMICOS

Cuando las personas están atrapadas en esta espiral descendente durante años, especialmente aquellas que están aguas abajo del desarrollo, a menudo se sienten atrapados en un sistema que ordena fallos y los deja impotentes para cambiar los resultados. Esta impotencia a menudo es seguida por agotamiento, con los sentimientos asociados de fatiga, cinismo e incluso desesperanza y desesperación.

Muchos psicólogos afirman que crear sistemas que causan sentimientos de impotencia es uno de las cosas más perjudiciales que podemos hacer a otros seres humanos: privamos a otras personas de su capacidad para controlar sus propios resultados e incluso crear una cultura donde las personas están miedo de hacer lo correcto por temor a castigo, fracaso o poner en peligro su sustento. Esto puede crear las condiciones de *impotencia aprendida* , donde las personas se vuelven reacio o incapaz de actuar de una manera que evite el mismo problema en el futuro.

Para nuestros empleados, significa largas horas, trabajar los fines de semana y una disminución de la calidad de vida, no solo para el empleado, sino para todos los que dependen de ellos, incluidos familiares y amigos. No es sorprendente que cuando esto ocurre, perdemos a nuestra mejor gente (excepto aquellos que sienten que no pueden irse, debido a un sentido del deber u obligación).

Además del sufrimiento humano que viene con la forma actual de trabajar, el El costo de oportunidad del valor que podríamos estar creando es asombroso: los autores creen que estamos perdiendo aproximadamente $ 2.6 billones de creación de valor por año, que es, Al momento de escribir este artículo, equivalente a la producción económica anual de Francia, el sexto La economía más grande del mundo.

Considere el siguiente cálculo: tanto IDC como Gartner estimaron que en 2011, aproximadamente el 5% del producto interno bruto mundial ($ 3,1 billones) se gastó en TI (hardware, servicios y telecomunicaciones). Si estimamos que el 50% de esos $ 3.1 billones se gastó en costos de operación y mantenimiento de los sistemas existentes, y que un tercio de ese 50% se gastó en trabajos o reprocesos urgentes y no planificados, se desperdiciaron aproximadamente $ 520 mil millones.

Si la adopción de DevOps nos podría permitir, a través de una mejor gestión y un aumento excelencia operativa, reducir a la mitad ese desperdicio y volver a desplegar ese potencial humano en algo que es cinco veces el valor (una propuesta modesta), podríamos crear $ 2.6 billones de valor por año.

LA ÉTICA DE LOS DEVOPS: HAY UNA MEJOR MANERA

En las secciones anteriores, describimos los problemas y las consecuencias negativas de status quo debido al conflicto central, crónico, de la incapacidad para lograr la organización objetivos, por el daño que infligimos a otros seres humanos. Al resolver estos problemas, DevOps asombrosamente nos permite mejorar simultáneamente el desempeño organizacional, lograr Los objetivos de todos los diversos roles de tecnología funcional (por ejemplo, desarrollo, control de calidad, TI Operaciones, Infosec), y mejorar la condición humana.

Esta combinación emocionante y rara puede explicar por qué DevOps ha generado tanto entusiasmo y entusiasmo en tantos en tan poco tiempo, incluidos los líderes tecnológicos,

ingenieros y gran parte del ecosistema de software en el que residimos.

ROMPIENDO LA ESPIRAL HACIA ABAJO CON DEVOPS

Idealmente, pequeños equipos de desarrolladores implementan independientemente sus características, validan sus corrección en entornos similares a la producción, y tener su código implementado en producción de forma rápida y segura. Las implementaciones de código son rutinarias y predecibles. En lugar de comenzando implementaciones a la medianoche del viernes y pasando todo el fin de semana trabajando para completar ellos, las implementaciones ocurren durante el día hábil cuando todos ya están en el oficina y sin que nuestros clientes se den cuenta, excepto cuando ven nuevas funciones y correcciones de errores que los deleitan. Y, mediante la implementación de código en medio de la jornada laboral, para el primera vez en décadas, Operaciones de TI está trabajando durante el horario comercial normal como todos más.

Al crear ciclos de retroalimentación rápidos en cada paso del proceso, todos pueden ver de inmediato Los efectos de sus acciones. Cada vez que se comprometan cambios en el control de versiones, rápido Las pruebas automatizadas se ejecutan en entornos similares a la producción, lo que garantiza continuamente que el código y los entornos funcionan según lo diseñado y siempre están en un lugar seguro y desplegable estado.

Las pruebas automatizadas ayudan a los desarrolladores a descubrir sus errores rápidamente (generalmente dentro de minutos), que permite soluciones más rápidas, así como un aprendizaje genuino, aprendizaje que es imposible cuando se descubren errores seis meses después durante las pruebas de integración, cuando Los recuerdos y el vínculo entre causa y efecto se han desvanecido por mucho tiempo. En lugar de acumularse deuda técnica, los problemas se resuelven a medida que se encuentran, movilizando a toda la organización si necesario, porque los objetivos globales superan los objetivos locales.

La telemetría de producción generalizada en nuestros entornos de código y producción garantiza que Los problemas se detectan y corrigen rápidamente, lo que confirma que todo funciona como previsto y los clientes obtienen valor del software que creamos.

En este escenario, todos se sienten productivos: la arquitectura permite que pequeños equipos trabajen Desacoplado de forma segura y arquitectónica del trabajo de otros equipos que utilizan el autoservicio plataformas que aprovechan la experiencia colectiva de Operaciones y Seguridad de la Información. En lugar de que todos esperen todo el tiempo, con grandes cantidades de retrabajo, trabajo urgente, equipos trabaje de manera independiente y productiva en pequeños lotes, entregando rápida y frecuentemente Nuevo valor para los clientes.

Incluso los lanzamientos de productos y funciones de alto perfil se vuelven rutinarios mediante el lanzamiento oscuro técnicas Mucho antes de la fecha de lanzamiento, colocamos todo el código requerido para la función en producción, invisible para todos, excepto para empleados internos y pequeños grupos de usuarios reales, lo que nos permite probar y desarrollar la función hasta que alcance el objetivo comercial deseado.

Y, en lugar de combatir incendios durante días o semanas para que la nueva funcionalidad funcione, nosotros simplemente cambie una función de alternancia o configuración. Este pequeño cambio hace que el nuevo característica visible para segmentos cada vez más grandes de clientes, retrocediendo automáticamente si Algo va mal. Como resultado, nuestros lanzamientos son controlados, predecibles, reversibles y

Estrés bajo.

No solo las versiones están más tranquilas: se están encontrando todo tipo de problemas y
arreglado temprano, cuando son más pequeños, más baratos y más fáciles de corregir. Con cada solución, también

generar aprendizajes organizacionales, lo que nos permite evitar que el problema se repita y
permitiéndonos detectar y corregir problemas similares más rápido en el futuro.

Además, todos están aprendiendo constantemente, fomentando una cultura basada en hipótesis donde
El método científico se utiliza para garantizar que nada se dé por sentado: no hacemos nada sin
medir y tratar el desarrollo de productos y la mejora de procesos como experimentos.

Debido a que valoramos el tiempo de todos, no pasamos años creando características que nuestro
los clientes no quieren, implementando código que no funciona o arreglando algo que no funciona
En realidad la causa de nuestro problema.

Debido a que nos preocupamos por lograr los objetivos, creamos equipos a largo plazo que son responsables de
reuniéndolos En lugar de equipos de proyecto donde los desarrolladores son reasignados y barajados
después de cada lanzamiento, sin recibir comentarios sobre su trabajo, mantenemos los equipos intactos
pueden seguir iterando y mejorando, utilizando esos aprendizajes para lograr mejor sus objetivos.
Esto es igualmente cierto para los equipos de productos que están resolviendo problemas para nuestros
clientes, así como nuestros equipos de plataforma interna que ayudan a otros equipos a ser más
productivo, seguro y seguro.

En lugar de una cultura del miedo, tenemos una cultura colaborativa de alta confianza, donde las personas están
recompensado por tomar riesgos. Son capaces de hablar sin temor sobre problemas en lugar de
ocultándolos o poniéndolos en segundo plano, después de todo, debemos ver los problemas en orden
para resolverlos

Y, debido a que todos poseen la calidad de su trabajo, todos construyen de manera automatizada
prueba en su trabajo diario y utiliza revisiones por pares para ganar confianza en que los problemas son
abordado mucho antes de que puedan afectar a un cliente. Estos procesos mitigan el riesgo, como
en oposición a las aprobaciones de autoridades distantes, lo que nos permite entregar valor de manera rápida y confiable,
y de forma segura, incluso demostrando a auditores escépticos que tenemos un sistema efectivo de
controles internos.

Y cuando algo sale mal, llevamos a cabo *autopsias sin culpa* , para no castigar
cualquiera, pero para comprender mejor qué causó el accidente y cómo prevenirlo. Esta
El ritual refuerza nuestra cultura de aprendizaje. También realizamos conferencias tecnológicas internas para
elevar nuestras habilidades y garantizar que todos estén siempre enseñando y aprendiendo.

Debido a que nos importa la calidad, incluso inyectamos fallas en nuestro entorno de producción, por lo que
puede aprender cómo falla nuestro sistema de manera planificada. Realizamos ejercicios planificados para
practicar fallas a gran escala, matar procesos al azar y calcular servidores en producción,
e inyectar latencias de red y otros actos nefastos para garantizar que crezcamos cada vez más

elástico. Al hacer esto, permitimos una mejor capacidad de recuperación, así como el aprendizaje organizacional y mejora.

En este mundo, todos son dueños de su trabajo, independientemente de su papel en el organización tecnológica Confían en que su trabajo es importante y es significativo contribuyendo a los objetivos organizacionales, probados por su ambiente de trabajo de bajo estrés y El éxito de su organización en el mercado. Su prueba es que la organización es de hecho ganando en el mercado.

EL VALOR EMPRESARIAL DE LOS DEVOPS

Página 28

Tenemos evidencia decisiva del valor comercial de DevOps. Desde 2013 hasta 2016, como parte del *informe State Of DevOps* de Puppet Labs , al que los autores Jez Humble y Gene Kim contribuido, recopilamos datos de más de veinticinco mil profesionales de la tecnología, con el objetivo de comprender mejor la salud y los hábitos de las organizaciones en todas las etapas de Adopción de DevOps.

La primera sorpresa que revelaron estos datos fue la cantidad de organizaciones de alto rendimiento que utilizan Las prácticas de DevOps superaron a sus pares de bajo rendimiento en lo siguiente áreas:

- Métricas de rendimiento
- Implementaciones de código y cambio (treinta veces más frecuentes)
- Codifique y cambie el tiempo de implementación (doscientas veces más rápido)
- Métricas de confiabilidad
- Implementaciones de producción (tasa de éxito de cambio sesenta veces mayor)
- Tiempo medio para restaurar el servicio (168 veces más rápido)
- Métricas de desempeño organizacional
- Objetivos de productividad, participación de mercado y rentabilidad (dos veces más probabilidades de exceder)
- Crecimiento de la capitalización de mercado (50% más en tres años)

En otras palabras, los de alto rendimiento fueron más ágiles y más confiables, proporcionando La evidencia empírica de que DevOps nos permite romper el conflicto central y crónico. Alto los artistas implementaron el código treinta veces más frecuentemente y el tiempo requerido para pasar de El "código comprometido" para "ejecutarse con éxito en la producción" fue doscientas veces más rápido: los de alto rendimiento tenían tiempos de entrega medidos en minutos u horas, mientras que los de bajo rendimiento tenían

plazos de entrega medidos en semanas, meses o incluso trimestres.

Además, los de alto rendimiento tenían el doble de probabilidades de superar la rentabilidad, la cuota de mercado, y objetivos de productividad. Y, para aquellas organizaciones que proporcionaron un símbolo de cotización bursátil, nosotros descubrió que los de alto rendimiento tuvieron un crecimiento de capitalización de mercado un 50% mayor en tres años. También tuvieron una mayor satisfacción laboral de los empleados, menores tasas de agotamiento de los empleados y su los empleados tenían 2,2 veces más probabilidades de recomendar su organización a sus amigos como un gran lugar para trabajar.†† Los de alto rendimiento también tuvieron mejores resultados de seguridad de la información. Por integrando objetivos de seguridad en todas las etapas de los procesos de desarrollo y operaciones, dedicaron un 50% menos de tiempo a solucionar problemas de seguridad.

DEVOPS AYUDA A LA PRODUCTIVIDAD DEL DESARROLLADOR DE ESCALA

Cuando aumentamos el número de desarrolladores, la productividad individual del desarrollador a menudo disminuye significativamente debido a la comunicación, integración y sobrecarga de pruebas. Esto es destacado en el famoso libro de Frederick Brook *, The Mythical Man-Month* , donde explica que cuando los proyectos llegan tarde, agregar más desarrolladores no solo disminuye individualmente productividad del desarrollador pero también disminuye la productividad general.

Por otro lado, DevOps nos muestra que cuando tenemos la arquitectura correcta, la correcta prácticas técnicas y las normas culturales correctas, pequeños equipos de desarrolladores pueden Desarrolle, integre, pruebe e implemente cambios de forma rápida, segura e independiente en producción. Como observó Randy Shoup, anteriormente director de ingeniería de Google, Las organizaciones que utilizan DevOps "tienen miles de desarrolladores, pero su arquitectura y las prácticas permiten que los equipos pequeños sigan siendo increíblemente productivos, como si fueran una startup ".

El *Informe del estado de DevOps de 2015* examinó no solo "implementaciones por día" sino también "implementaciones por día por desarrollador ". Presumimos que los de alto rendimiento podrían escalar sus número de implementaciones a medida que creció el tamaño del equipo.

Figura 1. *Implementaciones / día vs. número de desarrolladores (Fuente: Puppet Labs,* 2015 State Of Informe DevOps *.)* [‡‡](#)

De hecho, esto es lo que encontramos. La Figura 1 muestra que en los de bajo rendimiento, las implementaciones por día por el desarrollador baja a medida que aumenta el tamaño del equipo, se mantiene constante para los de mediano rendimiento y aumenta linealmente para artistas de alto rendimiento.

En otras palabras, las organizaciones que adoptan DevOps pueden aumentar linealmente el número de se implementa por día a medida que aumentan su número de desarrolladores, al igual que Google, Amazon y Netflix lo ha hecho.[§§](#)

LA UNIVERSALIDAD DE LA SOLUCIÓN

Uno de los libros más influyentes en el movimiento de manufactura esbelta es *The Goal: A Proceso de mejora continua* escrito por el Dr. Eliyahu M. Goldratt en 1984. Influyó Una generación entera de gerentes de planta profesionales en todo el mundo. Era una novela sobre un gerente de planta que tuvo que arreglar sus problemas de costos y fecha de vencimiento del producto en noventa días, de lo contrario su planta se cerraría.

Más adelante en su carrera, el Dr. Goldratt describió las cartas que recibió en respuesta a *The Goal.* Estas cartas normalmente decían: "Obviamente te has estado escondiendo en nuestra fábrica, porque has descrito mi vida [como gerente de planta] exactamente ... "Lo más importante, estas cartas mostró que las personas pudieron replicar los avances en el rendimiento que fueron descrito en el libro en sus propios entornos de trabajo.

El Proyecto Phoenix: una novela sobre TI, DevOps y cómo ayudar a su empresa a ganar , escrito por Gene Kim, Kevin Behr y George Spafford en 2013, se inspiró en el modelo *The Objetivo* . Es una novela que sigue a un líder de TI que enfrenta todos los problemas típicos

endémico en las organizaciones de TI: un proyecto con exceso de presupuesto y retraso que debe llegar a mercado para que la empresa pueda sobrevivir. Experimenta despliegues catastróficos; problemas de disponibilidad, seguridad y cumplimiento; Etcétera. En definitiva, él y su el equipo usa los principios y prácticas de DevOps para superar esos desafíos, ayudando a sus organización gana en el mercado. Además, la novela muestra cómo practican DevOps mejoró el entorno laboral para el equipo, creando menos estrés y más satisfacción debido a una mayor participación del profesional en todo el proceso.

Al igual que con *The Goal* , existe una tremenda evidencia de la universalidad de los problemas y soluciones descritas en *The Phoenix Project.* Considere algunas de las declaraciones que se encuentran en el Amazon comenta: "Me encuentro relacionado con los personajes de *The Phoenix Project* ... he probablemente conocí a la mayoría de ellos en el transcurso de mi carrera "" Si alguna vez has trabajado en alguna aspecto de TI, DevOps o Infosec definitivamente podrá relacionarse con este libro "o "No hay un personaje en *The Phoenix Project* que no identifique conmigo mismo o alguien que conozco en la vida real ... sin mencionar los problemas que enfrentan y superan aquellos caracteres."

En el resto de este libro, describiremos cómo replicar la transformación.
descrito en *The Phoenix Project* , también proporciona muchos estudios de casos de cómo otros
Las organizaciones han utilizado los principios y prácticas de DevOps para replicar esos resultados.

EL MANUAL DEVOPS: UNA GUÍA ESENCIAL

El propósito del *Manual de DevOps* es brindarle la teoría, los principios y las prácticas.
necesita iniciar con éxito su iniciativa DevOps y lograr los resultados deseados.
Esta guía se basa en décadas de teoría de gestión sólida, estudio de alto rendimiento.
organizaciones tecnológicas, trabajo que hemos realizado para ayudar a las organizaciones a transformarse, y
investigación que valida la efectividad de las prácticas DevOps prescritas. Tanto como
entrevistas con expertos en la materia relevantes y análisis de casi cien casos
estudios presentados en la Cumbre de DevOps Enterprise.

Dividido en seis partes, este libro cubre las teorías y principios de DevOps utilizando los Tres
Ways, una visión específica de la teoría subyacente introducida originalmente en *The Phoenix*
Proyecto . *El manual de DevOps* es para todos los que realizan o influyen en el trabajo en el
flujo de valor tecnológico (que generalmente incluye gestión de productos, desarrollo,
Control de calidad, operaciones de TI y seguridad de la información), así como para negocios y marketing
liderazgo, donde se originan la mayoría de las iniciativas tecnológicas.

No se espera que el lector tenga un amplio conocimiento de ninguno de estos dominios, o de
DevOps, Agile, ITIL, Lean o mejora de procesos. Cada uno de estos temas se presenta y
explicado en el libro cuando se hace necesario.

Nuestra intención es crear un conocimiento práctico de los conceptos críticos en cada uno de estos
dominios, tanto para servir como una cartilla e introducir el lenguaje necesario para ayudar
los profesionales trabajan con todos sus pares en todo el flujo de valor de TI y enmarcan
objetivos compartidos.

Este libro será de valor para los líderes empresariales y las partes interesadas que son cada vez más dependientes
sobre la organización tecnológica para el logro de sus objetivos.

Además, este libro está destinado a lectores cuyas organizaciones podrían no ser
experimentando todos los problemas descritos en el libro (por ejemplo, largos plazos de implementación o
despliegues dolorosos). Incluso los lectores en esta posición afortunada se beneficiarán de
comprender los principios de DevOps, especialmente aquellos relacionados con objetivos compartidos, comentarios y
aprendizaje continuo

En la Parte I, presentamos una breve historia de DevOps e introducimos la teoría subyacente y
temas clave de cuerpos de conocimiento relevantes que abarcan décadas. Luego presentamos el
principios de alto nivel de las tres formas: flujo, retroalimentación y aprendizaje continuo y
Experimentación

La Parte II describe cómo y dónde comenzar, y presenta conceptos tales como flujos de valor, principios y patrones de diseño organizacional, patrones de adopción organizacional y caso estudios.

La Parte III describe cómo acelerar Flow al construir los cimientos de nuestro despliegue pipeline: permitiendo pruebas automatizadas rápidas y efectivas, integración continua, continua entrega y arquitectura para lanzamientos de bajo riesgo.

La Parte IV discute cómo acelerar y amplificar la retroalimentación creando una producción efectiva telemetría para ver y resolver problemas, anticipar mejor los problemas y alcanzar objetivos, permitir retroalimentación para que Dev y Ops puedan implementar cambios de manera segura, integrar pruebas A / B en nuestro trabajo diario, y crear procesos de revisión y coordinación para aumentar la calidad de nuestro trabajo.

La Parte V describe cómo aceleramos el aprendizaje continuo al establecer una cultura justa, convirtiendo descubrimientos locales en mejoras globales, y reservando adecuadamente el tiempo para Crear aprendizaje organizacional y mejoras.

Finalmente, en la Parte VI describimos cómo integrar adecuadamente la seguridad y el cumplimiento en nuestro trabajo diario, integrando controles de seguridad preventivos en código fuente compartido repositorios y servicios, integrando la seguridad en nuestro canal de implementación, mejorando telemetría para permitir una mejor detección y recuperación, proteger la canalización de implementación, y logrando objetivos de gestión del cambio.

Al codificar estas prácticas, esperamos acelerar la adopción de prácticas DevOps, aumentar el éxito de las iniciativas DevOps y reducir la energía de activación requerida para Transformaciones de DevOps.

† Esta es solo una pequeña muestra de los problemas encontrados en las organizaciones de TI típicas.

‡ En el ámbito de la fabricación, existía un conflicto crónico similar: la necesidad de garantizar simultáneamente los envíos a tiempo a los clientes y controlar los costos. En el Apéndice 2 se describe cómo se rompió este conflicto crónico central.

§ En 2013, el banco europeo HSBC empleó más desarrolladores de software que Google.

¶ Por ahora, suspendamos la discusión sobre si el software debe ser financiado como un "proyecto" o un "producto". Esto se discute más adelante en el libro.

** Por ejemplo, el Dr. Vernon Richardson y sus colegas publicaron este sorprendente hallazgo. Estudiaron las presentaciones de 10-K SEC de 184 corporaciones públicas y los dividió en tres grupos: A) empresas con debilidades materiales con deficiencias relacionadas con TI, B) empresas con debilidades materiales sin problemas relacionados con TI deficiencias, y C) "empresas limpias" sin debilidades materiales. Las empresas del Grupo A registraron una rotación del CEO ocho veces mayor que el Grupo C, y hubo cuatro veces mayor rotación de CFO en el Grupo A que en el Grupo C. Claramente, TI puede importar mucho más de lo que normalmente pensamos.

†† Medido por la puntuación neta del promotor del empleado (eNPS). Este es un hallazgo significativo, ya que la investigación ha demostrado que "las empresas con trabajadores altamente comprometidos Los ingresos crecieron dos veces y media más que aquellos con bajos niveles de participación. Y acciones [cotizadas en bolsa] de compañías con un trabajo de alta confianza el medio ambiente superó los índices del mercado en un factor de tres desde 1997 hasta 2011 ".

‡‡ Solo se muestran las organizaciones que se implementan al menos una vez al día.

§§ Otro ejemplo más extremo es Amazon. En 2011, Amazon realizaba aproximadamente siete mil implementaciones por día. Para 2015, estaban realizando 130,000 implementaciones por día.

Parte

Introducción

En la Parte I del *Manual de DevOps* , exploraremos cómo la convergencia de varios movimientos importantes en gestión y tecnología prepararon el escenario para el Movimiento DevOps. Describimos flujos de valor, cómo DevOps es el resultado de aplicar los principios Lean a flujo de valor tecnológico y las tres formas: flujo, Retroalimentación, y aprendizaje y experimentación continua.

Los enfoques principales dentro de estos capítulos incluyen:

Los principios de Flow, que aceleran la entrega. de trabajo desde Desarrollo a Operaciones a nuestro clientes

Los principios de Feedback, que nos permiten crear sistemas de trabajo cada vez más seguros

Los principios del aprendizaje continuo y La experimentación fomenta una cultura de alta confianza y un enfoque científico para la mejora organizacional asumir riesgos como parte de nuestro trabajo diario

UNA BREVE HISTORIA

DevOps y sus resultados técnicos, arquitectónicos y
las prácticas culturales representan una convergencia de muchos
Movimientos filosóficos y de gestión. Mientras que muchos
las organizaciones han desarrollado estos principios
independientemente, entendiendo que DevOps resultó
de un amplio golpe de movimientos, un fenómeno
descrito por John Willis (uno de los coautores de este
libro) como la "convergencia de DevOps", muestra un
Increíble progresión de pensamiento e improbable
conexiones Hay décadas de lecciones aprendidas de
fabricación, organización de alta confiabilidad, alta confianza
modelos de gestión y otros que nos han traído a
las prácticas de DevOps que conocemos hoy.

DevOps es el resultado de aplicar el más confiable
principios del dominio de la fabricación física
y liderazgo en el flujo de valor de TI. DevOps confía en
cuerpos de conocimiento de Lean, Theory of Restraints,
el sistema de producción de Toyota, ingeniería de resiliencia,
organizaciones de aprendizaje, cultura de seguridad, factores humanos,
y muchos otros. Otros contextos valiosos que DevOps
se basa en incluir culturas de gestión de alta confianza,
liderazgo de servicio y cambio organizacional
administración. El resultado es una calidad de clase mundial,
fiabilidad, estabilidad y seguridad a un costo cada vez menor y
esfuerzo; y flujo acelerado y fiabilidad en todo
el flujo de valor tecnológico, incluido el producto
Gestión, desarrollo, control de calidad, operaciones de TI y
Infosec.

Si bien la base de DevOps puede verse como

derivado de Lean, la teoría de las restricciones y el

Movimiento Toyota Kata, muchos también ven a DevOps como el
continuación lógica del viaje de software ágil que
comenzó en 2001.

EL MOVIMIENTO LEAN

Técnicas como la asignación de flujo de valor, Kanban
Se codificaron tableros y mantenimiento productivo total
para el sistema de producción de Toyota en la década de 1980. En 1997,
el Lean Enterprise Institute comenzó a investigar
aplicaciones de Lean a otras corrientes de valor, como el
industria de servicios y asistencia sanitaria.

Dos de los principios fundamentales de Lean incluyen la creencia profundamente arraigada
ese *tiempo de fabricación* requerido para convertir crudo
materiales en productos terminados fue el mejor predictor de
calidad, satisfacción del cliente y felicidad de los empleados,
y que uno de los mejores predictores de plazos de entrega cortos
Fueron pequeños lotes de trabajo.

Los principios Lean se centran en cómo crear valor para el
cliente a través del pensamiento de sistemas creando
constancia de propósito, abrazando el pensamiento científico,
creando flujo y atracción (versus empuje), asegurando calidad en
la fuente, liderando con humildad y respetando cada
individual.

El manifiesto ágil

El Manifiesto Ágil fue creado en 2001 por diecisiete de Los pensadores fideres en desarrollo de software. Ellos quería crear un conjunto ligero de valores y principios contra el desarrollo de software pesado

procesos como el desarrollo en cascada, y metodologías como el proceso racional unificado.

Un principio clave era "entregar software de trabajo con frecuencia, de un par de semanas a un par de meses, con preferencia a la escala de tiempo más corta ", enfatizando El deseo de lotes pequeños, lanzamientos incrementales en lugar de grandes lanzamientos en cascada. Otros principios hizo hincapié en la necesidad de equipos pequeños y motivados, trabajando en un modelo de gestión de alta confianza.

Agile es acreditado por aumentar dramáticamente el productividad de muchas organizaciones de desarrollo. Y Curiosamente, muchos de los momentos clave en DevOps la historia también ocurrió dentro de la comunidad ágil o en Conferencias ágiles, como se describe a continuación.

INFRAESTRUCTURA ÁGIL Y VELOCIDAD MOVIMIENTO

En la conferencia Agile 2008 en Toronto, Canadá, Patrick Debois y Andrew Schafer sostuvieron un "pájaros de un sesión "pluma" sobre la aplicación de principios ágiles a infraestructura en lugar de código de aplicación. A pesar de que

fueron las únicas personas que aparecieron, rápidamente ganó seguidores de pensadores de ideas afines, incluyendo autor John Willis.

Más tarde, en la conferencia Velocity 2009, John Allspaw y Paul Hammond dio el seminal "10 despliegues por día: Dev y Ops Cooperación en la presentación de Flickr ", donde describieron cómo crearon objetivos compartidos entre Dev y Ops y prácticas de integración continua utilizadas hacer que la implementación sea parte del trabajo diario de todos.

Según las cuentas de primera mano, todos los asistentes al presentación de inmediato supo que estaban en el presencia de algo profundo e histórico significado.

Patrick Debois no estaba allí, pero estaba muy emocionado por La idea de Allspaw y Hammond de que él creó el primer DevOpsDays en Gante, Bélgica, (donde vivía) en 2009. Allí se acuñó el término "DevOps".

EL MOVIMIENTO CONTINUO DE ENTREGA

Sobre la base de la disciplina de desarrollo continuo construcción, prueba e integración, Jez Humble y David Farley extendió el concepto a *la entrega continua* , que definió el papel de una "tubería de implementación" para asegúrese de que el código y la infraestructura estén siempre en un estado desplegable, y que todo el código registrado en el tronco puede desplegarse de manera segura en producción. Esta idea fue

presentado por primera vez en la conferencia Agile 2006, y fue
También desarrollado independientemente en 2009 por Tim Fitz en un
publicación de blog en su sitio web titulada "Continuo
Despliegue."¶¶

TOYOTA KATA

En 2009, Mike Rother escribió *Toyota Kata: Gestión*
Personas para Mejora, Adaptabilidad y Superior
Resultados , que enmarcaron su viaje de veinte años a
Comprender y codificar el Sistema de Producción Toyota.
Había sido uno de los estudiantes de posgrado que voló con
Los ejecutivos de GM visitarán las plantas de Toyota y ayudaron a desarrollar
el kit de herramientas Lean, pero estaba perplejo cuando ninguno de los

Las empresas que adoptaron estas prácticas replicaron el nivel
de rendimiento observado en las plantas de Toyota.

Llegó a la conclusión de que la comunidad Lean era la que más extrañaba
importante práctica de todas, a la que llamó
mejora kata . Él explica que cada organización
tiene rutinas de trabajo, y la mejora que requiere kata
creando estructura para la práctica diaria y habitual de
trabajo de mejora, porque la práctica diaria es lo que
Mejora los resultados. El ciclo constante de establecimiento
estados futuros deseados, estableciendo resultados objetivo semanales,
y la mejora continua del trabajo diario es lo que
mejora guiada en Toyota.

Lo anterior describe la historia de DevOps y relevante movimientos sobre los que se basa. En todo el resto de Parte I, observamos los flujos de valor, cómo los principios Lean pueden ser aplicado a la corriente de valor de la tecnología, y los Tres Formas de flujo, retroalimentación y aprendizaje continuo y Experimentación.

¶¶ DevOps también extiende y desarrolla las prácticas de *infraestructura como código* , que fue promovido por el Dr. Mark Burgess, Luke Kanies y Adam Jacob. En infraestructura como código, el El trabajo de Operaciones es automatizado y tratado como un código de aplicación, de modo que Las prácticas de desarrollo se pueden aplicar a todo el flujo de desarrollo. Esto además permitió flujo de implementación rápido, incluida la integración continua (iniciada por Grady Booch y integrado como una de las 12 prácticas clave de programación extrema), entrega continua (promovido por Jez Humble y David Farley), y despliegue continuo (promovido por Etsy, Wealthfront y el trabajo de Eric Ries en IMVU).

1 Entrega y el Trágil, continuo

En este capítulo, una introducción a la base
Se presenta la teoría de Lean Manufacturing, así como la
Tres formas, los principios a partir de los cuales
se pueden derivar comportamientos observados de DevOps.

Nuestro enfoque aquí es principalmente en teoría y principios,
describiendo muchas décadas de lecciones aprendidas de
fabricación, organizaciones de alta confiabilidad, alta confianza
modelos de gestión y otros, desde los cuales DevOps
Se han derivado prácticas. El concreto resultante
principios y patrones, y su aplicación práctica a
el flujo de valor tecnológico, se presentan en el
capítulos restantes del libro.

EL VALOR DE FABRICACIÓN CORRIENTE

Uno de los conceptos fundamentales en Lean es el *valor.*
transmitir . Lo definiremos primero en el contexto de
fabricación y luego extrapolar cómo se aplica a
DevOps y el flujo de valor tecnológico.

Karen Martin y Mike Osterling definen flujo de valor en
su libro *Value Stream Mapping: Cómo visualizar*
Trabajar y alinear el liderazgo para la organización
Transformación como "la secuencia de actividades y

la organización se compromete a entregar a un cliente solicitud "o" la secuencia de actividades necesarias para diseñar, producir y entregar un bien o servicio a un cliente, incluidos los flujos duales de información y material."

En las operaciones de fabricación, el flujo de valor es a menudo fácil de ver y observar: comienza cuando un pedido del cliente se recibe y las materias primas se liberan en el piso de planta. Para permitir tiempos de entrega rápidos y predecibles en cualquier flujo de valor, generalmente hay un enfoque implacable en creando un flujo de trabajo uniforme y uniforme, utilizando técnicas como lotes pequeños, que reducen el trabajo en proceso (WIP), evitando el reprocesamiento para garantizar que no pasar defectos a los centros de trabajo aguas abajo, y constantemente optimizando nuestro sistema hacia nuestros objetivos globales.

La corriente de valor tecnológico

Los mismos principios y patrones que permiten el rápido el flujo de trabajo en procesos físicos es igualmente aplicable al trabajo tecnológico (y, para el caso, para todos el trabajo del conocimiento). En DevOps, generalmente definimos nuestro flujo de valor tecnológico como el proceso requerido para convertir una hipótesis de negocio en una tecnología habilitada servicio que entrega valor al cliente.

La entrada a nuestro proceso es la formulación de un negocio.
objetivo, concepto, idea o hipótesis, y comienza cuando
aceptamos el trabajo en Desarrollo, agregándolo a nuestro
compromiso de trabajo atrasado.

A partir de ahí, los equipos de desarrollo que siguen un típico
El proceso ágil o iterativo probablemente transformará esa idea
en historias de usuarios y algún tipo de especificación de características,
que luego se implementa en código en la aplicación
o servicio en construcción. El código se registra en el
repositorio de control de versiones, donde cada cambio es
integrado y probado con el resto del software
sistema.

Porque el valor se crea solo cuando nuestros servicios son
funcionando en producción, debemos asegurarnos de que no estamos
solo entregando un flujo rápido, pero que nuestras implementaciones pueden
También se realiza sin causar caos e interrupciones
como interrupciones del servicio, impedimentos del servicio o seguridad
o fallas de cumplimiento.

ENFOQUE EN EL DESPLIEGUE TIEMPO PLAZO

Para el resto de este libro, nuestra atención estará en
tiempo de implementación, un subconjunto del flujo de valor
descrito arriba. Este flujo de valor comienza cuando cualquiera
ingeniero *** en nuestro flujo de valor (que incluye
Desarrollo, control de calidad, operaciones de TI e Infosec) verifica
cambia al control de versiones y termina cuando ese cambio es
ejecutando con éxito en producción, proporcionando valor a
el cliente y generar comentarios útiles y
telemetría

La primera fase del trabajo que incluye Diseño y
El desarrollo es similar al desarrollo de productos Lean y
es muy variable y altamente incierto, a menudo requiere
altos niveles de creatividad y trabajo que tal vez nunca
realizado nuevamente, lo que resulta en una alta variabilidad del proceso
veces. En contraste, la segunda fase del trabajo, que
incluye pruebas y operaciones, es similar a Lean
Fabricación. Requiere creatividad y experiencia, y
se esfuerza por ser predecible y mecanicista, con el objetivo
de lograr resultados de trabajo con variabilidad minimizada
(p. ej., plazos de entrega cortos y predecibles, casi cero defectos).

En lugar de procesar grandes lotes de trabajo
secuencialmente a través del valor de diseño / desarrollo
transmitir y luego a través del valor de prueba / operaciones
corriente (como cuando tenemos una gran cascada de lotes
proceso o ramas de características de larga duración), nuestro objetivo es
tener pruebas y operaciones sucediendo simultáneamente
con diseño / desarrollo, permitiendo un flujo rápido y alto
calidad. Este método tiene éxito cuando trabajamos en pequeños
lotes y calidad de construcción en cada parte de nuestro valor
corriente.†††

Definición del tiempo de entrega frente al tiempo de procesamiento

En la comunidad Lean, el tiempo de entrega es una de dos medidas
comúnmente utilizado para medir el rendimiento en valor
flujos, con el otro tiempo de procesamiento
(a veces conocido como tiempo de contacto o tiempo de tarea).‡‡‡

Mientras que el reloj de tiempo de entrega comienza cuando la solicitud es

hecho y finaliza cuando se cumple, el reloj de tiempo de proceso comienza solo cuando comenzamos a trabajar en la solicitud del cliente

—Específicamente, omite el tiempo que el trabajo está en la cola, esperando ser procesado (figura 2).

Figura 2. *Tiempo de entrega frente al tiempo de proceso de una implementación operación*

Porque el tiempo de entrega es lo que experimenta el cliente, nosotros típicamente enfocamos nuestra atención de mejora de procesos allí en lugar de en tiempo de proceso. Sin embargo, la proporción de El tiempo de proceso hasta el tiempo de entrega sirve como una medida importante de eficiencia: lograr un flujo rápido y tiempos de entrega cortos casi siempre requiere reducir el tiempo que nuestro trabajo es esperando en las colas

El escenario común: tiempos de entrega de implementación Meses que requieren

En los negocios como siempre, a menudo nos encontramos en situaciones donde nuestros plazos de implementación requieren meses. Esto es especialmente común en organizaciones grandes y complejas que están trabajando con estrechamente acoplados, monolíticos

aplicaciones, a menudo con escasa prueba de integración
entornos, prueba larga y entorno de producción
plazos de entrega, alta dependencia en pruebas manuales y múltiples
procesos de aprobación requeridos. Cuando esto ocurre, nuestro valor
la transmisión puede verse como la figura 3:

Figura 3: *Un flujo de valor tecnológico con una implementación plazo de entrega de tres meses (Fuente: Damon Edwards, "DevOps Kaizen", 2015.)*

Cuando tenemos largos plazos de implementación, los heroicos son
requerido en casi todas las etapas del flujo de valor. Nosotros
puede descubrir que nada funciona al final del proyecto
cuando fusionamos todos los cambios del equipo de desarrollo
juntos, dando como resultado un código que ya no se construye correctamente
o pasa cualquiera de nuestras pruebas. Arreglar cada problema requiere
días o semanas de investigación para determinar quién rompió
el código y cómo se puede arreglar, y aún así da como resultado
resultados del cliente.

Nuestro ideal de DevOps: plazos de implementación de Minutos

En el ideal de DevOps, los desarrolladores reciben rápido, constante
retroalimentación sobre su trabajo, lo que les permite rápidamente
e implementar, integrar y validar de forma independiente

su código, y tener el código implementado en el
entorno de producción (ya sea mediante la implementación del código
ellos mismos o por otros).

Logramos esto al verificar continuamente el código pequeño
cambios en nuestro repositorio de control de versiones, realizando
pruebas automatizadas y exploratorias en su contra, y
desplegándolo en producción. Esto nos permite tener un
alto grado de confianza en que nuestros cambios funcionarán
según lo diseñado en producción y que cualquier problema puede ser
detectado y corregido rápidamente.

Esto se logra más fácilmente cuando tenemos arquitectura
que es modular, bien encapsulado y poco acoplado
para que los equipos pequeños puedan trabajar con altos grados de
autonomía, con fallas pequeñas y contenidas, y
sin causar interrupciones globales.

En este escenario, se mide nuestro tiempo de implementación
en minutos o, en el peor de los casos, horas. Nuestro resultante
el mapa de flujo de valor debería parecerse a la figura 4:

Figura 4: *Un flujo de valor tecnológico con un tiempo de espera de minutos*

OBSERVANDO "% C / A" COMO MEDIDA DE REHACER

Además de los plazos de entrega y los tiempos de proceso, la tercera clave métrica en el flujo de valor de la tecnología es por ciento completa y precisa (*% C / A*). Esta métrica refleja el calidad de la salida de cada paso en nuestro flujo de valor. Karen Martin y Mike Osterling afirman que "el% C / A se puede obtener preguntando a los clientes intermedios qué porcentaje del tiempo que reciben trabajo que es 'utilizable como es ", lo que significa que pueden hacer su trabajo sin tener que corregir la información que se proporcionó, agregar información faltante que debería haberse proporcionado, o aclarar información que debería haber y podría haber sido más claro ".

LOS TRES CAMINOS: LOS PRINCIPIOS DEVOPS BASES

The Phoenix Project presenta las tres formas como el set de principios subyacentes a partir de los cuales todos los observados Los comportamientos y patrones de DevOps se derivan (Figura 5)

The First Way permite un flujo rápido de trabajo de izquierda a derecha desde Desarrollo a Operaciones al cliente. A fin de que maximizar el flujo, necesitamos hacer visible el trabajo, reducir nuestro tamaños de lotes e intervalos de trabajo, calidad de construcción por

evitar que los defectos pasen a aguas abajo
centros de trabajo, y optimizar constantemente para el mundo
metas.

Figura 5: *Las tres formas (Fuente: Gene Kim, "Las tres Formas: Los principios que sustentan DevOps, "Revolución de TI Press blog, accedido el 9 de agosto de 2016, http://itrevolution.com/the-three-ways-principles-apuntalar-devops / .)*

Al acelerar el flujo a través del valor de la tecnología
flujo, reducimos el tiempo de entrega requerido para cumplir
solicitudes internas o de clientes, especialmente el tiempo
requerido para implementar código en la producción
ambiente. Al hacer esto, aumentamos la calidad de
trabajar tan bien como nuestro rendimiento y aumentar nuestra capacidad para
Superar a la competencia.

Las prácticas resultantes incluyen construcción continua, procesos de integración, prueba e implementación; creando ambientes bajo demanda; Limitando el trabajo en proceso (WIP); y construir sistemas y organizaciones que son seguro para cambiar.

The Second Way permite el flujo rápido y constante de retroalimentación de derecha a izquierda en todas las etapas de nuestro valor corriente. Requiere que amplifiquemos los comentarios para evitar los problemas vuelven a suceder o habilita más rápido Detección y recuperación. Al hacer esto, creamos calidad en la fuente y generar o incorporar conocimiento donde es necesario, esto nos permite crear sistemas cada vez más seguros de trabajar donde se encuentran y solucionan problemas mucho antes de un Se produce una falla catastrófica.

Al ver los problemas a medida que ocurren y enjambrarlos hasta que las contramedidas efectivas estén en su lugar, nosotros acortar y amplificar continuamente nuestros circuitos de retroalimentación, un principio básico de prácticamente todas las mejoras de procesos modernos metodologías Esto maximiza las oportunidades para nuestro organización para aprender y mejorar.

La Tercera Vía permite la creación de una generación generosa confiar en una cultura que apoya una dinámica, disciplinada y

enfoque científico para la experimentación y la toma de riesgos, facilitando la creación de aprendizaje organizacional, tanto

de nuestros éxitos y fracasos. Además, por
acortando y amplificando continuamente nuestros comentarios
bucles, creamos sistemas de trabajo cada vez más seguros y somos
más capaz de correr riesgos y realizar experimentos que
ayúdanos a aprender más rápido que nuestra competencia y ganar en el
mercado.

Como parte de la Tercera Vía, también diseñamos nuestro sistema de
trabajar para que podamos multiplicar los efectos de nuevos
conocimiento, transformando los descubrimientos locales en globales
mejoras Independientemente de donde alguien realice
trabajo, lo hacen con el acumulado y colectivo
experiencia de todos en la organización.

CONCLUSIÓN

En este capítulo, describimos los conceptos de valor
corrientes, el tiempo de entrega como una de las medidas clave de la
efectividad tanto para la fabricación como para la tecnología
flujos de valor y los conceptos de alto nivel detrás de cada
de las Tres Vías, los principios que sustentan DevOps.

En los siguientes capítulos, los principios para cada una de las
Tres formas se describen con mayor detalle. El primero de
estos principios son Flow, que se centra en cómo
cree el flujo rápido de trabajo en cualquier flujo de valor, ya sea
está en fabricación o trabajo tecnológico. Las practicas
que permiten un flujo rápido se describen en la Parte III.

*** En adelante, el *ingeniero se* refiere a cualquier persona que trabaje en nuestro flujo de valor, no solo a los desarrolladores.

††† De hecho, con técnicas como el desarrollo basado en pruebas, las pruebas se realizan incluso antes de la primera Se escribe una línea de código.

‡‡‡ En este libro, el término *tiempo de proceso* se verá favorecido por la misma razón Karen Martin y Mike Cita de Osterling: “Para minimizar la confusión, evitamos usar el término tiempo de ciclo ya que tiene varios definiciones sinónimo de tiempo de procesamiento y ritmo o frecuencia de salida, por nombrar algunos ".

2 La primera forma: *Los principios de Fluir*

En el flujo de valor de la tecnología, el trabajo generalmente fluye de Desarrollo a Operaciones, las áreas funcionales entre nuestros negocios y nuestros clientes. La primera forma requiere el rápido y flujo de trabajo fluido desde el desarrollo hasta las operaciones, para entregar valor a los clientes rápidamente. Optimizamos para este objetivo global en lugar de objetivos locales, como la finalización de la función de desarrollo tasas, prueba de encontrar / corregir relaciones, o medidas de disponibilidad de operaciones.

Aumentamos el flujo al hacer visible el trabajo, al reducir el tamaño de los lotes e intervalos de trabajo, y construyendo calidad en, previniendo defectos por pasar a centros de trabajo aguas abajo. Por acelerando el flujo a través del flujo de valor de la tecnología, nosotros reducir el tiempo de entrega requerido para cumplir con los requisitos internos y externos solicitudes de clientes, aumentando aún más la calidad de nuestro trabajo mientras nos hace más ágiles y capaces de experimentar mejor competencia.

Nuestro objetivo es disminuir la cantidad de tiempo requerido para los cambios en desplegarse en producción y aumentar la fiabilidad y calidad de esos servicios. Pistas sobre cómo hacemos esto en el El flujo de valor de la tecnología se puede deducir de cómo el Lean

Los principios se aplicaron al flujo de valor de fabricación.

HAGA VISIBLE NUESTRO TRABAJO

Una diferencia significativa entre tecnología y fabricación.
El flujo de valor es que nuestro trabajo es invisible. A diferencia de lo físico
procesos, en el flujo de valor de la tecnología no podemos ver fácilmente
donde se impide el flujo o cuando el trabajo se acumula delante de
centros de trabajo restringidos Transferencia de trabajo entre trabajo
los centros suelen ser muy visibles y lentos porque el inventario debe
ser movido físicamente

Sin embargo, en el trabajo tecnológico, el movimiento se puede hacer con un clic de
un botón, como reasignar un ticket de trabajo a otro equipo.
Porque es tan fácil, el trabajo puede rebotar interminablemente entre equipos
debido a información incompleta, o el trabajo puede pasarse a
centros de trabajo aguas abajo con problemas que permanecen completamente
invisible hasta que lleguemos tarde entregando lo que prometimos a la
cliente o nuestra aplicación falla en el entorno de producción.

Para ayudarnos a ver dónde fluye bien el trabajo y dónde está el trabajo
en cola o estancado, necesitamos hacer nuestro trabajo lo más visible posible.
Uno de los mejores métodos para hacerlo es usar tableros de trabajo visuales,
como tableros kanban o tableros de planificación de sprint, donde podemos
representar el trabajo en tarjetas físicas o electrónicas. El trabajo se origina en
la izquierda (a menudo extraída de una cartera de pedidos), se extrae del trabajo
centro a centro de trabajo (representado en columnas) y termina cuando
llega al lado derecho del tablero, generalmente en una columna etiquetada
"Hecho" o "en producción".

Figura 6: *Un ejemplo de tablero kanban, que abarca los requisitos, Dev, Prueba, puesta en escena y en producción (Fuente: David J. Andersen y Dominica DeGrandis,* Kanban para ITOps , *materiales de capacitación para taller, 2012.)*

Nuestro trabajo no solo se hace visible, también podemos gestionar nuestro trabaje para que fluya de izquierda a derecha lo más rápido posible. Además, podemos medir el tiempo de espera de cuando una tarjeta es colocado en el tablero para cuando se mueve a la columna "Listo".

Idealmente, nuestro tablero kanban abarcará todo el flujo de valor, definir el trabajo como completado solo cuando llega al lado derecho de

el tablero (figura 6). El trabajo no se realiza cuando el desarrollo completa la implementación de una característica; más bien, solo se hace cuando nuestra aplicación se ejecuta con éxito en producción, entregando valor al cliente.

Al poner todo el trabajo para cada centro de trabajo en colas y hacerlo visible, todos los interesados pueden priorizar más fácilmente el trabajo en el contexto de objetivos globales. Hacer esto permite que cada centro de trabajo tarea única en el trabajo de mayor prioridad hasta que se complete, Aumento del rendimiento.

LÍMITE DE TRABAJO EN PROCESO (WIP)

En la manufactura, el trabajo diario generalmente es dictado por una producción. horario que se genera regularmente (por ejemplo, diariamente, semanalmente),

establecer qué trabajos deben ejecutarse en función de los pedidos de los clientes, fechas de vencimiento del pedido, piezas disponibles, etc.

En tecnología, nuestro trabajo suele ser mucho más dinámico, esto es especialmente el caso en servicios compartidos, donde los equipos deben satisfacer Las demandas de muchas partes interesadas diferentes. Como resultado, diariamente el trabajo se vuelve dominado por la prioridad *del día* , a menudo con solicitudes de trabajo urgente que llegan a través de cada comunicación posible mecanismo, incluidos sistemas de tickets, llamadas de interrupción, correos electrónicos, llamadas telefónicas, salas de chat y escaladas de administración.

Las interrupciones en la fabricación también son muy visibles y costosas, a menudo requiere romper el trabajo actual y eliminar cualquier trabajo incompleto en proceso para comenzar el nuevo trabajo. Este alto nivel de El esfuerzo desalienta las interrupciones frecuentes.

Sin embargo, interrumpir a los trabajadores de la tecnología es fácil, porque el

las consecuencias son invisibles para casi todos, a pesar de que
El impacto negativo en la productividad puede ser mucho mayor que en
fabricación. Por ejemplo, un ingeniero asignado a múltiples
los proyectos deben cambiar entre tareas, incurriendo en todos los costos de
tener que restablecer el contexto, así como las reglas y objetivos cognitivos.

Los estudios han demostrado que el tiempo para completar incluso tareas simples,
como ordenar formas geométricas, se degrada significativamente cuando
multitarea Por supuesto, porque nuestro trabajo en el valor tecnológico
la secuencia es mucho más compleja cognitivamente que la clasificación geométrica
formas, los efectos de la multitarea en el tiempo de proceso son mucho peores.

Podemos limitar la multitarea cuando usamos un tablero kanban para administrar
nuestro trabajo, como codificar y aplicar WIP (trabajo en
progreso) límites para cada columna o centro de trabajo que pone un límite
limitar el número de tarjetas que pueden estar en una columna.

Por ejemplo, podemos establecer un límite de WIP de tres tarjetas para la prueba.
Cuando ya hay tres cartas en la línea de prueba, no hay cartas nuevas
se puede agregar al carril a menos que se complete o retire una tarjeta
de la columna "en el trabajo" y volver a poner en la cola (es decir, poner
la tarjeta de regreso a la columna a la izquierda). Nada puede ser

Página 55

trabajado hasta que se represente primero en una tarjeta de trabajo, reforzando
que todo el trabajo debe hacerse visible.

Dominica DeGrandis, uno de los principales expertos en el uso de kanbans
en las secuencias de valor de DevOps, señala que "controlar el tamaño de la cola [WIP]
es una herramienta de administración extremadamente poderosa, ya que es una de las pocas
Indicadores principales del tiempo de entrega: con la mayoría de los elementos de trabajo, no
sepa cuánto tiempo tomará hasta que se complete realmente ".

Limitar WIP también facilita ver problemas que impiden
finalización del trabajo.† Por ejemplo, cuando limitamos WIP, encontramos
para que no tengamos nada que hacer porque estamos esperando

alguien más. Aunque puede ser tentador comenzar un nuevo trabajo (es decir, "Es mejor estar haciendo algo que nada"), mucho mejor la acción sería averiguar qué está causando el retraso y ayudar a solucionar ese problema. La multitarea mala a menudo ocurre cuando las personas son asignado a múltiples proyectos, lo que resulta en muchas prioridades problemas.

En otras palabras, como David J. Andersen, autor de *Kanban: Cambio evolutivo exitoso para su negocio de tecnología* , bromeó: “Deja de comenzar. Comienza a terminar.

REDUCIR TAMAÑOS DE LOTE

Otro componente clave para crear un flujo suave y rápido es realizar trabajos en lotes pequeños. Antes de la inclinación revolución de fabricación, era una práctica común fabricar en lotes grandes (o lotes), especialmente para operaciones donde La configuración del trabajo o el cambio entre trabajos era lento o costoso. Por ejemplo, la producción de paneles de carrocería grandes requiere configuración troqueles grandes y pesados en máquinas de estampado de metal, un proceso que Podría tomar días. Cuando el costo de cambio es tan caro, lo haríamos a menudo estampan tantos paneles a la vez como sea posible, creando grandes lotes para reducir el número de cambios.

Sin embargo, los tamaños de lotes grandes dan como resultado niveles vertiginosos de WIP y altos niveles de variabilidad en el flujo que caen en cascada a través del planta de fabricación completa. El resultado es largos plazos de entrega y pobre calidad: si se encuentra un problema en un panel de cuerpo, todo el lote tiene que ser desechado

Una de las lecciones clave en Lean es que para reducir los tiempos de entrega y aumentar la calidad, debemos esforzarnos por reducir continuamente el lote Tamaños. El límite inferior teórico para el tamaño del lote es de una *sola pieza.* *flujo* , donde cada operación se realiza una unidad a la vez. [1]

Las diferencias dramáticas entre los lotes grandes y pequeños pueden ser visto en la simple simulación de envío de boletines descritos en *Lean Thinking: desterre el desperdicio y cree riqueza en su Corporación* de James P. Womack y Daniel T. Jones.

Supongamos que en nuestro propio ejemplo tenemos diez folletos para enviar y enviar cada folleto por correo requiere cuatro pasos: doblar el papel, insertar el papel en el sobre, selle el sobre y selle el sobre.

La estrategia de lote grande (es decir, "producción en masa") sería realice secuencialmente una operación en cada uno de los diez folletos. En otras palabras, primero doblaríamos las diez hojas de papel, luego inserte cada uno de ellos en sobres, luego selle los diez sobres, y luego sellarlos.

Por otro lado, en la estrategia de lotes pequeños (es decir, "pieza única flujo "), todos los pasos necesarios para completar cada folleto son realizado secuencialmente antes de comenzar en el próximo folleto. En En otras palabras, doblamos una hoja de papel, la insertamos en el sobre, sellarlo y sellarlo; solo entonces comenzamos el proceso de nuevo con La siguiente hoja de papel.

La diferencia entre usar lotes grandes y pequeños es dramático (ver figura 7) Supongamos que cada una de las cuatro operaciones toma diez segundos para cada uno de los diez sobres. Con el gran tamaño de lote estrategia, se produce el primer sobre completado y sellado solo después de 310 segundos.

Peor aún, supongamos que descubrimos durante la operación de sellado de sobres

que cometimos un error en el primer paso de plegado, en este caso, el
lo primero que descubriríamos es que el error es a los doscientos segundos,
y tenemos que replegar y reinsertar los diez folletos en nuestro lote
de nuevo.

Figura 7: *Simulación del "juego de sobres" (doblar, insertar, sellar y sellar el sobre)*

(Fuente: Stefan Luyten, "Flujo de una sola pieza: por qué la producción en masa no es la forma más eficiente de hacer 'cosas' ", Medium.com, 8 de agosto, 2014, https://medium.com/@stefanluyten/single-piece-flow-5d2c2bec845b#.9o7sn74ns .)

En contraste, en la estrategia de lotes pequeños se completó la primera
el sobre estampado se produce en solo cuarenta segundos, ocho veces
más rápido que la estrategia de lotes grandes. Y, si cometimos un error en
El primer paso, solo tenemos que rehacer un folleto en nuestro lote.
Los tamaños de lote pequeños resultan en menos WIP, tiempos de entrega más rápidos, más rápido
detección de errores y menos retrabajo.

Los resultados negativos asociados con grandes tamaños de lote son tan
relevante para el flujo de valor de la tecnología como en la fabricación.
Considere cuándo tenemos un cronograma anual para lanzamientos de software,
donde el código de todo un año que tiene el desarrollo
Trabajado en se lanza a la implementación de producción.

Al igual que en la fabricación, este lanzamiento de lotes grandes crea repentinamente,
altos niveles de WIP e interrupciones masivas en todos los niveles posteriores
centros de trabajo, lo que resulta en un flujo deficiente y resultados de baja calidad.
Esto valida nuestra experiencia común de que cuanto mayor sea el cambio
entrando en producción, cuanto más difíciles son los errores de producción
para diagnosticar y reparar, y cuanto más tardan en remediar.

En una publicación sobre *Lecciones aprendidas de inicio* , Eric Ries afirma: "El lote el tamaño es la unidad en la que los productos de trabajo se mueven entre etapas en un proceso de desarrollo [o DevOps]. Para el software, el lote más fácil para ver es el código. Cada vez que un ingeniero registra el código, son agrupando una cierta cantidad de trabajo. Hay muchas tecnicas para controlar estos lotes, que van desde los pequeños lotes necesario para el despliegue continuo en sucursales más tradicionales desarrollo basado, donde todo el código de múltiples los desarrolladores que trabajan durante semanas o meses se agrupan y integrados juntos ".

El equivalente al flujo de una sola pieza en el flujo de valor de la tecnología. se realiza con despliegue continuo, donde cada cambio comprometido con el control de versiones se integra, se prueba y se implementa en producción Las prácticas que permiten esto se describen en Parte IV

REDUCIR EL NÚMERO DE ESPOSAS

En el flujo de valor de la tecnología, siempre que tengamos mucho plazos de implementación medidos en meses, a menudo es porque hay cientos (o incluso miles) de operaciones requeridas para mover nuestro código del control de versiones a la producción ambiente. Para transmitir código a través del flujo de valor se requiere múltiples departamentos para trabajar en una variedad de tareas, que incluyen pruebas funcionales, pruebas de integración, creación de entornos, administración del servidor, administración de almacenamiento, redes, carga equilibrio y seguridad de la información.

Cada vez que el trabajo pasa de un equipo a otro, requerimos todo tipo de comunicación: solicitud, especificación, señalización, coordinación, y a menudo priorizando, programando, desconfiando, probando y verificar Esto puede requerir el uso de diferentes tickets o proyectos sistemas de gestión; redacción de documentos de especificaciones técnicas;

comunicarse a través de reuniones, correos electrónicos o llamadas telefónicas; y usando el archivo recursos compartidos del sistema, servidores FTP y páginas Wiki.

Cada uno de estos pasos es una cola potencial donde el trabajo esperará cuando confiamos en recursos que se comparten entre diferentes valores flujos (por ejemplo, operaciones centralizadas). Los plazos de entrega de estos las solicitudes son a menudo tan largas que hay una escalada constante para tener trabajo realizado dentro de los plazos necesarios.

Incluso en las mejores circunstancias, algo de conocimiento es inevitable perdido con cada traspaso. Con suficientes traspasos, el trabajo puede perder completamente el contexto del problema que se está resolviendo o el objetivo organizativo que se apoya. Por ejemplo, un servidor el administrador puede ver un ticket recién creado solicitando a ese usuario crear cuentas sin saber qué aplicación o servicio es por qué necesita ser creado, cuáles son todas las dependencias, o si es realmente un trabajo recurrente.

Para mitigar este tipo de problemas, nos esforzamos por reducir el cantidad de traspasos, ya sea automatizando porciones significativas de el trabajo o reorganizando equipos para que puedan entregar valor al cliente, en lugar de tener que estar constantemente Depende de los demás. Como resultado, aumentamos el flujo al reducir el cantidad de tiempo que nuestro trabajo pasa esperando en la cola, así como la cantidad de tiempo sin valor agregado. Ver Apéndice 4 .

IDENTIFICAR Y ELEVAR CONTINUAMENTE NUESTRAS RESTRICCIONES

Para reducir los tiempos de entrega y aumentar el rendimiento, necesitamos Identificar continuamente las limitaciones de nuestro sistema y mejorar su trabajo. capacidad. En *Beyond the Goal* , el Dr. Goldratt afirma: "En cualquier valor

corriente, siempre hay una dirección de flujo, y siempre hay una y solo restricción; cualquier mejora no realizada en esa restricción es una ilusión ". Si mejoramos un centro de trabajo que se posiciona antes de la restricción, el trabajo simplemente se acumulará en el cuello de botella aún más rápido, esperando que el cuello de botella realice el trabajo centro de trabajo.

Por otro lado, si mejoramos un centro de trabajo posicionado *después* el cuello de botella, sigue hambriento, esperando que el trabajo despeje el embotellamiento. Como solución, el Dr. Goldratt definió los "cinco enfoques pasos":

> Identificar la restricción del sistema.
>
> Decide cómo explotar la restricción del sistema.
>
> Subordinar todo lo demás a las decisiones anteriores.
>
> Eleve la restricción del sistema.
>
> Si en los pasos anteriores se ha roto una restricción, regrese al primer paso, pero no permita que la inercia provoque un sistema restricción.

En las transformaciones típicas de DevOps, a medida que avanzamos desde plazos de implementación medidos en meses o trimestres para liderar veces medidas en minutos, la restricción generalmente sigue esto progresión:

> **Creación del entorno:** no podemos lograr implementaciones en exigir si siempre tenemos que esperar semanas o meses para entornos de producción o prueba. La contramedida es crear entornos que sean a la demanda y completamente autónomos

servicio, para que siempre estén disponibles cuando los necesitemos.

Implementación de código: no podemos lograr implementaciones en exigir si cada una de nuestras implementaciones de código de producción toma semanas o meses para realizar (es decir, cada implementación requiere 1.300 pasos manuales propensos a errores, que involucran hasta tres cien ingenieros). La contramedida es automatizar nuestro despliegues tanto como sea posible, con el objetivo de ser completamente automatizado para que pueda ser autoservicio por cualquier desarrollador.

Página 61

Configuración y ejecución de prueba: no podemos lograr implementaciones en exigir si cada implementación de código requiere dos semanas para configurar nuestros entornos de prueba y conjuntos de datos, y otras cuatro semanas para ejecutar manualmente todas nuestras pruebas de regresión. La contramedida es automatizar nuestras pruebas para que podamos ejecutar implementaciones de manera segura y para paralelizarlos para que la tasa de prueba pueda mantenerse al día con nuestra tasa de desarrollo del código.

Arquitectura demasiado **ajustada:** no podemos lograr implementaciones bajo demanda si la arquitectura demasiado ajustada significa que cada vez queremos hacer un cambio de código tenemos que enviar a nuestros ingenieros a decenas de reuniones del comité para obtener permiso para Haz nuestros cambios. Nuestra contramedida es crear más arquitectura débilmente acoplada para que se puedan hacer cambios de forma segura y con más autonomía, aumentando el desarrollador productividad.

Después de que se hayan roto todas estas restricciones, nuestra restricción probablemente sea el desarrollo o los propietarios del producto. Porque nuestro objetivo es

para permitir que pequeños equipos de desarrolladores se desarrollen independientemente, prueba y despliegue valor a los clientes de forma rápida y confiable, esto es donde queremos que esté nuestra restricción. Alto rendimiento, independientemente de si un ingeniero está en Desarrollo, QA, Ops o Infosec, declare que su objetivo es ayudar a maximizar la productividad del desarrollador.

Cuando la restricción está aquí, solo estamos limitados por el número de buenas hipótesis de negocio que creamos y nuestra capacidad para desarrollar el código necesario para probar estas hipótesis con clientes reales.

La progresión de las restricciones enumeradas anteriormente son generalizaciones de transformaciones típicas: técnicas para identificar la restricción en flujos de valor reales, como a través del mapeo de flujo de valor y medidas, se describen más adelante en este libro.

ELIMINAR DIFICULTADES Y RESIDUOS EN LA CORRIENTE DE VALOR

Shigeo Shingo, uno de los pioneros de la producción de Toyota Sistema, cree que los residuos constituyen la mayor amenaza para viabilidad comercial: la definición comúnmente utilizada en Lean es "la uso de cualquier material o recurso más allá de lo que el cliente requiere y está dispuesto a pagar ". Definió siete tipos principales de Residuos de fabricación: inventario, sobreproducción, extra procesamiento, transporte, espera, movimiento y defectos.

Las interpretaciones más modernas de Lean han señalado que "eliminar desperdicio ”puede tener un contexto degradante y deshumanizante; en lugar, El objetivo se reformula para reducir las dificultades y el trabajo pesado en nuestro diario trabajar a través del aprendizaje continuo para lograr el objetivos de la organización. Para el resto de este libro, el término *desperdicio* implicará esta definición más moderna, ya que

coincide con los ideales de DevOps y los resultados deseados.

En el libro *Implementing Lean Software Development: From Concept to Cash* , Mary y Tom Poppendieck describen el desperdicio y dificultades en el flujo de desarrollo de software como cualquier cosa que causa demoras para el cliente, como actividades que pueden ser anulado sin afectar el resultado.

Las siguientes categorías de desechos y dificultades provienen de *Implementación de desarrollo de software Lean a* menos que se indique lo contrario célebre:

Trabajo realizado parcialmente: esto incluye cualquier trabajo en el valor transmisión que no se ha completado (por ejemplo, requisito documentos u órdenes de cambio aún no revisados) y trabajo que es sentado en la cola (por ejemplo, esperando la revisión de control de calidad o el administrador del servidor boleto). El trabajo parcialmente realizado se vuelve obsoleto y pierde valor a medida que pasa el tiempo

Procesos adicionales: cualquier trabajo adicional que se esté realizando realizado en un proceso que no agrega valor al cliente. Esto puede incluir documentación no utilizada en un centro de trabajo aguas abajo, o revisiones o aprobaciones que no

agregar valor a la salida. Los procesos adicionales agregan esfuerzo y aumentar los plazos de entrega

Características adicionales: características integradas en el servicio que no son que necesita la organización o el cliente (por ejemplo, "oro enchapado"). Las características adicionales agregan complejidad y esfuerzo a las pruebas y gestión de funcionalidad.

Cambio de tareas: cuando las personas se asignan a múltiples

proyectos y flujos de valor, que requieren que cambien de contexto y administrar dependencias entre el trabajo, agregando adicional esfuerzo y tiempo en el flujo de valor.

En espera: cualquier retraso entre trabajos que requiera recursos para esperar hasta que puedan completar el trabajo actual. Los retrasos aumentan el ciclo tiempo y evitar que el cliente obtenga valor.

Movimiento: la cantidad de esfuerzo para mover información o materiales de un centro de trabajo a otro. Lata de residuos de movimiento ser creado cuando las personas que necesitan comunicarse con frecuencia No están colocados. Las transferencias también crean desperdicio de movimiento y, a menudo requieren comunicación adicional para resolver ambigüedades.

Defectos: información, materiales incorrectos, faltantes o poco claros, o los productos generan desperdicio, ya que se necesita un esfuerzo para resolverlos cuestiones. Cuanto mayor sea el tiempo entre la creación del defecto y el defecto detección, cuanto más difícil es resolver el defecto.

Trabajo no estándar o manual: dependencia de no estándar o trabajo manual de otros, como el uso de no reconstrucción servidores, entornos de prueba y configuraciones. Idealmente, cualquier Las dependencias de las operaciones deben ser automáticas, con servicio y disponible bajo demanda.

Heroica: para que una organización logre objetivos, los individuos y los equipos se colocan en una posición donde deben realizar actos irracionales, que incluso pueden formar parte de

su trabajo diario (p. ej., problemas nocturnos a las 2:00 a. m. en producción, creando cientos de tickets de trabajo como parte de cada lanzamiento de software). §

Nuestro objetivo es generar estos desechos y dificultades en cualquier lugar la heroicidad se hace necesaria, visible y hacer sistemáticamente lo que es necesario para aliviar o eliminar estas cargas y dificultades para Alcanzar nuestro objetivo de flujo rápido.

CONCLUSIÓN

Mejorar el flujo a través del flujo de valor de la tecnología es esencial para lograr los resultados de DevOps. Hacemos esto haciendo trabajo visible, limitando WIP, reduciendo los tamaños de lote y la cantidad de transferencias, identificando y evaluando continuamente nuestras limitaciones, y eliminando dificultades en nuestro trabajo diario.

Las prácticas específicas que permiten un flujo rápido en el valor DevOps flujo se presentan en la Parte IV. En el próximo capítulo, presentamos La segunda vía: los principios de la retroalimentación.

† Taiichi Ohno comparó el cumplimiento de los límites de WIP con el drenaje de agua del río de inventario para revelar todo Los problemas que obstruyen el flujo rápido.

‡ También conocido como "tamaño de lote de uno" o "flujo 1x1", términos que se refieren al tamaño de lote y un límite WIP de uno.

§ Aunque la heroicidad no se incluye en las categorías de desechos de Poppendieck, se incluye aquí debido a la frecuencia ocurre, especialmente en la operación de servicios compartidos.

3 La segunda forma: *Los principios de Realimentación*

Mientras que First Way describe los principios que permiten el flujo rápido de trabajar de izquierda a derecha, la Segunda Vía describe los principios que permitir la retroalimentación recíproca rápida y constante de derecha a izquierda etapas del flujo de valor. Nuestro objetivo es crear un lugar cada vez más seguro y más sistema de trabajo resistente.

Esto es especialmente importante cuando se trabaja en sistemas complejos, cuando La primera oportunidad para detectar y corregir errores es típicamente cuando un evento catastrófico está en marcha, como un trabajador de fabricación herido en el trabajo o la fusión de un reactor nuclear en progreso.

En tecnología, nuestro trabajo ocurre casi por completo dentro de complejos sistemas con alto riesgo de consecuencias catastróficas. Como en fabricación, a menudo descubrimos problemas solo cuando hay grandes fallas en curso, como un corte de producción masivo o una violación de seguridad resultando en el robo de datos del cliente.

Hacemos que nuestro sistema de trabajo sea más seguro mediante la creación rápida, frecuente y alta flujo de información de calidad a lo largo de nuestro flujo de valor y nuestro organización, que incluye retroalimentación y bucles de retroalimentación. Esta nos permite detectar y remediar problemas mientras son más pequeños, más barato y más fácil de arreglar; evitar problemas antes de que causen catástrofe; y crear aprendizaje organizacional que integramos en trabajo futuro. Cuando ocurren fallas y accidentes, los tratamos como oportunidades de aprendizaje, en oposición a una causa de castigo y culpa. Para lograr todo lo anterior, primero exploremos la naturaleza de sistemas complejos y cómo pueden hacerse más seguros.

TRABAJAR CON SEGURIDAD DENTRO DEL COMPLEJO SISTEMAS

Una de las características definitorias de un sistema complejo es que desafía la capacidad de cualquier persona para ver el sistema como un todo y comprender cómo encajan todas las piezas Los sistemas complejos suelen tener un alto grado de interconexión de componentes estrechamente acoplados, y el comportamiento a nivel del sistema no puede explicarse simplemente en términos de comportamiento de los componentes del sistema.

El Dr. Charles Perrow estudió la crisis de Three Mile Island y observó que era imposible que alguien entendiera cómo funcionaría el reactor comportarse en todas las circunstancias y cómo podría fallar. Cuando un problema era en marcha en un componente, era difícil aislarlo del otro componentes, que fluyen rápidamente a través de los caminos de menor resistencia en formas impredecibles

Dr. Sidney Dekker, quien también codificó algunos de los elementos clave de seguridad. cultura, observó otra característica de los sistemas complejos: hacer el Lo mismo dos veces no conducirá previsiblemente o necesariamente a lo mismo resultado. Es esta característica que hace listas de verificación estáticas y mejor prácticas, aunque valiosas, insuficientes para evitar catástrofes ocurriendo VerApéndice 5.

Por lo tanto, porque el fracaso es inherente e inevitable en complejos sistemas, debemos diseñar un sistema de trabajo seguro, ya sea en fabricación o tecnología, donde podemos realizar trabajos sin miedo, confía en que cualquier error se detectará rápidamente, mucho antes de que causar resultados catastróficos, como lesiones laborales, defectos del producto o Impacto negativo en el cliente.

Después de descifrar el mecanismo causal detrás del Producto Toyota Sistema como parte de su tesis doctoral en la Harvard Business School, Dr. Steven Spear declaró que el diseño de sistemas perfectamente seguros es probable más allá de nuestras capacidades, pero podemos hacer que sea más seguro trabajar en complejos sistemas cuando se cumplen las cuatro condiciones siguientes:†

El trabajo complejo se gestiona de modo que los problemas de diseño y las operaciones se revelan

Los problemas se enjambran y se resuelven, lo que resulta en una construcción rápida de nuevos conocimientos

El nuevo conocimiento local se explota a nivel mundial en todo el organización

Los líderes crean otros líderes que continuamente hacen crecer este tipo de capacidades

Se requiere que cada una de estas capacidades trabaje de manera segura en un complejo sistema. En las siguientes secciones, las dos primeras capacidades y sus importancia se describen, así como cómo se han creado en otros dominios y qué prácticas les permiten en el valor tecnológico corriente. (Las capacidades tercera y cuarta se describen en el capítulo 4).

VEA LOS PROBLEMAS A medida que OCURREN

En un sistema de trabajo seguro, debemos probar constantemente nuestro diseño y supuestos de funcionamiento. Nuestro objetivo es aumentar el flujo de información en nuestro sistema de tantas áreas como sea posible, antes, más rápido, más barato y con tanta claridad entre causa y efecto como sea posible. Mas supuestos que podemos invalidar, cuanto más rápido podamos encontrar y solucionar problemas, aumentando nuestra capacidad de recuperación, agilidad y capacidad de aprender e innovar.

Hacemos esto creando retroalimentación y bucles de retroalimentación en nuestro sistema de trabajo. Dr. Peter Senge en su libro *La Quinta Disciplina: El Arte y La práctica de la Organización de Aprendizaje* describió los circuitos de retroalimentación como Parte crítica del aprendizaje de las organizaciones y el pensamiento sistémico. Realimentación y los bucles de avance hacen que los componentes dentro de un sistema se refuercen

o contrarrestarse unos a otros.

En la fabricación, la ausencia de retroalimentación efectiva a menudo contribuye a Principales problemas de calidad y seguridad. En un caso bien documentado en el Planta de fabricación de General Motors Fremont, no hubo efectivo procedimientos establecidos para detectar problemas durante el proceso de ensamblaje, ni hubo procedimientos explícitos sobre qué hacer cuando los problemas eran encontró. Como resultado, hubo instancias de motores en funcionamiento. hacia atrás, a los autos les faltan volantes o llantas, y los autos incluso tienen que ser remolcado fuera de la línea de montaje porque no comenzarían.

Página 69

Por el contrario, en las operaciones de fabricación de alto rendimiento hay un rápido, flujo de información frecuente y de alta calidad en todo el valor flujo: cada operación de trabajo se mide y supervisa, y cualquier Los defectos o desviaciones significativas se encuentran rápidamente y se actúa sobre ellos. Estos son los cimientos de lo que permite la calidad, la seguridad y la continuidad. aprendizaje y mejora.

En el flujo de valor de la tecnología, a menudo obtenemos malos resultados debido a La ausencia de retroalimentación rápida. Por ejemplo, en un software de cascada proyecto, podemos desarrollar código para todo un año y no recibir comentarios sobre calidad hasta que comenzamos la fase de prueba, o peor, cuando lanzamos nuestro software para clientes. Cuando la retroalimentación es tan tardía e infrecuente, es demasiado lento para permitirnos prevenir resultados no deseados.

Por el contrario, nuestro objetivo es crear retroalimentación rápida y bucles de avance rápido dondequiera que se realice el trabajo, en todas las etapas del valor de la tecnología flujo, que abarca la gestión de productos, desarrollo, control de calidad, Infosec, y Operaciones. Esto incluye la creación de compilación automatizada, integración y procesos de prueba, para que podamos detectar inmediatamente cuándo Se ha introducido un cambio que nos saca de un correcto estado funcional y desplegable.

También creamos telemetría generalizada para que podamos ver cómo todo nuestro sistema los componentes operan en el entorno de producción, por lo que puede detectar rápidamente cuando no están funcionando como se esperaba. Telemetría

también nos permite medir si estamos logrando nuestros objetivos previstos e, idealmente, se irradia a todo el flujo de valor para que podamos ver cómo Nuestras acciones afectan a otras partes del sistema como un todo.

Los bucles de retroalimentación no solo permiten la detección y recuperación rápidas de problemas, pero también nos informan sobre cómo prevenir estos problemas de ocurrir nuevamente en el futuro. Hacer esto aumenta la calidad y seguridad de nuestro sistema de trabajo, y crea aprendizaje organizacional.

Como Elisabeth Hendrickson, vicepresidenta de ingeniería de Pivotal Software, Inc. y autor de *Explore It !: Reduce el riesgo y aumenta la confianza con Pruebas exploratorias* , dijo: "Cuando dirigí la ingeniería de calidad, yo describió mi trabajo como "crear ciclos de retroalimentación". La retroalimentación es crítica porque es lo que nos permite dirigirnos. Debemos validar constantemente

entre las necesidades del cliente, nuestras intenciones y nuestras implementaciones. Las pruebas son simplemente un tipo de retroalimentación ”.

ENJUAGUE Y RESUELVE PROBLEMAS PARA CONSTRUIR NUEVOS CONOCIMIENTOS

Obviamente, no es suficiente detectar simplemente cuando lo inesperado ocurre. Cuando ocurren problemas, debemos enjambrarlos, movilizar quien sea requerido para resolver el problema.

Según el Dr. Spear, el objetivo del enjambre es contener problemas antes de que tengan la oportunidad de propagarse y diagnosticar y tratar el problema para que no pueda repetirse. "Al hacerlo", dice, "construyen siempre-conocimiento más profundo sobre cómo administrar los sistemas para hacer nuestro trabajo, convirtiendo la inevitable ignorancia inicial en conocimiento ".

El modelo de este principio es el *cable* Toyota *Andon* . En da Toyota planta de fabricación, encima de cada centro de trabajo hay un cable que cada

el trabajador y el gerente están capacitados para tirar cuando algo sale mal; para ejemplo, cuando una parte es defectuosa, cuando una parte requerida no está disponible, o incluso cuando el trabajo lleva más tiempo de lo documentado. ‡

Cuando se tira del cordón de Andon, se alerta al líder del equipo y inmediatamente trabaja para resolver el problema. Si el problema no puede ser resuelto dentro de un tiempo especificado (por ejemplo, cincuenta y cinco segundos), la producción la línea se detiene para que toda la organización pueda movilizarse para ayudar con resolución de problemas hasta que se haya tomado una contramedida exitosa desarrollado.

En lugar de solucionar el problema o programar una solución "cuando tener más tiempo ", pululamos para arreglarlo de inmediato, esto es casi el opuesto al comportamiento en la planta de GM Fremont descrito anteriormente. El enjambre es necesario por las siguientes razones:

- Impide que el problema progrese aguas abajo, donde el el costo y el esfuerzo para repararlo aumentan exponencialmente y la deuda técnica se permite acumular

- Impide que el centro de trabajo comience un nuevo trabajo, lo que probablemente Introducir nuevos errores en el sistema.

Page 71

- Si no se soluciona el problema, el centro de trabajo podría potencialmente tiene el mismo problema en la próxima operación (por ejemplo, cincuenta y cinco segundos más tarde), que requieren más arreglos y trabajo. VerApéndice 6.

Esta práctica de enjambre parece contraria a la gestión común práctica, ya que deliberadamente permitimos que un problema local interrumpa operaciones a nivel mundial. Sin embargo, el enjambre permite el aprendizaje. Previene la pérdida de información crítica debido a recuerdos que se desvanecen o cambian circunstancias. Esto es especialmente crítico en sistemas complejos, donde muchos problemas ocurren debido a algunos inesperados, idiosincrásicos interacción de personas, procesos, productos, lugares y circunstancias: A medida que pasa el tiempo, se hace imposible reconstruir exactamente lo que fue

sucediendo cuando ocurrió el problema.
Como señala el Dr. Spear, el enjambre es parte del "ciclo disciplinado de reconocimiento de problemas de tiempo, diagnóstico, ... y tratamiento (contramedidas o medidas correctivas en la fabricación vernácula). Es el disciplina del ciclo de Shewhart: planificar, hacer, verificar, actuar, popularizado por W. Edwards Deming, pero aceleró a la velocidad de deformación ".

Es solo a través del enjambre de problemas cada vez más pequeños descubiertos cada vez más temprano en el ciclo de vida que podemos desviar problemas antes de un Se produce una catástrofe. En otras palabras, cuando el reactor nuclear se derrite abajo, ya es demasiado tarde para evitar los peores resultados.

Para permitir una retroalimentación rápida en el flujo de valor de la tecnología, debemos crear el equivalente de un cable Andon y la respuesta de enjambre relacionada.
Esto requiere que también creamos la cultura que lo haga seguro, y incluso alentado, a tirar del cordón de Andon cuando algo sale mal, si es cuando ocurre un incidente de producción o cuando ocurren errores anteriormente en la secuencia de valor, como cuando alguien introduce un cambio eso rompe nuestros procesos continuos de construcción o prueba.

Cuando las condiciones desencadenan un tirón del cable Andon, enjambramos para resolver el problema y evitar la introducción de nuevos trabajos hasta que el problema haya resuelto§ Esto proporciona retroalimentación rápida para todos en el valor stream (especialmente la persona que causó la falla del sistema), nos permite para aislar y diagnosticar rápidamente el problema, y evita más factores complicados que pueden ocultar la causa y el efecto.

Prevenir la introducción de nuevos trabajos permite la continua integración y despliegue, que es flujo de una sola pieza en el flujo de valor tecnológico. Todos los cambios que pasan nuestra construcción continua y las pruebas de integración se implementan en producción, y cualquier cambio que hacen que las pruebas fallen, activan nuestro cable de Andon y se enjambran hasta que se resuelva.

MANTENGA LA CALIDAD DE EMPUJE MÁS CERCA DE LA FUENTE

Inadvertidamente podemos perpetuar sistemas de trabajo inseguros debido a la forma en que respondemos a accidentes e incidentes. En sistemas complejos, agregar más pasos de inspección y procesos de aprobación en realidad aumentan el probabilidad de fallas futuras. La efectividad de los procesos de aprobación. disminuye a medida que empujamos la toma de decisiones más lejos de donde el Se realiza el trabajo. Hacerlo no solo disminuye la calidad de las decisiones sino que también aumenta nuestro tiempo de ciclo, disminuyendo así la fuerza del retroalimentación entre causa y efecto, y reducir nuestra capacidad de aprender de éxitos y fracasos. ¶

Esto se puede ver incluso en sistemas más pequeños y menos complejos. Cuando arriba abajo, los sistemas burocráticos de mando y control se vuelven ineficaces, generalmente se debe a la diferencia entre "quién debería hacer algo" y "quién realmente está haciendo algo" es demasiado grande, debido a insuficiente claridad y actualidad.

Los ejemplos de controles de calidad ineficaces incluyen:

- Requerir que otro equipo complete tedioso, propenso a errores y tareas manuales que podrían automatizarse fácilmente y ejecutarse según sea necesario el equipo que necesita el trabajo realizado

- Requerir aprobaciones de personas ocupadas que están distantes del trabajar, obligándolos a tomar decisiones sin una adecuada conocimiento del trabajo o las posibles implicaciones, o simplemente sello de goma sus aprobaciones

- Crear grandes volúmenes de documentación de detalles cuestionables. que se vuelven obsoletas poco después de que se escriben

- Empujar grandes lotes de trabajo a equipos y comités especiales para

aprobación y procesamiento y luego esperando respuestas

En cambio, necesitamos que todos en nuestro flujo de valor encuentren y solucionen problemas en su área de control como parte de nuestro trabajo diario. Al hacer esto, empujamos responsabilidades de calidad y seguridad y toma de decisiones donde el trabajo se realiza, en lugar de depender de aprobaciones de distancia ejecutivos

Utilizamos revisiones por pares de nuestros cambios propuestos para obtener lo que sea Se necesita garantía de que nuestros cambios funcionarán según lo diseñado. Nosotros Automatizar la mayor parte del control de calidad que generalmente realiza un control de calidad o departamento de seguridad de la información como sea posible. En lugar de desarrolladores necesitando solicitar o programar una prueba para que se ejecute, estas pruebas pueden ser realizado bajo demanda, permitiendo a los desarrolladores probar rápidamente sus propios codificar e incluso implementar esos cambios en la producción ellos mismos.

Al hacer esto, realmente hacemos de la calidad la responsabilidad de todos como opuesto a que sea responsabilidad exclusiva de un departamento separado. La seguridad de la información no es solo el trabajo de la Seguridad de la Información, sino también la disponibilidad no es simplemente el trabajo de Operaciones.

Hacer que los desarrolladores compartan la responsabilidad de la calidad de los sistemas. construyen no solo mejoran los resultados sino que también aceleran el aprendizaje. Esto es especialmente importante para los desarrolladores, ya que generalmente son equipo que está más alejado del cliente. Gary Gruver observa: "Es imposible que un desarrollador aprenda algo cuando alguien les grita por algo que rompieron hace seis meses, eso es ¿Por qué necesitamos proporcionar comentarios a todos lo más rápido posible, en minutos, no meses ".

HABILITAR OPTIMIZAR PARA BAJAR CENTROS DE TRABAJO

En la década de 1980, los principios de Diseño para la Manufacturabilidad buscaban diseñar piezas y procesos para que los productos terminados puedan crearse con El costo más bajo, la más alta calidad y el flujo más rápido. Ejemplos incluyen diseñando partes que son muy asimétricas para evitar que

colocarse al revés, y diseñar sujetadores de tornillo para que estén
imposible de apretar demasiado.

Esto fue una desviación de cómo se hacía típicamente el diseño, que
centrado en los clientes externos pero pasado por alto internos
partes interesadas, como las personas que realizan la fabricación.

Lean define dos tipos de clientes para los que debemos diseñar:
cliente externo (que probablemente paga por el servicio que somos
entrega) y el cliente interno (que recibe y procesa el
trabajar inmediatamente después de nosotros). Según Lean, nuestro más importante
El cliente es nuestro siguiente paso aguas abajo. Optimizando nuestro trabajo para ellos
requiere que tengamos empatía por sus problemas para mejorar
Identificar los problemas de diseño que impiden un flujo rápido y suave.

En el flujo de valor tecnológico, optimizamos para el trabajo posterior
centros diseñando para operaciones, donde operacional no funcional
requisitos (por ejemplo, arquitectura, rendimiento, estabilidad, capacidad de prueba,
configurabilidad y seguridad) tienen una prioridad tan alta como las características del usuario.

Al hacer esto, creamos calidad en la fuente, lo que probablemente resulte en un conjunto de
requisitos no funcionales codificados que podemos integrar de forma proactiva
en cada servicio que construimos.

CONCLUSIÓN

Crear comentarios rápidos es fundamental para lograr calidad, confiabilidad y
seguridad en el flujo de valor tecnológico. Hacemos esto al ver los problemas como
ocurren, pululando y resolviendo problemas para construir nuevos conocimientos,
acercando la calidad a la fuente y optimizando continuamente para
Centros de trabajo aguas abajo.

Las prácticas específicas que permiten un flujo rápido en la secuencia de valor de DevOps
se presentan en la Parte IV. En el próximo capítulo, presentamos la Tercera Vía:
Los principios de retroalimentación

† El Dr. Spear extendió su trabajo para explicar los éxitos duraderos de otras organizaciones, como el proveedor de Toyota network, Alcoa, y el Programa de Propulsión de Energía Nuclear de la Marina de los EE. UU.

‡ En algunas de sus plantas, Toyota ha pasado a usar un botón Andon.

§ Sorprendentemente, cuando el número de tirones del cable Andon disminuye, los gerentes de planta disminuirán las tolerancias para obtener un aumento en el número de tirones de cable Andon para continuar permitiendo más aprendizajes y mejoras y para detectar señales de falla cada vez más débiles.

¶ En la década de 1700, el gobierno británico participó en un espectacular ejemplo de mando y control burocrático de arriba hacia abajo, que resultó notablemente ineficaz. En ese momento, Georgia todavía era una colonia, y a pesar del hecho de que los británicos

Página 75

el gobierno estaba a tres mil millas de distancia y carecía de conocimiento de primera mano sobre la química local de la tierra, la rocosa topografía, accesibilidad al agua y otras condiciones, trató de planificar toda la economía agrícola de Georgia. los
Los resultados del intento fueron pésimos y dejaron a Georgia con los niveles más bajos de prosperidad y población en los trece colonias

4 Los principios de la tercera vía: *Aprendizaje continuo y experimentación*

Mientras que First Way aborda el flujo de trabajo de izquierda a derecha y el Second Way aborda la retroalimentación recíproca rápida y constante de de derecha a izquierda, la Tercera Vía se enfoca en crear una cultura de continuo aprendizaje y experimentación. Estos son los principios que permiten creación constante de conocimiento individual, que luego se convierte en equipo y conocimiento organizacional.

En operaciones de fabricación con calidad y seguridad sistémicas. problemas, el trabajo suele estar rígidamente definido y aplicado. Por ejemplo, en la planta de GM Fremont descrita en el capítulo anterior, los trabajadores tenía poca capacidad para integrar mejoras y aprendizajes en sus trabajo diario, con sugerencias de mejora "apto para enfrentarse a una pared de ladrillos de indiferencia ".

En estos entornos, a menudo también existe una cultura de miedo y baja

confianza, donde los trabajadores que cometen errores son castigados, y aquellos que hacer sugerencias o señalar problemas son vistos como denunciantes y alborotadores. Cuando esto ocurre, el liderazgo es activamente suprimir, incluso castigar, aprender y mejorar, perpetuar problemas de calidad y seguridad.

Por el contrario, las operaciones de fabricación de alto rendimiento requieren y Promover activamente el aprendizaje: en lugar de definir rígidamente el trabajo, el el sistema de trabajo es dinámico, con trabajadores de línea que realizan experimentos en su trabajo diario para generar nuevas mejoras, habilitadas por rigurosos estandarización de los procedimientos de trabajo y documentación de los resultados.

En el flujo de valor de la tecnología, nuestro objetivo es crear una alta confianza cultura, reforzando que todos somos aprendices de por vida que debemos tomar riesgos en nuestro trabajo diario Aplicando un enfoque científico a ambos procesos mejora y desarrollo de productos, aprendemos de nuestros éxitos y fracasos, identificando qué ideas no funcionan y reforzando esas Esto hace. Además, cualquier aprendizaje local se convierte rápidamente en global. mejoras, para que las nuevas técnicas y prácticas puedan ser utilizadas por el organización entera.

Reservamos tiempo para la mejora del trabajo diario y para avanzar acelerar y asegurar el aprendizaje Constantemente introducimos el estrés en nuestros sistemas para forzar la mejora continua. Incluso simulamos y inyectar fallas en nuestros servicios de producción bajo condiciones controladas para Aumentar nuestra resistencia.

Al crear este sistema continuo y dinámico de aprendizaje, permitimos equipos para adaptarse rápida y automáticamente a un cambio constante entorno, que en última instancia nos ayuda a ganar en el mercado.

HABILITAR EL APRENDIZAJE ORGANIZACIONAL Y UNA CULTURA DE SEGURIDAD

Cuando trabajamos dentro de un sistema complejo, por definición es imposible
para que podamos predecir perfectamente todos los resultados de cualquier acción que tomemos. Esta
es lo que contribuye a resultados inesperados, o incluso catastróficos, y
accidentes en nuestro trabajo diario, incluso cuando tomamos precauciones y trabajamos
cuidadosamente.

Cuando estos accidentes afectan a nuestros clientes, buscamos entender por qué
sucedió. La causa raíz a menudo se considera un error humano, y el
Una respuesta administrativa muy común es "nombre, culpa y vergüenza"
La persona que causó el problema. † Y, ya sea sutil o explícitamente,
la gerencia insinúa que la persona culpable de cometer el error
ser castigado. Luego crean más procesos y aprobaciones para evitar
el error vuelve a suceder.

Dr. Sidney Dekker, quien codificó algunos de los elementos clave de seguridad.
cultura y acuñó el término *cultura justa* , escribió: "Respuestas a incidentes
y los accidentes que se consideran injustos pueden impedir las investigaciones de seguridad,

Página 79

Promover el miedo en lugar de la atención plena en las personas que son críticas para la seguridad.
trabajar, hacer que las organizaciones sean más burocráticas en lugar de tener más cuidado,
y cultivar el secreto profesional, la evasión y la autoprotección ".

Estos problemas son especialmente problemáticos en el flujo de valor de la tecnología.
—Nuestro trabajo casi siempre se realiza dentro de un sistema complejo, y
cómo la gerencia elige reaccionar ante fallas y accidentes conduce a una
cultura del miedo, lo que hace improbable que los problemas y el fracaso
las señales son reportadas alguna vez. El resultado es que los problemas permanecen ocultos.
hasta que ocurra una catástrofe.

El Dr. Ron Westrum fue uno de los primeros en observar la importancia de
cultura organizacional en seguridad y desempeño. Observó que en
organizaciones de salud, la presencia de culturas "generativas" fue uno
de los principales predictores de seguridad del paciente. El Dr. Westrum definió tres tipos
de Cultura:

Las organizaciones patológicas se caracterizan por grandes cantidades de
Miedo y amenaza. La gente suele acumular información, retenerla por
razones políticas, o distorsionarlo para que se vean mejor.
El fracaso a menudo está oculto.

Las organizaciones burocráticas se caracterizan por reglas y procesos,
a menudo para ayudar a departamentos individuales a mantener su "territorio". El fracaso es
procesado a través de un sistema de juicio, lo que resulta en
castigo o justicia y misericordia.

Las organizaciones generativas se caracterizan por buscar activamente y
compartir información para permitir mejor a la organización lograr su
misión. Las responsabilidades se comparten en todo el flujo de valor,
y el fracaso da como resultado una reflexión y una investigación genuina .

Figura 8: *El modelo de tipología organizacional de Westrum: cómo organizaciones procesan información (Fuente: Ron Westrum, "Una tipología de cultura organizacional ", BMJ Quality & Safety 13, no. 2 (2004), doi: 10.1136 / qshc.2003.009522.)*

Tal como el Dr. Westrum encontró en las organizaciones de atención médica, un grupo de alta confianza, la cultura generativa también predijo el desempeño de TI y organizacional en flujos de valor tecnológico.

En el flujo de valor tecnológico, establecemos las bases de un cultura generativa esforzándose por crear un sistema de trabajo seguro. Cuando ocurren accidentes y fallas, en lugar de buscar errores humanos, nosotros Busque cómo podemos rediseñar el sistema para evitar el accidente. sucediendo de nuevo

Por ejemplo, podemos realizar una autopsia sin culpa después de cada incidente para obtener la mejor comprensión de cómo ocurrió el accidente y acordar cuáles son las mejores contramedidas para mejorar el sistema, idealmente evitando que el problema vuelva a ocurrir y permitiendo una detección y recuperación más rápidas.

Al hacer esto, creamos aprendizaje organizacional. Como Bethany Macri, un ingeniero en Etsy que dirigió la creación de la herramienta Morgue para ayudar con grabación de autopsias, declaró: "Al eliminar la culpa, eliminas temor; al eliminar el miedo, habilitas la honestidad; y la honestidad permite prevención."

El Dr. Spear observa que el resultado de eliminar la culpa y poner El aprendizaje organizacional en su lugar es que "las organizaciones se vuelven siempre más autodiagnóstico y superación personal, experto en detectar problemas [y] resolviéndolos ".

Página 81

Muchos de estos atributos también fueron descritos por el Dr. Senge como atributos de organizaciones de aprendizaje. En *La quinta disciplina,* escribió que estos características ayudan a los clientes, aseguran la calidad, crean competitividad ventaja y una fuerza laboral activa y comprometida, y descubrir el verdad.

INSTITUCIONALIZAR LA MEJORA DE TRABAJO DIARIO

Los equipos a menudo no pueden o no quieren mejorar los procesos que operar dentro de El resultado no es solo que continúen sufriendo de sus problemas actuales, pero su sufrimiento también empeora con el tiempo. Mike Rother observó en *Toyota Kata* que en ausencia de mejoras, los procesos no permanecen igual, debido al caos y Entropía, los procesos en realidad se degradan con el tiempo.

En el flujo de valor de la tecnología, cuando evitamos solucionar nuestros problemas, confiando en soluciones diarias, nuestros problemas y deudas técnicas se acumula hasta que todo lo que estamos haciendo es realizar soluciones alternativas, tratando de evitar desastres, sin ciclos sobrantes para realizar un trabajo productivo. Esto es por qué Mike Orzen, autor de *Lean IT* , observó: "Aún más importante que el trabajo diario es la mejora del trabajo diario ".

Mejoramos el trabajo diario reservando explícitamente el tiempo para pagar deuda técnica, corregir defectos y refactorizar y mejorar áreas problemáticas de nuestro código y entornos: lo hacemos reservando ciclos en cada intervalo de desarrollo, o programando *bombardeos kaizen* , que son períodos en que los ingenieros se autoorganizan en equipos para trabajar en la fijación de cualquier problema que quieren

El resultado de estas prácticas es que todos encuentran y solucionan problemas en su área de control, todo el tiempo, como parte de su trabajo diario. Cuando finalmente solucionamos los problemas diarios que hemos trabajado durante meses (o años), podemos erradicar de nuestro sistema los problemas menos obvios. Al detectar y responder a estas señales de falla cada vez más débiles, solucionamos problemas cuando no solo es más fácil y económico sino también cuando Las consecuencias son menores.

Considere el siguiente ejemplo que mejoró la seguridad laboral en
Alcoa, un fabricante de aluminio con ingresos de $ 7.8 mil millones en 1987.
La fabricación de aluminio requiere calor extremadamente alto, presiones altas,
y productos químicos corrosivos. En 1987, Alcoa tenía un historial de seguridad aterrador,
con un 2% de los noventa mil empleados empleados heridos cada uno
año: son siete lesiones por día. Cuando Paul O'Neill comenzó como CEO,
su primer objetivo era tener cero lesiones para empleados, contratistas y
visitantes

O'Neill quería ser notificado dentro de las veinticuatro horas de que alguien
heridos en el trabajo, no para castigar, sino para asegurar y promover eso
se estaban generando e incorporando aprendizajes para crear un lugar más seguro
lugar de trabajo. En el transcurso de diez años, Alcoa redujo su tasa de lesiones
en un 95%.

La reducción en las tasas de lesiones permitió a Alcoa centrarse en los más pequeños
problemas y señales de falla más débiles, en lugar de notificar solo a O'Neill
cuando ocurrieron lesiones, comenzaron a informar también de cualquier llamada cercana.
Al hacer esto, mejoraron la seguridad en el lugar de trabajo sobre el posterior
veinte años y tener uno de los registros de seguridad más envidiable en el
industria.

Como escribe el Dr. Spear, “los alcoanos gradualmente dejaron de trabajar alrededor del
dificultades, inconvenientes e impedimentos que experimentaron.
Afrontamiento, lucha contra incendios y recuperación se fueron reemplazando gradualmente.
en toda la organización mediante una dinámica de identificación de oportunidades
para la mejora de procesos y productos. Como esas oportunidades eran
identificado y los problemas fueron investigados, los focos de ignorancia
que reflejaron se convirtieron en pepitas de conocimiento ". Esta
ayudó a dar a la compañía una mayor ventaja competitiva en el
mercado.

Del mismo modo, en el flujo de valor de la tecnología, a medida que hacemos nuestro sistema de
trabajamos de manera más segura, encontramos y solucionamos problemas de señales de falla cada vez más débiles
Por ejemplo, inicialmente podemos realizar autopsias sin culpa solo para
incidentes que impactan al cliente. Con el tiempo, podemos realizarlos para
incidentes de menor impacto en el equipo y casi accidentes también.

TRANSFORME LOS DESCUBRIMIENTOS LOCALES EN MEJORAS GLOBALES

Cuando se descubren nuevos aprendizajes localmente, también debe haber algunos mecanismo para permitir que el resto de la organización use y se beneficie de ese conocimiento En otras palabras, cuando los equipos o individuos tienen experiencias que crean experiencia, nuestro objetivo es convertir ese tácito conocimiento (es decir, conocimiento que es difícil de transferir a otra persona) por escrito o verbalizando) en explícito, codificado conocimiento, que se convierte en la experiencia de otra persona a través de la práctica.

Esto asegura que cuando alguien más haga un trabajo similar, lo haga con la experiencia acumulativa y colectiva de todos en el organización que alguna vez ha hecho el mismo trabajo. Un ejemplo notable de convertir el conocimiento local en conocimiento global es la Marina de los EE. UU. Programa de propulsión de energía nuclear (también conocido como "NR" para "Naval Reactores "), que tiene más de 5,700 años de operación sin un Víctima relacionada con un solo reactor o escape de radiación.

El NR es conocido por su intenso compromiso con los procedimientos programados. y trabajo estandarizado, y la necesidad de informes de incidentes para cualquier desviación del procedimiento u operaciones normales para acumular aprendizajes, no importa cuán pequeña sea la señal de falla: constantemente actualizar procedimientos y diseños de sistemas basados en estos aprendizajes.

El resultado es que cuando una nueva tripulación se embarca en su primera despliegue, ellos y sus oficiales se benefician del colectivo conocimiento de 5.700 años de reactores sin accidentes. Igualmente impresionante es que sus propias experiencias en el mar se agregarán a este colectivo conocimiento, ayudando a futuras tripulaciones a cumplir con seguridad sus propias misiones.

En el flujo de valor de la tecnología, debemos crear mecanismos similares para crear conocimiento global, como hacer todos nuestros post-mortem irreprochables informes que pueden buscar los equipos que intentan resolver problemas similares y crear repositorios de código fuente compartido que abarcan todo organización, donde el código compartido, las bibliotecas y las configuraciones que encarnar el mejor conocimiento colectivo de toda la organización puede ser

Utilizado fácilmente. Todos estos mecanismos ayudan a convertir la experiencia individual en artefactos que el resto de la organización puede usar.

PATRONES DE RESILIENCIA INYECTOS EN NUESTROS TRABAJO DIARIO

Las organizaciones de fabricación de bajo rendimiento se amortiguan de las interrupciones de muchas maneras; en otras palabras, se acumulan o agregan flacidez Por ejemplo, para reducir el riesgo de que un centro de trabajo esté inactivo (debido a inventario que llega tarde, inventario que tuvo que desecharse, etc.), los gerentes pueden optar por almacenar más inventario en cada centro de trabajo. Sin embargo, ese búfer de inventario también aumenta WIP, que tiene todo tipo de resultados no deseados, como se discutió anteriormente.

Del mismo modo, para reducir el riesgo de que un centro de trabajo caiga debido a falla de maquinaria, los gerentes pueden aumentar la capacidad comprando más equipo de capital, contratando a más personas o incluso aumentando el espacio de piso. Todas estas opciones aumentan los costos.

En contraste, los de alto rendimiento logran los mismos resultados (o mejores) al Mejorando las operaciones diarias, introduciendo continuamente tensión para elevar rendimiento, así como la ingeniería de mayor resistencia en su sistema.

Considere un experimento típico en uno de los colchones de Aisin Seiki Global fábricas, uno de los principales proveedores de Toyota. Supongamos que tuvieran dos líneas de producción, cada una capaz de producir cien unidades por día. En días lentos, enviarían toda la producción a una línea, experimentar con formas de aumentar la capacidad e identificar vulnerabilidades en su proceso, sabiendo que si sobrecarga la línea Si fallaba, podían enviar toda la producción a la segunda línea.

Por la experimentación incesante y constante en su trabajo diario, ellos fueron capaces de aumentar continuamente la capacidad, a menudo sin agregar equipo nuevo o contratar más personas. El patrón emergente que

Los resultados de este tipo de rituales de mejora no solo mejoran
rendimiento pero también mejora la capacidad de recuperación, porque la organización es
siempre en estado de tensión y cambio. Este proceso de aplicar estrés
para aumentar la resiliencia fue nombrado *antifragilidad* por autor y riesgo
analista Nassim Nicholas Taleb.

En el flujo de valor de la tecnología, podemos introducir el mismo tipo de
tensión en nuestros sistemas buscando reducir siempre el liderazgo de implementación
veces, aumentar la cobertura de prueba, disminuir los tiempos de ejecución de prueba e incluso

rediseñando si es necesario para aumentar la productividad del desarrollador o
Aumentar la fiabilidad.

También podemos realizar ejercicios de *Game Day* , donde ensayamos grandes
fallas de escala, como apagar centros de datos completos. O podemos inyectar
fallas cada vez mayores en el entorno de producción (como el
famoso "Chaos Monkey" de Netflix, que mata aleatoriamente procesos y
calcular servidores en producción) para garantizar que somos tan resistentes como nosotros
quiero ser.

LÍDERES REFORZAN UNA CULTURA DE APRENDIZAJE

Tradicionalmente, se esperaba que los líderes fueran responsables de establecer
objetivos, asignando recursos para lograr esos objetivos, y
establecer la combinación correcta de incentivos. Los líderes también establecen
El tono emocional para las organizaciones que dirigen. En otras palabras,
líderes liderados por "tomar todas las decisiones correctas".

Sin embargo, existe evidencia significativa que muestra que la grandeza no es
logrado por los líderes que toman todas las decisiones correctas; en cambio,
El papel es crear las condiciones para que su equipo pueda descubrir la grandeza en
su trabajo diario En otras palabras, crear grandeza requiere tanto
líderes y trabajadores, cada uno de los cuales depende mutuamente de cada uno
otro.

Jim Womack, autor de *Gemba Walks,* describió el complemento
relación de trabajo y respeto mutuo que debe ocurrir entre
líderes y trabajadores de primera línea. Según Womack, esta relación
es necesario porque ninguno de los dos puede resolver problemas solos: los líderes no
lo suficientemente cerca del trabajo, que se requiere para resolver cualquier problema, y
los trabajadores de primera línea no tienen el contexto organizacional más amplio o
autoridad para realizar cambios fuera de su área de trabajo. §

Los líderes deben elevar el valor del aprendizaje y el problema disciplinado
resolviendo Mike Rother formalizó estos métodos en lo que él llama
entrenando kata. El resultado es uno que refleja el método científico,
donde declaramos explícitamente nuestros objetivos del Norte verdadero, como "sostener cero
accidentes "en el caso de Alcoa, o" doble rendimiento en un año "en
El caso de Aisin.

Estos objetivos estratégicos luego informan la creación de iterativos, a corto plazo
objetivos, que se conectan en cascada y luego se ejecutan estableciendo el objetivo
condiciones en el flujo de valor o en el nivel del centro de trabajo (p. ej., “reducir plomo
tiempo en un 10% dentro de las próximas dos semanas ").

Estas condiciones objetivo enmarcan el experimento científico: explícitamente
Indicar el problema que buscamos resolver, nuestra hipótesis de cómo
la contramedida propuesta lo resolverá, nuestros métodos para probar eso
hipótesis, nuestra interpretación de los resultados y nuestro uso de los aprendizajes para
informar la próxima iteración.

El líder ayuda a entrenar a la persona que realiza el experimento con
preguntas que pueden incluir:

¿Cuál fue tu último paso y qué pasó?

¿Qué aprendiste?

¿Cuál es tu condición ahora?

¿Cuál es su próxima condición objetivo?

¿En qué obstáculo estás trabajando ahora?

cual es tu siguiente paso?

¿Cuál es su resultado esperado?

¿Cuándo podemos verificar?

Este enfoque de resolución de problemas en el que los líderes ayudan a los trabajadores a ver y Resolver problemas en su trabajo diario es el núcleo del Toyota Sistema de producción, de organizaciones de aprendizaje, el Kata de mejora, y organizaciones de alta confiabilidad. Mike Rother observa que él ve Toyota "como una organización definida principalmente por el comportamiento único rutinas que enseña continuamente a todos sus miembros ".

En el flujo de valor tecnológico, este enfoque científico e iterativo El método guía todos nuestros procesos de mejora interna, pero también cómo Realizamos experimentos para asegurar que los productos que construimos realmente Ayudar a nuestros clientes internos y externos a alcanzar sus objetivos.

CONCLUSIÓN

Los principios de la Tercera Vía abordan la necesidad de valorar Aprendizaje organizacional, que permite una alta confianza y un alcance de límites entre funciones, aceptando que las fallas siempre ocurrirán en complejos sistemas, y hacer aceptable hablar sobre problemas para que podamos Crear un sistema de trabajo seguro. También requiere institucionalizar el mejora del trabajo diario, convirtiendo los aprendizajes locales en globales aprendizajes que pueden ser utilizados por toda la organización, así como inyectando tensión continuamente en nuestro trabajo diario.

Aunque fomentando una cultura de aprendizaje y experimentación continua es el principio de la Tercera Vía, también está entretejido en la Primera y Segundas maneras. En otras palabras, mejorar el flujo y la retroalimentación requiere un

enfoque iterativo y científico que incluye el encuadre de un objetivo condición, estableciendo una hipótesis de lo que nos ayudará a llegar allí, diseñando y realizando experimentos y evaluando los resultados.

Los resultados no solo son un mejor rendimiento, sino también un aumento resiliencia, mayor satisfacción laboral y mejor organización adaptabilidad.

PARTE I CONCLUSIÓN

En la Parte I del *Manual de DevOps,* analizamos varios movimientos en la historia que ayudaron a conducir al desarrollo de DevOps. También observamos los tres principios principales que forman la base para organizaciones exitosas de DevOps: los principios de Flow, Feedback,

y aprendizaje continuo y experimentación. En la Parte II, comenzaremos para ver cómo iniciar un movimiento DevOps en su organización.

† El patrón de "nombre, culpa y vergüenza" es parte de la teoría de la mala manzana criticada por el Dr. Sydney Dekker y ampliamente discutido en su libro *The Field Guide to Understanding Human Error.*

‡ Es sorprendente, instructivo y realmente conmovedor ver el nivel de convicción y pasión que Paul O'Neill tiene sobre el Los líderes de responsabilidad moral deben crear seguridad en el lugar de trabajo.

§ Los líderes son responsables del diseño y la operación de procesos en un nivel superior de agregación donde otros tienen menos perspectiva y autoridad.

Parte

Introducción

¿Cómo decidimos dónde comenzar un DevOps?
transformación en nuestra organización? Quien necesita ser
¿involucrado? ¿Cómo debemos organizar nuestros equipos, proteger
su capacidad de trabajo y maximizar sus posibilidades de
tener éxito? Estas son las preguntas que pretendemos responder en
Parte II del *Manual de DevOps* .

En los siguientes capítulos, veremos los
proceso de iniciar una transformación DevOps. Empezamos
evaluando los flujos de valor en nuestra organización,
localizar un buen lugar para comenzar y formar una estrategia para
crear un equipo de transformación dedicado con específico
objetivos de mejora y eventual expansión. Para cada
transformando el flujo de valor, identificamos el trabajo
se realiza y luego mira el diseño organizacional
estrategias y arquetipos organizacionales que mejor
Apoyar los objetivos de transformación.

Los enfoques principales en estos capítulos incluyen:

Seleccionar con qué flujos de valor comenzar

Comprender el trabajo que se realiza en nuestro candidato
flujos de valor

Diseñando nuestra organización y arquitectura con
Ley de Conway en mente

Permitir resultados orientados al mercado a través de más
colaboración efectiva entre funciones en todo
el flujo de valor

Proteger y habilitar a nuestros equipos.

Comenzar cualquier transformación está llena de incertidumbre: nosotros
están trazando un viaje hacia un estado final ideal, pero donde
Prácticamente todos los pasos intermedios son desconocidos. Estas
los siguientes capítulos tienen la intención de proporcionar un proceso de pensamiento
para guiar nuestras decisiones, proporcionar pasos procesables que podamos
tomar e ilustrar casos de estudio como ejemplos.

'

5 Seleccionar cuál Flujo de valor para Empezar con

Elegir un flujo de valor para la transformación de DevOps merece cuidado consideración. El flujo de valor que elegimos no solo dicta la dificultad de nuestra transformación, pero también dicta quién estará involucrado en la transformación. Afectará cómo debemos organizarnos en equipos y cómo podemos habilitar mejor la equipos e individuos en ellos.

Michael Rembetsy notó otro desafío, quien ayudó a dirigir DevOps transformación como Director de Operaciones en Etsy en 2009. Observó: "Nosotros debemos elegir nuestros proyectos de transformación con cuidado, cuando estamos en problemas, no conseguir muchas fotos Por lo tanto, debemos elegir cuidadosamente y luego proteger esos proyectos de mejora que mejorarán más el estado de nuestra organización ".

Examinemos cómo el equipo de Nordstrom comenzó su transformación DevOps iniciativa en 2013, que Courtney Kissler, su vicepresidenta de comercio electrónico y tienda Tecnologías, descritas en la Cumbre empresarial de DevOps en 2014 y 2015.

Fundada en 1901, Nordstrom es un minorista de moda líder que se centra en entregando la mejor experiencia de compra posible a sus clientes. En 2015, Nordstrom tuvo ingresos anuales de $ 13.5 mil millones.

El escenario para el viaje DevOps de Nordstrom probablemente se estableció en 2011 durante uno de sus reuniones anuales de la junta directiva. Ese año, uno de los temas estratégicos. se discutió la necesidad de aumentar los ingresos en línea. Estudiaron la difícil situación de Blockbusters, Borders y Barnes & Nobles, que demostraron la extrema consecuencias cuando los minoristas tradicionales llegaron tarde creando e- competitividad capacidades comerciales: estas organizaciones estaban claramente en riesgo de perder sus posicionarse en el mercado o incluso cerrar completamente el negocio. †

En ese momento, Courtney Kissler era el director senior de Systems Delivery y

Tecnología de venta, responsable de una parte importante de la tecnología.
organización, incluidos sus sistemas en la tienda y el sitio de comercio electrónico en línea. Como
Kissler describió: “En 2011, la organización tecnológica de Nordstrom fue muy
mucho optimizado para el costo: habíamos subcontratado muchas de nuestras funciones tecnológicas,
tuvimos un ciclo de planificación anual con grandes lotes, lanzamientos de software 'en cascada'.
A pesar de que tuvimos una tasa de éxito del 97% de alcanzar nuestro cronograma, presupuesto y alcance

objetivos, estábamos mal equipados para lograr lo que la estrategia comercial de cinco años
requerido por nosotros, ya que Nordstrom comenzó a optimizar la velocidad en lugar de simplemente
optimizando el costo ".

Kissler y el equipo de gestión de tecnología de Nordstrom tuvieron que decidir dónde
comenzar sus esfuerzos iniciales de transformación. No querían causar agitación en el
todo el sistema En cambio, querían centrarse en áreas muy específicas del negocio.
para que puedan experimentar y aprender. Su objetivo era demostrar temprano
gana, lo que daría a todos la confianza de que estas mejoras podrían ser
replicado en otras áreas de la organización. Cómo exactamente eso se lograría
Aún era desconocido.

Se centraron en tres áreas: la aplicación móvil del cliente, su tienda
sistemas de restaurantes y sus propiedades digitales. Cada una de estas áreas tenía negocios
objetivos que no se estaban cumpliendo; por lo tanto, fueron más receptivos a considerar un
diferente forma de trabajar Las historias de los dos primeros se describen a continuación.

La aplicación móvil Nordstrom había experimentado un inicio desfavorable. Como
Kissler dijo: "Nuestros clientes estaban extremadamente frustrados con el producto, y nosotros
tenía críticas negativas uniformes cuando lo lanzamos en la App Store. Peor aún, el
la estructura y los procesos existentes (también conocidos como "el sistema") habían diseñado sus procesos
para que solo puedan lanzar actualizaciones dos veces al año ". En otras palabras, cualquier solución
a la aplicación tendría que esperar meses para llegar al cliente.

Su primer objetivo era habilitar lanzamientos más rápidos o bajo demanda, proporcionando más rápido
iteración y la capacidad de responder a los comentarios de los clientes. Crearon un
equipo de producto dedicado que se dedicó exclusivamente a brindar soporte para dispositivos móviles
aplicación, con el objetivo de permitir que ese equipo pueda
implementar, probar y entregar valor al cliente. Al hacer esto, no
ya no tiene que depender y coordinarse con los puntajes de otros equipos dentro
Nordstrom Además, pasaron de la planificación una vez al año a un
Proceso de planificación continua. El resultado fue una única acumulación de trabajo priorizada
para la aplicación móvil basada en la necesidad del cliente, ya no existían conflictos

prioridades cuando el equipo tuvo que soportar múltiples productos.

Durante el año siguiente, eliminaron las pruebas como una fase separada del trabajo, en cambio, integrándolo en el trabajo diario de todos. [‡] Duplicaron las características siendo entregado por mes y redujo a la mitad el número de defectos, creando un éxito Salir.

Su segunda área de enfoque fueron los sistemas que respaldan su *Café Bistro* en la tienda. restaurantes. A diferencia del flujo de valor de la aplicación móvil donde la necesidad comercial era Reduzca el tiempo de comercialización y aumente el rendimiento de las funciones, la empresa necesita aquí fue para disminuir el costo y aumentar la calidad. En 2013, Nordstrom había completado once "re-conceptos de restaurante" que requirieron cambios en la tienda aplicaciones, causando una serie de incidentes que impactan al cliente. Inquietantemente

Page 95

habían planeado cuarenta y cuatro más de estos re-conceptos para 2014, cuatro veces más tantos como en el año anterior.

Como Kissler declaró: “Uno de nuestros líderes empresariales sugirió que tripliquemos nuestro equipo tamaño para manejar estas nuevas demandas, pero propuse que tuviéramos que dejar de tirar más cuerpos ante el problema y, en cambio, mejorar la forma en que trabajamos ".

Pudieron identificar áreas problemáticas, como en su ingesta de trabajo y procesos de implementación, que es donde enfocaron sus esfuerzos de mejora. Pudieron reducir los tiempos de entrega de implementación de código en un 60% y reducir el Número de incidentes de producción 60% a 90%.

Estos éxitos dieron a los equipos la confianza de que los principios y prácticas de DevOps eran aplicables a una amplia variedad de flujos de valor. Kissler fue ascendido a vicepresidente de Comercio electrónico y tecnologías de tienda en 2014.

En 2015, Kissler dijo que para la venta o la tecnología orientada al cliente organización para permitir que la empresa cumpla con sus objetivos, "... necesitábamos aumentar productividad en todos nuestros flujos de valor de tecnología, no solo en unos pocos. En el nivel de gestión, creamos un mandato general para reducir los tiempos de ciclo en un 20% para todos los servicios orientados al cliente ".

Ella continuó: “Este es un desafío audaz. Tenemos muchos problemas en nuestro estado actual: los tiempos de proceso y ciclo no se miden de manera consistente equipos, ni son visibles. Nuestra primera condición objetivo requiere que ayudemos a todos nuestros los equipos miden, lo hacen visible y realizan experimentos para comenzar a reducir su tiempos de proceso, iteración por iteración ".

Kissler concluyó: "Desde una perspectiva de alto nivel, creemos que las técnicas tales como mapeo de flujo de valor, reduciendo nuestros tamaños de lote hacia el flujo de una sola pieza, como así como el uso de la entrega continua y microservicios nos llevará a nuestro deseado estado. Sin embargo, mientras todavía estamos aprendiendo, estamos seguros de que nos dirigimos la dirección correcta, y todos saben que este esfuerzo cuenta con el apoyo de los más altos niveles de gestión ".

En este capítulo, se presentan varios modelos que nos permitirán replicar el procesos de pensamiento que el equipo de Nordstrom utilizó para decidir qué flujos de valor Empezar con. Evaluaremos nuestros flujos de valor candidatos de muchas maneras, incluyendo ya sea un servicio *greenfield* o *brownfield* , un *sistema de compromiso* o un *sistema de registro* . También calcularemos el balance riesgo / recompensa de la transformación y evaluar el nivel probable de resistencia que podemos obtener de los equipos en los que trabajaríamos con.

GREENFIELD VS. SERVICIOS BROWNFIELD

A menudo categorizamos nuestros servicios o productos de software como greenfield o brownfield Estos términos se usaron originalmente para la planificación urbana y la construcción. proyectos El desarrollo greenfield es cuando construimos en tierras subdesarrolladas. El desarrollo de Brownfield es cuando construimos en terrenos que anteriormente se usaban para fines industriales, potencialmente contaminados con residuos peligrosos o contaminación. En el desarrollo urbano, muchos factores pueden hacer que los proyectos greenfield sean más simples que proyectos brownfield: no hay estructuras existentes que necesiten ser demolidas tampoco hay materiales tóxicos que deban eliminarse.

En tecnología, un proyecto greenfield es un nuevo proyecto o iniciativa de software, probablemente en las primeras etapas de planificación o implementación, donde construimos nuestras aplicaciones e infraestructura de nuevo, con pocas limitaciones. Comenzando con un software greenfield el proyecto puede ser más fácil, especialmente si el proyecto ya está financiado y un equipo está ya sea creado o ya está en su lugar. Además, porque estamos empezando Desde cero, podemos preocuparnos menos por las bases de código existentes, los procesos y los equipos.

Los proyectos Greenfield DevOps a menudo son pilotos para demostrar la viabilidad del público o nubes privadas, automatización de despliegue piloto y herramientas similares. Un ejemplo de un proyecto DevOps greenfield es el producto Hosted LabVIEW en 2009 en National

Instruments, una organización de treinta años con cinco mil empleados y $ 1
mil millones en ingresos anuales. Para llevar este producto al mercado rápidamente, un nuevo equipo
fue creado y se le permitió operar fuera de los procesos de TI existentes y
explorar el uso de nubes públicas. El equipo inicial incluyó una aplicación
arquitecto, arquitecto de sistemas, dos desarrolladores, un desarrollador de sistemas de automatización, un
jefe de operaciones y dos empleados de operaciones en alta mar. Mediante el uso de prácticas DevOps,
pudieron entregar LabVIEW Hospedado al mercado en la mitad del tiempo de su
introducciones normales de productos.

En el otro extremo del espectro están los proyectos Brownfield DevOps, estos son
Productos o servicios existentes que ya están sirviendo a clientes y tienen
potencialmente ha estado en funcionamiento durante años o incluso décadas. Brownfield proyecta a menudo
vienen con cantidades significativas de deuda técnica, como no tener prueba
automatización o ejecución en plataformas no compatibles. En el ejemplo de Nordstrom
presentado anteriormente en este capítulo, tanto los sistemas de restaurante en la tienda como e-
Los sistemas de comercio eran proyectos brownfield.

Aunque muchos creen que DevOps es principalmente para proyectos nuevos, DevOps
se ha utilizado para transformar con éxito proyectos brownfield de todo tipo. De hecho,
Más del 60% de las historias de transformación compartidas en la Cumbre de DevOps Enterprise
en 2014 fueron para proyectos brownfield. En estos casos, hubo una gran
brecha de rendimiento entre lo que el cliente necesitaba y lo que la organización
actualmente estaba entregando, y las transformaciones de DevOps crearon tremendas
beneficio comercial

Page 97

De hecho, uno de los hallazgos en el *Informe sobre el estado de DevOps de 2015 confirmó* que
la edad de la aplicación no fue un predictor significativo del rendimiento; en lugar,
lo que predijo el rendimiento fue si la aplicación fue diseñada (o
podría ser rediseñado) para la comprobabilidad y la capacidad de implementación.

Los equipos que apoyan proyectos brownfield pueden ser muy receptivos a experimentar
con DevOps, particularmente cuando existe una creencia generalizada de que
los métodos son insuficientes para lograr sus objetivos, y especialmente si hay un fuerte
sentido de urgencia en torno a la necesidad de mejora. §

Al transformar proyectos brownfield, podemos enfrentar impedimentos significativos y
problemas, especialmente cuando no existen pruebas automatizadas o cuando hay una
arquitectura acoplada que evita que pequeños equipos desarrollen, prueben y

desplegar código de forma independiente. Se discute cómo superamos estos problemas a lo largo de este libro.

Los ejemplos de transformaciones brownfield exitosas incluyen:

CSG (2013): en 2013, CSG International tuvo $ 747 millones en ingresos y más de 3,500 empleados, lo que permite más de noventa mil agentes de servicio al cliente para proporcionar operaciones de facturación y atención al cliente a más de cincuenta millones de videos, clientes de voz y datos, ejecutando más de seis mil millones de transacciones e imprimiendo y el envío de más de setenta millones de estados de cuenta en papel cada mes. Su El alcance inicial de la mejora fue la impresión de facturas, uno de sus principales negocios, e involucró una aplicación COBOL mainframe y los veinte plataformas tecnológicas. Como parte de su transformación, comenzaron a actuar implementaciones diarias en un entorno similar a la producción, y duplicó el frecuencia de lanzamientos de clientes de dos veces al año a cuatro veces al año. Como Como resultado, aumentaron significativamente la confiabilidad de la aplicación y Reducción de los tiempos de implementación del código de dos semanas a menos de un día.

Etsy (2009): en 2009, Etsy tenía treinta y cinco empleados y generaba $ 87 millones en ingresos, pero después de que "apenas sobrevivieron a la temporada de compras navideñas" comenzaron a transformar prácticamente todos los aspectos de cómo la organización trabajó, eventualmente convirtiendo a la compañía en uno de los DevOps más admirados organizaciones y preparan el escenario para una exitosa OPI 2015.

CONSIDERE AMBOS SISTEMAS DE REGISTRO Y SISTEMAS DE COMPROMISO

La firma de investigación Gartner ha popularizado recientemente la noción de *TI bimodal* , refiriéndose al amplio espectro de servicios que soportan las empresas típicas. Dentro TI bimodal hay *sistemas de registro* , los sistemas tipo ERP que ejecutan nuestro negocios (p. ej., MRP, RRHH, sistemas de información financiera), donde la corrección de

las transacciones y los datos son primordiales; y *sistemas de compromiso* , que son sistemas orientados al cliente o empleados, como los sistemas de comercio electrónico y aplicaciones de productividad.

Los sistemas de registro suelen tener un ritmo de cambio más lento y a menudo tienen regulaciones y requisitos de cumplimiento (por ejemplo, SOX). Gartner llama a este tipo de sistemas "Tipo 1", donde la organización se centra en "hacerlo bien".

Los sistemas de compromiso generalmente tienen un ritmo de cambio mucho más alto para respaldar bucles de retroalimentación rápida que les permiten realizar experimentos para descubrir cómo satisfacer mejor las necesidades del cliente. Gartner llama a este tipo de sistemas "Tipo 2" donde la organización se enfoca en "hacerlo rápido".

Puede ser conveniente dividir nuestros sistemas en estas categorías; Sin embargo, nos saber que el conflicto central y crónico entre "hacerlo bien" y "hacerlo rápido" se puede romper con DevOps. Los datos de los informes del estado de DevOps de Puppet Labs —Siguiendo las lecciones de Lean Manufacturing — muestra que el alto rendimiento Las organizaciones pueden ofrecer simultáneamente mayores niveles de rendimiento y fiabilidad.

Además, debido a lo interdependientes que son nuestros sistemas, nuestra capacidad de hacer los cambios en cualquiera de estos sistemas están limitados por el sistema que es más difícil de cambiar con seguridad, que casi siempre es un sistema de registro.

Scott Prugh, vicepresidente de desarrollo de productos en CSG, observó: “Hemos adoptado un filosofía que rechaza la TI bimodal, porque cada uno de nuestros clientes merece Velocidad y calidad. Esto significa que necesitamos excelencia técnica, ya sea el equipo está apoyando una aplicación mainframe de 30 años, una aplicación Java o un aplicación movil."

En consecuencia, cuando mejoramos los sistemas brownfield, no solo debemos esforzarnos por reducir su complejidad y mejorar su fiabilidad y estabilidad, también debemos hacerlos más rápidos, seguros y fáciles de cambiar. Incluso cuando la nueva funcionalidad es agregados solo a los sistemas de interacción nuevos, a menudo causan confiabilidad problemas en los sistemas de registro brownfield en los que confían. Al hacer estos sistemas posteriores más seguros para cambiar, ayudamos a toda la organización más Alcanzar sus objetivos de forma rápida y segura.

COMIENCE CON EL MÁS SIMPÁTICO Y GRUPOS INNOVADORES

Dentro de cada organización, habrá equipos e individuos con una amplia gama. de actitudes hacia la adopción de nuevas ideas. Geoffrey A. Moore primero representado este espectro en forma del ciclo de vida de adopción de tecnología en *Crossing The*

Abismo , donde el abismo representa la dificultad clásica de alcanzar grupos más allá de los *innovadores* y los *primeros en adoptar* (verFigura 9).

En otras palabras, las nuevas ideas a menudo son rápidamente aceptadas por los innovadores y los primeros adoptantes, mientras que otros con actitudes más conservadoras se resisten a ellos (los *primeros mayoría* , *mayoría tardía* y *rezagados*). Nuestro objetivo es encontrar aquellos equipos que ya cree en la necesidad de los principios y prácticas de DevOps, y quién posee un deseo y capacidad demostrada para innovar y mejorar sus propios procesos. Idealmente, estos grupos serán entusiastas partidarios del viaje de DevOps.

Figura 9: *La curva de adopción de tecnología (Fuente: Moore y McKenna,* Crossing El abismo , *15.)*

Especialmente en las primeras etapas, no pasaremos mucho tiempo tratando de convertir el grupos más conservadores. En cambio, enfocaremos nuestra energía en crear éxitos con menos grupos de aversión al riesgo y construir nuestra base desde allí (un proceso que es discutido más adelante en la siguiente sección). Incluso si tenemos los niveles más altos de patrocinio ejecutivo, evitaremos el *enfoque* del *big bang* (es decir, comenzar en todas partes a la vez), eligiendo en cambio enfocar nuestros esfuerzos en algunas áreas de la organización, asegurando que esas iniciativas sean exitosas, y expandiéndose desde allí.¶

DEVOPS AMPLIADOS A TRAVÉS DE NUESTRA ORGANIZACIÓN

Independientemente de cómo alcancemos nuestro esfuerzo inicial, debemos demostrar las primeras victorias y Transmitir nuestros éxitos. Hacemos esto rompiendo nuestros objetivos de mejora más grandes en pequeños pasos incrementales. Esto no solo crea nuestras mejoras más rápido, sino que también también nos permite descubrir cuándo tomamos la decisión equivocada de flujo de valor: Al detectar nuestros errores temprano, podemos retroceder rápidamente e intentarlo nuevamente, haciendo diferentes decisiones armadas con nuestros nuevos aprendizajes.

A medida que generamos éxitos, nos ganamos el derecho de ampliar el alcance de nuestros DevOps iniciativa. Queremos seguir una secuencia segura que aumente metódicamente nuestros niveles de credibilidad, influencia y apoyo. La siguiente lista, adaptada de un curso. impartido por el Dr. Roberto Fernández, profesor de administración William F. Pounds en MIT, describe las fases ideales utilizadas por los agentes de cambio para construir y expandir su coalición y base de apoyo:

1. **Encuentra innovadores y primeros adoptantes:** al principio, enfocamos nuestro esfuerzos en equipos que realmente quieren ayudar: estos son nuestros espíritus afines y compañeros de viaje que son los primeros en ofrecerse como voluntarios para comenzar el viaje DevOps. En lo ideal, también son personas respetadas y con un alto grado de influencia sobre el resto de la organización, dando a nuestra iniciativa más credibilidad.

2. **Construir una masa crítica y una mayoría silenciosa:** en la siguiente fase, buscamos expandir las prácticas de DevOps a más equipos y flujos de valor con el objetivo de creando una base estable de apoyo. Al trabajar con equipos que son receptivos a nuestras ideas, incluso si no son los grupos más visibles o influyentes, ampliamos nuestra coalición que está generando más éxitos, creando un "carro" efecto "que aumenta aún más nuestra influencia. Específicamente evitamos el peligro batallas políticas que podrían poner en peligro nuestra iniciativa.

3. **Identificar los holdouts:** los "holdouts" son de alto perfil, influyentes detractores que tienen más probabilidades de resistir (y tal vez incluso sabotear) nuestros esfuerzos. En general, abordamos este grupo solo después de haber logrado una mayoría silenciosa, cuando hemos establecido suficientes éxitos para proteger con éxito nuestra iniciativa.

Expandir DevOps en una organización no es una tarea pequeña. Puede crear riesgos para individuos, departamentos y la organización en su conjunto. Pero como Ron van Kemenade, CIO de ING, quien ayudó a transformar la organización en uno de los La mayoría de las organizaciones tecnológicas admiradas dijo: "El cambio principal requiere coraje, especialmente en entornos corporativos donde la gente tiene miedo y pelea contigo. Pero si comienzas pequeño, realmente no tienes nada que temer. Cualquier líder necesita ser valiente lo suficiente como para asignar equipos para hacer una toma de riesgos calculada ".

CONCLUSIÓN

Peter Drucker, líder en el desarrollo de la educación gerencial, observó que "los peces pequeños aprenden a ser peces grandes en pequeños estanques". Al elegir cuidadosamente dónde y

cómo comenzar, podemos experimentar y aprender en áreas de nuestra organización que crear valor sin poner en peligro al resto de la organización. Al hacer esto, nosotros construya nuestra base de soporte, gane el derecho de expandir el uso de DevOps en nuestro

organización, y ganar el reconocimiento y la gratitud de un cada vez más grande distrito electoral.

† Estas organizaciones a veces se conocían como los "Asesinos B que están muriendo".

‡ La práctica de confiar en una fase de estabilización o en una fase de endurecimiento al final de un proyecto a menudo tiene resultados muy pobres, porque significa Los problemas no se encuentran ni se solucionan como parte del trabajo diario y no se abordan, lo que puede convertirse en problemas más grandes.

§ Que los servicios que tienen el mayor beneficio comercial potencial son sistemas brownfield no debería sorprender. Después de todo, estos son los sistemas en los que se confía más y tienen el mayor número de clientes existentes o la mayor cantidad de ingresos dependiendo de ellos.

¶ Big Bang, son posibles las transformaciones de arriba hacia abajo, como la transformación Agile en PayPal en 2012 que fue dirigida por su vicepresidente de tecnología, Kirsten Wolberg. Sin embargo, como con cualquier transformación sostenible y exitosa, esto requirió el más alto nivel de apoyo administrativo y un enfoque incesante y sostenido para impulsar los resultados necesarios.

6 Trabaja en nuestro valor Stream, haciéndolo Visible y Entendiendo el Expandiéndolo a través de la organización

Una vez que hemos identificado un flujo de valor al que queremos aplicar los principios de DevOps y patrones, nuestro siguiente paso es obtener una comprensión suficiente de cómo se entrega el valor a el cliente: qué trabajo se realiza y por quién, y qué pasos podemos tomar para mejorar el flujo

En el capítulo anterior, aprendimos sobre la transformación DevOps dirigida por Courtney Kissler y el equipo de Nordstrom. Con los años, han aprendido que uno de los La forma más eficiente de comenzar a mejorar cualquier flujo de valor es realizar un taller con todos las principales partes interesadas y realizan un ejercicio de mapeo de flujo de valor: un proceso (descrito más adelante en este capítulo) diseñado para ayudar a capturar todos los pasos necesarios para crear valor.

El ejemplo favorito de Kissler de las ideas valiosas e inesperadas que pueden provenir de el mapeo de flujo de valor es cuando intentaron mejorar los largos tiempos de espera asociados con solicita pasar por la aplicación Cosmetics Business Office, un mainframe COBOL aplicación que respaldaba a todos los gerentes de piso y departamento de su tienda departamentos de belleza y cosmética.

Esta aplicación permitió a los gerentes de departamento registrar nuevos vendedores para varios

líneas de productos transportadas en sus tiendas, para que puedan rastrear las comisiones de ventas, habilitar reembolsos de proveedores, y así sucesivamente.

Kissler explicó:

> Conocía bien esta aplicación particular de mainframe; anteriormente en mi carrera, apoyé este equipo de tecnología, así que sé de primera mano que durante casi una década, durante cada ciclo de planificación anual, debatiríamos sobre cómo necesitábamos obtener esta aplicación fuera de la unidad central. Por supuesto, como en la mayoría de las organizaciones, incluso cuando estaba lleno soporte de gestión, nunca parecíamos llegar a migrarlo.
>
> Mi equipo quería realizar un ejercicio de mapeo de flujo de valor para determinar si la aplicación COBOL realmente fue el problema, o tal vez hubo un problema mayor que necesitábamos abordar Realizaron un taller que reunió a todos con cualquier responsabilidad por entregar valor a nuestros clientes internos, incluido nuestro socios comerciales, el equipo de mainframe, los equipos de servicios compartidos, etc.

Página 104

> Lo que descubrieron fue que cuando los gerentes de departamento presentaban formulario de solicitud de 'asignación de línea de producto', les pedíamos un empleado número, que no tenían, por lo que lo dejarían en blanco o lo pondrían algo como 'No lo sé' Peor aún, para completar el formulario, departamento los gerentes tendrían que abandonar la tienda de manera inconveniente para usar una PC El back office. El resultado final fue todo este tiempo perdido, con el trabajo recuperado y adelante en el proceso.

Durante el taller, los participantes realizaron varios experimentos, incluidos eliminar el campo de número de empleado en el formulario y dejar que otro departamento obtenga eso información en un paso descendente. Estos experimentos, realizados con la ayuda de Los gerentes de departamento mostraron una reducción de cuatro días en el tiempo de procesamiento. El equipo después reemplazó la aplicación de PC con una aplicación de iPad, lo que permitió a los gerentes enviar la información necesaria sin salir de la tienda, y el tiempo de procesamiento fue reducido aún más a segundos.

Ella dijo con orgullo: "Con esas mejoras increíbles, todas las demandas para conseguir esto aplicación fuera de la unidad central desapareció. Además, otros líderes empresariales tomaron cuenta y comenzó a llegar a nosotros con una lista completa de más experimentos que querían Conducir con nosotros en sus propias organizaciones. Todos en el negocio y la tecnología. los equipos estaban entusiasmados con el resultado porque resolvieron un problema comercial real y, lo más importante, aprendieron algo en el proceso ".

En el resto de este capítulo, veremos los siguientes pasos: identificar todos los equipos requeridos para crear valor para el cliente, creando un mapa de flujo de valor para hacer visible

todo el trabajo requerido y su uso para guiar a los equipos sobre cómo mejorar y más rápidamente crear valor. Al hacer esto, podemos replicar los sorprendentes resultados descritos en este Ejemplo de Nordstrom.

IDENTIFICANDO LOS EQUIPOS QUE APOYAN NUESTRO VALOR CORRIENTE

Como demuestra este ejemplo de Nordstrom, en flujos de valor de cualquier complejidad, nadie la persona conoce todo el trabajo que debe realizarse para crear valor para el cliente, especialmente porque el trabajo requerido debe ser realizado por muchas personas diferentes equipos, a menudo muy alejados entre sí en los organigramas, geográficamente o por incentivos.

Como resultado, después de seleccionar una aplicación o servicio candidato para nuestra iniciativa DevOps, debemos identificar a todos los miembros de la cadena de valor responsables de trabajar juntos para crear valor para los clientes que se atienden. En general, esto incluye:

Propietario del producto: la voz interna de la empresa que define el siguiente conjunto de funcionalidad en el servicio

Desarrollo: el equipo responsable de desarrollar la funcionalidad de la aplicación en el Servicio

QA: el equipo responsable de garantizar que existan bucles de retroalimentación para garantizar el servicio funciona como se desee

Operaciones: el equipo a menudo responsable de mantener la producción. entorno y ayudar a garantizar que se cumplan los niveles de servicio requeridos

Infosec: el equipo responsable de asegurar sistemas y datos

Release managers: las personas responsables de gestionar y coordinar el implementación de producción y procesos de lanzamiento

Ejecutivos de tecnología o gerente de flujo de valor: en la literatura Lean, alguien quién es responsable de "garantizar que el flujo de valor cumpla o supere al cliente requisitos [y organizacionales] para el flujo de valor general, de principio a fin "

CREAR UN MAPA DE STREAM DE VALOR PARA VER EL TRABAJO

Después de identificar nuestros miembros de flujo de valor, nuestro siguiente paso es obtener un

comprensión de cómo se realiza el trabajo, documentado en forma de flujo de valor
mapa. En nuestro flujo de valor, el trabajo probablemente comienza con el propietario del producto, en forma de
solicitud del cliente o la formulación de una hipótesis comercial. Algún tiempo después, este trabajo
es aceptado por Desarrollo, donde las características se implementan en código y se registran para
nuestro repositorio de control de versiones. Las compilaciones se integran, se prueban en una producción
entorno, y finalmente desplegados en producción, donde (idealmente) crean valor para
nuestro cliente.

En muchas organizaciones tradicionales, este flujo de valor consistirá en cientos, si no
miles, de pasos, que requieren trabajo de cientos de personas. Porque documentar cualquier
El mapa de flujo de valor de este complejo probablemente requiera varios días, podemos realizar una
taller de un día, donde ensamblamos todos los componentes clave y los retiramos del
distracciones de su trabajo diario.

Nuestro objetivo no es documentar cada paso y las minucias asociadas, sino lo suficiente
Comprender las áreas de nuestro flujo de valor que ponen en peligro nuestros objetivos de flujo rápido,
cortos plazos de entrega y resultados confiables para el cliente. Idealmente, hemos reunido esos
personas con autoridad para cambiar su parte de la corriente de valor.†

Damon Edwards, coanfitrión del podcast *DevOps Café* , observó: “En mi experiencia, estos
Los tipos de ejercicios de mapeo de flujo de valor son siempre reveladores. A menudo es el primero
momento en que la gente ve cuánto trabajo y heroicidad se requieren para entregar valor al
cliente. Para Operaciones, puede ser la primera vez que ven las consecuencias que
se produce cuando los desarrolladores no tienen acceso a entornos configurados correctamente, que
contribuye a un trabajo aún más loco durante las implementaciones de código. Para el desarrollo, puede
ser la primera vez que ven todos los heroicos requeridos por Test and Operations para
para implementar su código en producción, mucho después de que marquen una característica como 'completada' ".

Utilizando toda la amplitud de conocimiento aportada por los equipos involucrados en la cadena de valor,
debemos centrar nuestra investigación y escrutinio en las siguientes áreas:

Page 106

Lugares donde el trabajo debe esperar semanas o incluso meses, como ser productivo
entornos, procesos de aprobación de cambios o procesos de revisión de seguridad

Lugares donde se genera o recibe un trabajo significativo

Nuestro primer paso para documentar nuestro flujo de valor solo debe consistir en un proceso de alto nivel
bloques Por lo general, incluso para flujos de valor complejos, los grupos pueden crear un diagrama con cinco
a quince bloques de proceso en pocas horas. Cada bloque de proceso debe incluir el plomo
tiempo y tiempo de proceso para procesar un elemento de trabajo, así como el% C / A medido
por los consumidores intermedios de la producción. ‡

Figura 10: *Un ejemplo de un mapa de flujo de valor*
(Fuente: Humble, Molesky y O'Reilly, Lean Enterprise *, 139.)*

Utilizamos las métricas de nuestro mapa de flujo de valor para guiar nuestros esfuerzos de mejora. En el Ejemplo de Nordstrom, se centraron en las bajas tasas de% C / A en el formulario de solicitud enviado por gerentes de departamento debido a la ausencia de números de empleados. En otros casos, puede ser largos tiempos de entrega o bajas tasas de% C / A al entregar una prueba configurada correctamente entornos para equipos de desarrollo, o podrían ser los largos plazos de entrega necesarios para ejecutar y pasar pruebas de regresión antes de cada lanzamiento de software.

Una vez que identificamos la métrica que queremos mejorar, debemos realizar el siguiente nivel de observaciones y mediciones para comprender mejor el problema y luego construir un Mapa de flujo de valor futuro idealizado, que sirve como una condición objetivo para lograr por algunos fecha (por ejemplo, generalmente de tres a doce meses).

El liderazgo ayuda a definir este estado futuro y luego guía y permite al equipo lluvia de ideas hipótesis y contramedidas para lograr la mejora deseada a ese declarar, realizar experimentos para probar esas hipótesis e interpretar los resultados para determinar si las hipótesis fueron correctas. Los equipos siguen repitiendo e iterando, usando cualquier nuevo aprendizaje para informar los próximos experimentos.

CREANDO UN EQUIPO DE TRANSFORMACIÓN DEDICADO

Uno de los desafíos inherentes a iniciativas como las transformaciones de DevOps es que inevitablemente están en conflicto con las operaciones comerciales en curso. Parte de esto es natural resultado de cómo evolucionan las empresas exitosas. Una organización que ha tenido éxito. por cualquier período extendido de tiempo (años, décadas o incluso siglos) ha creado

mecanismos para perpetuar las prácticas que los hicieron exitosos, como el producto
desarrollo, administración de pedidos y operaciones de la cadena de suministro.

Se utilizan muchas técnicas para perpetuar y proteger el funcionamiento de los procesos actuales, como
como especialización, enfoque en eficiencia y repetibilidad, burocracias que imponen
procesos de aprobación y controles para proteger contra la variación. En particular, las burocracias.
son increíblemente resistentes y están diseñados para sobrevivir a condiciones adversas: se puede eliminar
la mitad de los burócratas, y el proceso aún sobrevivirá.

Si bien esto es bueno para preservar el status quo, a menudo necesitamos cambiar la forma en que trabajamos para
adaptarse a las condiciones cambiantes en el mercado. Hacer esto requiere interrupción y
innovación, lo que nos pone en desacuerdo con los grupos que actualmente son responsables diariamente
operaciones y las burocracias internas, y quién casi siempre ganará.

En su libro *The Other Side of Innovation: Solving the Execution Challenge,* Dr. Vijay
Govindarajan y el Dr. Chris Trimble, ambos miembros de la facultad de Tuck de Dartmouth College
School of Business, describió sus estudios sobre cómo se logra la innovación disruptiva
a pesar de estas poderosas fuerzas de las operaciones diarias. Documentaron cómo el cliente
los productos de seguros de automóviles impulsados se desarrollaron y comercializaron con éxito en Allstate,
cómo se creó el rentable negocio de publicación digital en el *Wall Street Journal* ,
el desarrollo de la innovadora zapatilla de trail running en Timberland, y el
desarrollo del primer coche eléctrico en BMW.

Con base en su investigación, el Dr. Govindarajan y el Dr. Trimble afirman que las organizaciones
necesita crear un equipo de transformación dedicado que pueda operar fuera del resto
de la organización responsable de las operaciones diarias (que llaman
"Equipo dedicado" y "motor de rendimiento" respectivamente).

En primer lugar, responsabilizaremos a este equipo dedicado por lograr un claro
resultado definido, medible, a nivel del sistema (por ejemplo, reducir el tiempo de implementación desde
"Código comprometido en el control de versiones para ejecutarse con éxito en producción" en un 50%). En
Para ejecutar dicha iniciativa, hacemos lo siguiente:

- Asigne miembros del equipo dedicado para que se asignen únicamente a DevOps
  esfuerzos de transformación (en lugar de "mantener todas sus responsabilidades actuales, pero
  dedica el 20% de tu tiempo a esta nueva cosa de DevOps ").

- Seleccione miembros del equipo que sean generalistas, que tengan habilidades en una amplia variedad de
  dominios

- Seleccionar miembros del equipo que tengan relaciones duraderas y de respeto mutuo.
  con el resto de la organización.

- Cree un espacio físico separado para el equipo dedicado, si es posible, para maximizar
  flujo de comunicación dentro del equipo, y creando cierto aislamiento del resto del
  organización.

Si es posible, liberaremos al equipo de transformación de muchas de las reglas y políticas que restringir al resto de la organización, como lo hizo National Instruments, descrito en el capítulo previo. Después de todo, los procesos establecidos son una forma de memoria institucional: nosotros necesita el equipo dedicado para crear los nuevos procesos y aprendizajes necesarios para generar nuestros resultados deseados, creando nueva memoria institucional.

Crear un equipo dedicado no solo es bueno para el equipo, sino también bueno para el motor de rendimiento. Al crear un equipo separado, creamos el espacio para que ellos puedan experimentar con nuevas prácticas, protegiendo al resto de la organización del potencial interrupciones y distracciones asociadas con él.

ACUERDAN UN OBJETIVO COMPARTIDO

Una de las partes más importantes de cualquier iniciativa de mejora es definir una medida mensurable objetivo con un plazo claramente definido, entre seis meses y dos años en el futuro. Eso debería requerir un esfuerzo considerable pero aún ser alcanzable. Y el logro de la meta debería crear un valor obvio para la organización en general y para nuestros clientes.

Estos objetivos y el plazo deben ser acordados por los ejecutivos y conocidos por todos en la organización. También queremos limitar el número de estos tipos de iniciativas que se llevan a cabo simultáneamente para evitar que gravemos demasiado la organización cambiar la capacidad de gestión de los líderes y la organización. Ejemplos de mejora los objetivos pueden incluir:

- Reduzca el porcentaje del presupuesto gastado en soporte de productos y trabajo no planificado en un 50%.
- Asegúrese de que el tiempo de espera desde el registro del código hasta el lanzamiento de producción sea de una semana o menos para el 95% de cambios
- Asegúrese de que los lanzamientos siempre se puedan realizar durante el horario comercial normal con cero falta del tiempo.
- Integre todos los controles de seguridad de la información necesarios en la tubería de implementación para pasar todos los requisitos de cumplimiento requeridos.

Una vez que se aclara el objetivo de alto nivel, los equipos deben decidir una cadencia regular para conducir El trabajo de mejora. Al igual que el trabajo de desarrollo de productos, queremos trabajo de transformación para hacerse de manera iterativa e incremental. Una iteración típica estará en el rango de De dos a cuatro semanas. Para cada iteración, los equipos deben acordar un pequeño conjunto de objetivos que genera valor y progresa hacia la meta a largo plazo. Al final de cada iteración, los equipos deben revisar su progreso y establecer nuevas metas para la próxima iteración.

MANTENGA NUESTROS HORIZONTES DE PLANIFICACIÓN DE MEJORA CORTO

En cualquier proyecto de transformación de DevOps, debemos mantener nuestros horizontes de planificación cortos, solo como si estuviéramos en una startup haciendo desarrollo de productos o clientes. Nuestra iniciativa debe esforzarse por generar mejoras medibles o datos procesables en semanas (o, en el peor caso, meses).

Al mantener cortos nuestros horizontes de planificación e intervalos de iteración, logramos lo siguiente:

Flexibilidad y la capacidad de priorizar y volver a planificar rápidamente

Disminuya el retraso entre el trabajo gastado y la mejora realizada, que
fortalece nuestro ciclo de retroalimentación, lo que aumenta la probabilidad de reforzar los comportamientos deseados:
cuando las iniciativas de mejora son exitosas, fomenta más inversión

Aprendizaje más rápido generado desde la primera iteración, lo que significa una integración más rápida de nuestro
aprendizajes en la próxima iteración

Reducción de la energía de activación para obtener mejoras.

Realización más rápida de mejoras que marcan diferencias significativas en nuestro diario
trabajo

Menos riesgo de que nuestro proyecto sea eliminado antes de que podamos generar resultados demostrables

RESERVE EL 20% DE LOS CICLOS PARA REQUISITOS NO FUNCIONALES Y REDUCIR LA DEUDA TÉCNICA

Un problema común a cualquier esfuerzo de mejora de procesos es cómo priorizarlo adecuadamente:
después de todo, las organizaciones que más lo necesitan son aquellas que tienen la menor cantidad de tiempo para
gastar en mejoras. Esto es especialmente cierto en las organizaciones tecnológicas debido a
deuda técnica.

Las organizaciones que luchan con la deuda financiera solo hacen pagos de intereses y nunca
reducir el capital del préstamo, y eventualmente pueden encontrarse en situaciones donde
ya no puede atender los pagos de intereses. Del mismo modo, las organizaciones que no pagan
la deuda técnica puede verse tan cargada de soluciones diarias para los problemas restantes
sin reparar que ya no pueden completar ningún trabajo nuevo. En otras palabras, ahora son
solo haciendo el pago de intereses sobre su deuda técnica.

Gestionaremos activamente esta deuda técnica asegurándonos de invertir al menos el 20% de todos
Ciclos de desarrollo y operaciones de refactorización, inversión en trabajos de automatización y
arquitectura y requisitos no funcionales (NFR, a veces referidos como
"Ilities"), como mantenibilidad, manejabilidad, escalabilidad, confiabilidad, comprobabilidad,
implementabilidad y seguridad.

Figura 11: *Invierta el 20% de los ciclos en aquellos que crean un valor positivo, invisible para el usuario (Fuente: "Aprendizaje automático y deuda técnica con D. Sculley",* Software Engineering Daily *podcast, 17 de noviembre de 2015, http://softwareengineeringdaily.com/2015/11/17/machine-aprendizaje-y-técnico-deuda-con-d-sculley / .)*

Después de la experiencia cercana a la muerte de eBay a fines de la década de 1990, Marty Cagan, autor de *Inspirado: Cómo crear productos que los clientes adoren,* el libro seminal sobre diseño de productos y gestión, codificó la siguiente lección:

> El trato [entre los propietarios del producto y] la ingeniería es así: Producto
> la administración toma el 20% de la capacidad del equipo desde el principio y le da esto a
> ingeniería para gastar como mejor les parezca. Pueden usarlo para reescribir, rediseñar o volver a diseñar
> factorizar partes problemáticas de la base del código ... lo que crean que es necesario evitar
> alguna vez tener que venir al equipo y decir: 'tenemos que parar y reescribir [todo nuestro código]'.
> Si hoy estás en muy mal estado, es posible que necesites hacer este 30% o incluso más de
> Los recursos. Sin embargo, me pongo nervioso cuando encuentro equipos que piensan que pueden escapar
> con mucho menos del 20%.

Cagan señala que cuando las organizaciones no pagan su "impuesto del 20%", la deuda técnica lo hará aumentar hasta el punto en que una organización gasta inevitablemente todos sus ciclos pagando abajo de la deuda técnica. En algún momento, los servicios se vuelven tan frágiles que la entrega de funciones se detiene porque todos los ingenieros están trabajando en problemas de confiabilidad o trabajando alrededor de los problemas

Al dedicar el 20% de nuestros ciclos para que Dev y Ops puedan crear contramedidas duraderas A los problemas que encontramos en nuestro trabajo diario, nos aseguramos de que la deuda técnica no impedir nuestra capacidad de desarrollar y operar nuestros servicios en producción de manera rápida y segura.

Elevar la presión adicional de la deuda técnica de los trabajadores también puede reducir los niveles de agotamiento

Operation InVersion de LinkedIn presenta un interesante caso de estudio que ilustra
La necesidad de pagar la deuda técnica como parte del trabajo diario. Seis meses después de su
IPO exitosa en 2011, LinkedIn continuó luchando con problemas
implementaciones que se volvieron tan dolorosas que lanzaron Operation InVersion,
donde detuvieron todo el desarrollo de características durante dos meses para revisar
sus entornos informáticos, implementaciones y arquitectura.

Página 111

LinkedIn fue creado en 2003 para ayudar a los usuarios a "conectarse a su red para un mejor trabajo
oportunidades ". Al final de su primera semana de operaciones, tenían 2.700 miembros.
Un año después, tenían más de un millón de miembros y crecieron exponencialmente.
desde entonces. Para noviembre de 2015, LinkedIn tenía más de 350 millones de miembros, que
generar decenas de miles de solicitudes por segundo, lo que resulta en millones de consultas
por segundo en los sistemas de back-end de LinkedIn.

Desde el principio, LinkedIn se ejecutó principalmente en su aplicación Leo local, un
Aplicación Java monolítica que sirvió a cada página a través de servlets y gestionó
Conexiones JDBC a varias bases de datos Oracle de back-end. Sin embargo, para mantenerse al día
Con el creciente tráfico en sus primeros años, dos servicios críticos fueron desconectados de
Leo: las primeras consultas manejadas alrededor del gráfico de conexión de miembros enteramente
memoria, y el segundo fue la búsqueda de miembros, que superó al primero.

Para 2010, la mayoría de los nuevos desarrollos ocurrían en nuevos servicios, con casi uno
Cien servicios que se ejecutan fuera de Leo. El problema era que Leo solo era
desplegarse una vez cada dos semanas.

Josh Clemm, gerente senior de ingeniería en LinkedIn, explicó que para 2010,
La empresa tenía problemas importantes con Leo. A pesar de escalar verticalmente
Leo al agregar memoria y CPU, "Leo a menudo bajaba en producción, era
difícil de solucionar y recuperar, y difícil de liberar nuevo código ... Estaba claro
necesitábamos 'matar a Leo' y dividirlo en muchos pequeños funcionales y apátridas
servicios."

En 2013, la periodista Ashlee Vance de Bloomberg describió cómo "cuando LinkedIn
intentaría agregar un montón de cosas nuevas a la vez, el sitio se desmoronaría
desorden roto, que requiere que los ingenieros trabajen hasta bien entrada la noche y solucionen los problemas ".
Para el otoño de 2011, las noches ya no eran un rito de iniciación o una actividad de unión,
porque los problemas se habían vuelto intolerables. Algunos de los mejores ingenieros de LinkedIn,

incluido Kevin Scott, que se había unido como vicepresidente de ingeniería de LinkedIn tres meses antes de su oferta pública inicial, decidió dejar por completo de ingeniería trabajar en nuevas funciones y dedicar todo el departamento a arreglar el núcleo del sitio infraestructura. Llamaron al esfuerzo Operación Inversión.

Scott lanzó Operation InVersion como una forma de "inyectar los inicios de una cultura manifiesto en la cultura de ingeniería de su equipo. No habría una nueva característica desarrollo hasta que se modernizó la arquitectura informática de LinkedIn, eso es lo que los negocios *y* su equipo lo necesitaban ".

Scott describió una desventaja: "Haz público, tienes todo el mundo mirándote, y luego le decimos a la gerencia que no vamos a entregar nada nuevo mientras todos de trabajos de ingeniería en este proyecto [InVersion] para los próximos dos meses. Era un cosa aterradora ".

Sin embargo, Vance describió los resultados masivamente positivos de Operation InVersion. "LinkedIn creó un conjunto completo de software y herramientas para ayudarlo a desarrollar código para el sitio. En lugar de esperar semanas para que sus nuevas funciones lleguen a El sitio principal de LinkedIn, los ingenieros podrían desarrollar un nuevo servicio, tener una serie de los sistemas automatizados examinan el código en busca de errores y problemas que el servicio pueda

interactuar con las funciones existentes y lanzarlo directamente al sitio de LinkedIn en vivo ... El cuerpo de ingeniería de LinkedIn [ahora] realiza importantes actualizaciones al sitio tres veces un día." Al crear un sistema de trabajo más seguro, el valor que crearon incluyó menos sesiones nocturnas de hacinamiento, con más tiempo para desarrollar características nuevas e innovadoras.

Como Josh Clemm describió en su artículo sobre el escalado en LinkedIn, "el escalado puede ser medido en muchas dimensiones, incluida la organización ... [Operación InVersion] permitió a toda la organización de ingeniería centrarse en mejorar herramientas e implementación, infraestructura y productividad del desarrollador. Era exitosos en permitir la agilidad de ingeniería, necesitamos construir el nuevo escalable productos que tenemos hoy ... [En] 2010, ya teníamos más de 150 servicios separados. Hoy tenemos más de 750 servicios ".

Kevin Scott declaró: “Tu trabajo como ingeniero y tu propósito como tecnología El equipo es ayudar a su empresa a ganar. Si lideras un equipo de ingenieros, es mejor tomar la perspectiva de un CEO. Su trabajo es descubrir qué es lo que su empresa, necesita su negocio, su mercado, su entorno competitivo. Aplicar eso a su equipo de ingeniería para que su empresa gane ".

Al permitir que LinkedIn pague casi una década de deuda técnica, Project InVersion permitió estabilidad y seguridad, al tiempo que estableció la siguiente etapa de crecimiento para el empresa. Sin embargo, requirió dos meses de enfoque total en no funcional requisitos, a expensas de todas las funciones prometidas hechas al público

mercados durante una salida a bolsa. Al encontrar y solucionar problemas como parte de nuestro trabajo diario, nosotros administrar nuestra deuda técnica para evitar estas experiencias de "casi muerte".

AUMENTAR LA VISIBILIDAD DEL TRABAJO

Para poder saber si estamos progresando hacia nuestra meta, es esencial que Todos en la organización conocen el estado actual del trabajo. Hay muchas maneras de hacer visible el estado actual, pero lo más importante es que la información que la pantalla está actualizada y revisamos constantemente lo que medimos para asegurarnos de que ayudándonos a comprender el progreso hacia nuestras condiciones objetivo actuales.

La siguiente sección discute patrones que pueden ayudar a crear visibilidad y alineación entre equipos y funciones.

UTILICE HERRAMIENTAS PARA REFORZAR EL COMPORTAMIENTO DESEADO

Como Christopher Little, un ejecutivo de software y uno de los primeros cronistas de DevOps, observó: “Los antropólogos describen las herramientas como un artefacto cultural. Cualquier discusión sobre cultura después de la invención del fuego también debe tratarse de herramientas ". Del mismo modo, en el valor DevOps Stream, utilizamos herramientas para reforzar nuestra cultura y acelerar los cambios de comportamiento deseados.

Un objetivo es que nuestras herramientas refuercen que Desarrollo y Operaciones no solo tienen objetivos compartidos, pero tienen un trabajo atrasado común, idealmente almacenado en un trabajo común sistema y usando un vocabulario compartido, para que el trabajo pueda ser priorizado globalmente.

Al hacer esto, Desarrollo y Operaciones pueden terminar creando una cola de trabajo compartida, en lugar de que cada silo use uno diferente (por ejemplo, Desarrollo usa JIRA mientras que Operaciones

usa ServiceNow). Un beneficio significativo de esto es que cuando los incidentes de producción son mostrado en los mismos sistemas de trabajo que el trabajo de desarrollo, será obvio cuando esté en curso los incidentes deberían detener otro trabajo, especialmente cuando tenemos una junta kanban.

Otro beneficio de tener Desarrollo y Operaciones usando una herramienta compartida es un trabajo atrasado, donde todos priorizan proyectos de mejora desde una perspectiva global, Seleccionar el trabajo que tiene el mayor valor para la organización o que más reduce deuda. A medida que identificamos la deuda técnica, la agregamos a nuestra reserva prioritaria si no podemos abordarlo de inmediato. Para problemas que no se han resuelto, podemos usar nuestro "20% de tiempo para requisitos no funcionales "para corregir los elementos principales de nuestra cartera de pedidos.

Otras tecnologías que refuerzan los objetivos compartidos son las salas de chat, como los canales IRC, HipChat, Campfire, Slack, Flowdock y OpenFire. Las salas de chat permiten compartir rápidamente información (en lugar de completar formularios que se procesan a través de predefinidos flujos de trabajo), la capacidad de invitar a otras personas según sea necesario y registros de historial que son

se registra automáticamente para la posteridad y se puede analizar durante las sesiones post-mortem. Se crea una dinámica sorprendente cuando tenemos un mecanismo que permite a cualquier equipo miembro para ayudar rápidamente a otros miembros del equipo, o incluso a personas ajenas a su equipo: El tiempo requerido para obtener información o el trabajo necesario puede ir de días a minutos. En Además, debido a que todo se está grabando, es posible que no necesitemos pedirle a otra persona ayuda en el futuro, simplemente la buscamos.

Sin embargo, el entorno de comunicación rápida facilitado por las salas de chat también puede ser un retirarse. Como Ryan Martens, el fundador y CTO de Rally Software, observa: "En un chat habitación, si alguien no recibe una respuesta en un par de minutos, es totalmente aceptado y esperaba que puedas volver a molestarlos hasta que obtengan lo que necesitan ".

Las expectativas de respuesta inmediata pueden, por supuesto, conducir a resultados no deseados. UN El aluvión constante de interrupciones y preguntas puede evitar que las personas trabajo necesario hecho. Como resultado, los equipos pueden decidir que ciertos tipos de solicitudes deberían pasar por herramientas más estructuradas y asincrónicas.

CONCLUSIÓN

En este capítulo, identificamos a todos los equipos que respaldan nuestro flujo de valor y los capturamos en un mapa de flujo de valor qué trabajo se requiere para entregar valor al cliente. los El mapa de flujo de valor proporciona la base para comprender nuestro estado actual, incluido nuestro tiempo de entrega y métricas de% C / A para áreas problemáticas, e informa cómo establecemos un futuro estado.

Esto permite a los equipos de transformación dedicados iterar y experimentar rápidamente para mejorar el rendimiento. También nos aseguramos de asignar una cantidad de tiempo suficiente para mejora, solucionando problemas conocidos y problemas arquitectónicos, incluyendo nuestros requerimientos funcionales. Los estudios de caso de Nordstrom y LinkedIn demuestran cómo se pueden hacer mejoras dramáticas en los plazos y la calidad cuando encontramos problemas en nuestro flujo de valor y pagar deuda técnica.

† Lo que hace que sea aún más importante que limitemos el nivel de detalle que se recopila: el tiempo de todos es valioso y escaso.

• Por el contrario, hay muchos ejemplos de uso de herramientas de una manera que garantiza que no ocurran cambios de comportamiento. Por ejemplo, una organización se compromete a un herramienta de planificación ágil, pero luego la configura para un proceso en cascada, que simplemente mantiene el status quo.

116 de 1189.

7 Organización y Arquitectura con Ley de Conway en Mente: Cómo diseñar nuestro

En los capítulos anteriores, identificamos un flujo de valor para comenzar nuestra transformación DevOps y estableció objetivos y prácticas compartidas para permitir que un equipo de transformación dedicado a Mejorar la forma en que entregamos valor al cliente.

En este capítulo, comenzaremos a pensar en cómo organizarnos para lograr mejor nuestro objetivos de flujo de valor. Después de todo, cómo organizamos nuestros equipos afecta cómo realizamos nuestro trabajo. El Dr. Melvin Conway realizó un famoso experimento en 1968 con una investigación por contrato. organización que tenía ocho personas encargadas de producir un COBOL y un Compilador ALGOL. Observó: "Después de algunas estimaciones iniciales de dificultad y tiempo, cinco las personas fueron asignadas al trabajo de COBOL y tres al trabajo de ALGOL. El COBOL resultante el compilador se ejecutó en cinco fases, el compilador ALGOL se ejecutó en tres ".

Estas observaciones condujeron a lo que ahora se conoce como la Ley de Conway, que establece que "Las organizaciones que diseñan sistemas ... están obligadas a producir diseños que son copias de las estructuras de comunicación de estas organizaciones ... Cuanto más grande es una organización, más menos flexibilidad tiene y más pronunciado es el fenómeno ". Eric S. Raymond, autor del libro *La catedral y el bazar: reflexiones sobre Linux y código abierto por un Accidental Revolutionary* , creó una versión simplificada (y ahora más famosa) de Ley de Conway en su archivo de jerga: "La organización del software y la organización de el equipo de software será congruente; comúnmente indicado como 'si tienes cuatro grupos trabajando en un compilador, obtendrás un compilador de 4 pasadas ".

En otras palabras, cómo organizamos nuestros equipos tiene un poderoso efecto en el software que producir, así como nuestros resultados arquitectónicos y de producción resultantes. Para obtener Flujo rápido de trabajo desde el desarrollo hasta las operaciones, con alta calidad y excelente cliente resultados, debemos organizar nuestros equipos y nuestro trabajo para que la Ley de Conway funcione para nuestros ventaja. Si se hace mal, la Ley de Conway evitará que los equipos trabajen de manera segura y independientemente; en cambio, estarán estrechamente unidos, todos esperándose el uno al otro trabajo por hacer, incluso con pequeños cambios que crean catástrofes potencialmente globales Consecuencias.

Un ejemplo de cómo la Ley de Conway puede impedir o reforzar nuestros objetivos se puede ver en un tecnología desarrollada en Etsy llamada Sprouter. El viaje de DevOps de Etsy comenzó en 2009, y es una de las organizaciones DevOps más admiradas, con ingresos en 2014 de casi $ 200 millones y una OPV exitosa en 2015.

Desarrollado originalmente en 2007, Sprouter conectó personas, procesos y tecnología en formas que crearon muchos resultados no deseados. Sprouter, abreviatura de "procedimiento almacenado

enrutador ", fue diseñado originalmente para ayudar a facilitar la vida de los desarrolladores y la base de datos equipos Como dijo Ross Snyder, ingeniero senior en Etsy, durante su presentación en Surge 2011, "Sprouter fue diseñado para permitir que los equipos de desarrollo escriban código PHP en la aplicación, los DBA para escribir SQL dentro de Postgres, con Sprouter ayudándolos a encontrarse en el medio ".

Sprouter residía entre su aplicación PHP front-end y la base de datos Postgres, centralizando el acceso a la base de datos y ocultando la implementación de la base de datos capa de aplicación. El problema era que agregar cualquier cambio en la lógica de negocios resultó en fricción significativa entre los desarrolladores y los equipos de bases de datos. Como observó Snyder, "para casi cualquier nueva funcionalidad del sitio, Sprouter requería que los DBA escribieran un nuevo archivo almacenado procedimiento. Como resultado, cada vez que los desarrolladores quisieran agregar una nueva funcionalidad, lo harían necesitan algo de los DBA, que a menudo les obligaba a atravesar un montón de burocracia." En otras palabras, los desarrolladores que crean nuevas funcionalidades dependen de el equipo de DBA, que necesitaba ser priorizado, comunicado y coordinado, resultando en el trabajo sentado en colas, reuniones, plazos de entrega más largos, etc. Esto es porque Sprouter creó un fuerte acoplamiento entre los equipos de desarrollo y de bases de datos, evitar que los desarrolladores puedan desarrollar, probar e implementar de forma independiente sus código en producción.

Además, los procedimientos almacenados de la base de datos estaban estrechamente acoplados a Sprouter, en cualquier momento se cambió el procedimiento, también requirió cambios en Sprouter. El resultado fue que Sprouter se convirtió en un único punto de falla cada vez más grande. Snyder explicó que todo era tan estrechamente acoplado y requirió un nivel tan alto de sincronización como resultado, que casi cada despliegue causó una mini interrupción.

Se pueden explicar tanto los problemas asociados con Sprouter como su eventual solución. por la ley de Conway. Etsy inicialmente tenía dos equipos, los desarrolladores y los DBA, que eran cada uno responsable de dos capas del servicio, la capa lógica de la aplicación y almacenada capa de procedimiento Dos equipos trabajando en dos capas, como predice la Ley de Conway. Sprouter fue pretendía facilitar la vida de ambos equipos, pero no funcionó como se esperaba, cuando los negocios las reglas cambiaron, en lugar de cambiar solo dos capas, ahora necesitaban hacer cambios en tres capas (en la aplicación, en los procedimientos almacenados y ahora en Sprouter). los Los desafíos resultantes de coordinar y priorizar el trabajo en tres equipos significativamente aumentaron los tiempos de entrega y causaron problemas de confiabilidad.

En la primavera de 2009, como parte de lo que Snyder llamó "el gran cultural de Etsy transformación ", se unió Chad Dickerson como su nuevo CTO. Dickerson se puso en movimiento muchas cosas, incluida una inversión masiva en la estabilidad del sitio, hacer que los desarrolladores realicen sus propios despliegues en producción, así como comenzar un viaje de dos años para Eliminar Sprouter.

Para hacer esto, el equipo decidió mover toda la lógica empresarial de la capa de base de datos al capa de aplicación, eliminando la necesidad de Sprouter. Crearon un pequeño equipo que escribió un Capa de mapeo relacional de objetos PHP (ORM),† permitiendo a los desarrolladores front-end hacer llama directamente a la base de datos y reduce la cantidad de equipos necesarios para cambiar lógica de negocios desde tres equipos hasta un equipo.

Como describió Snyder, “Comenzamos a usar el ORM para cualquier área nueva del sitio y

migró pequeñas partes de nuestro sitio desde Sprouter a ORM con el tiempo. Nos tomó dos años

para migrar todo el sitio fuera de Sprouter. Y a pesar de que todos nos quejamos de Sprouter
todo el tiempo, se mantuvo en producción en todo momento ".

Al eliminar Sprouter, también eliminaron los problemas asociados con múltiples equipos
necesidad de coordinar cambios de lógica de negocios, disminución del número de traspasos y
aumentó significativamente la velocidad y el éxito de las implementaciones de producción, mejorando el sitio
estabilidad. Además, debido a que los equipos pequeños podrían desarrollar e implementar independientemente
código sin requerir que otro equipo realice cambios en otras áreas del sistema,
La productividad del desarrollador aumentó.

Sprouter finalmente se eliminó de la producción y los repositorios de control de versiones de Etsy en
principios de 2001. Como dijo Snyder, "Wow, se sintió bien". [1]

Como Snyder y Etsy experimentaron, cómo diseñamos nuestra organización dicta cómo es el trabajo
realizado, y, por lo tanto, los resultados que logramos. Durante el resto de este capítulo, nosotros
explorará cómo la Ley de Conway puede afectar negativamente el rendimiento de nuestro flujo de valor,
y, lo que es más importante, cómo organizamos nuestros equipos para utilizar la Ley de Conway en nuestro beneficio.

ARQUETÍAS ORGANIZACIONALES

En el campo de las ciencias de decisión, existen tres tipos principales de estructuras organizativas.
que informan cómo diseñamos nuestros flujos de valor DevOps teniendo en cuenta la Ley de Conway:
funcional , *matriz* y *mercado* . Los define el Dr. Roberto Fernández de la siguiente manera:

> Las organizaciones orientadas a la funcionalidad optimizan la experiencia, la división del trabajo o la reducción
> costo. Estas organizaciones centralizan la experiencia, lo que ayuda a permitir el crecimiento profesional y
> desarrollo de habilidades, y a menudo tienen estructuras organizativas jerárquicas altas. Esto tiene
> ha sido el método predominante de organización para Operaciones (es decir, administradores de servidores, redes
> administradores, administradores de bases de datos, etc., están organizados en grupos separados).
>
> Las organizaciones orientadas a matrices intentan combinar la orientación funcional y de mercado.
> Sin embargo, como muchos de los que trabajan o administran organizaciones matriciales observan, la matriz
> Las organizaciones a menudo resultan en estructuras organizacionales complicadas, como
> contribuyentes que reportan a dos gerentes o más, y algunas veces no logran ninguno de
> Los objetivos de orientación funcional o de mercado.
>
> Las organizaciones orientadas al mercado optimizan para responder rápidamente a las necesidades del cliente.
> Estas organizaciones tienden a ser planas, compuestas de múltiples disciplinas multifuncionales
> (por ejemplo, marketing, ingeniería, etc.), que a menudo conducen a posibles redundancias en
> la organización. Así es como operan muchas organizaciones prominentes que adoptan DevOps

—En ejemplos extremos, como en Amazon o Netflix, cada equipo de servicio es simultáneamente responsable de la entrega de funciones y el soporte del servicio.§

Con estas tres categorías de organizaciones en mente, exploremos más a fondo cómo La orientación funcional, especialmente en las operaciones, puede causar resultados no deseados en el flujo de valor tecnológico, como predeciría la Ley de Conway.

PROBLEMAS A MENUDO CAUSADOS POR FUNCIONAMIENTO EXCESIVO ORIENTACIÓN ("OPTIMIZAR POR EL COSTO")

En las organizaciones tradicionales de operaciones de TI, a menudo utilizamos orientación funcional para organizar nuestros equipos por sus especialidades. Ponemos a los administradores de la base de datos en un grupo, el administradores de red en otro, los administradores del servidor en un tercero, y así sucesivamente. Uno Una de las consecuencias más visibles de esto son los largos plazos de entrega, especialmente para actividades complejas. como implementaciones grandes donde debemos abrir tickets con múltiples grupos y coordinar transferencias de trabajo, lo que resulta en nuestro trabajo esperando en largas colas en cada paso.

Para agravar el problema, la persona que realiza el trabajo a menudo tiene poca visibilidad o comprensión de cómo su trabajo se relaciona con cualquier objetivo de flujo de valor (por ejemplo, "Solo soy configurando servidores porque alguien me lo dijo "). Esto coloca a los trabajadores en una creatividad y vacío de motivación

El problema se exacerba cuando cada área funcional de Operaciones tiene que servir a múltiples flujos de valor (es decir, múltiples equipos de desarrollo) que compiten por sus escasos ciclos. En Para que los equipos de desarrollo realicen su trabajo de manera oportuna, a menudo tenemos que escalar problemas a un gerente o director, y eventualmente a alguien (generalmente un ejecutivo) quien finalmente puede priorizar el trabajo contra los objetivos organizacionales globales en lugar de objetivos funcionales del silo. Esta decisión debe luego caer en cascada en cada una de las funciones áreas para cambiar las prioridades locales, y esto, a su vez, ralentiza a otros equipos. Cuando cada equipo agiliza su trabajo, el resultado neto es que cada proyecto termina moviéndose al mismo lento arrastre

Además de las largas colas y los largos plazos de entrega, esta situación resulta en transferencias pobres, grandes cantidades de reelaboración, problemas de calidad, cuellos de botella y demoras. Este embotellamiento impide la logro de objetivos organizacionales importantes, que a menudo superan con creces el deseo de reducir costos. ¶

Del mismo modo, la orientación funcional también se puede encontrar con QA centralizado e Infosec funciones, que pueden haber funcionado bien (o al menos, lo suficientemente bien) al realizar menos lanzamientos frecuentes de software. Sin embargo, a medida que aumentamos el número de equipos de desarrollo y sus frecuencias de despliegue y lanzamiento, la mayoría de las organizaciones orientadas funcionalmente

tendrá dificultades para mantenerse y entregar resultados satisfactorios, especialmente cuando su
El trabajo se realiza manualmente. Ahora estudiaremos cómo las organizaciones orientadas al mercado
trabajo.

HABILITAR EQUIPOS ORIENTADOS AL MERCADO ("OPTIMIZAR PARA VELOCIDAD")

En términos generales, para lograr los resultados de DevOps, necesitamos reducir los efectos de
orientación ("optimización de costos") y permitir la orientación del mercado ("optimización de velocidad")
para que podamos tener muchos equipos pequeños trabajando de forma segura e independiente, entregando rápidamente
valor para el cliente.

Llevado al extremo, los equipos orientados al mercado son responsables no solo de la función
desarrollo, pero también para probar, asegurar, implementar y respaldar su servicio en
producción, desde la concepción de la idea hasta la jubilación. Estos equipos están diseñados para ser cruzados

funcional e independiente: capaz de diseñar y ejecutar experimentos de usuarios, construir y entregar
nuevas características, despliegue y ejecute su servicio en producción, y repare cualquier defecto sin
dependencias manuales de otros equipos, lo que les permite moverse más rápido. Este modelo tiene
sido adoptado por Amazon y Netflix y Amazon lo promociona como uno de los principales
razones detrás de su capacidad de moverse rápido incluso a medida que crecen.

Para lograr la orientación al mercado, no haremos una gran reorganización de arriba hacia abajo, que a menudo
crea grandes cantidades de interrupción, miedo y parálisis. En su lugar, integraremos el
ingenieros funcionales y habilidades (por ejemplo, Ops, QA, Infosec) en cada equipo de servicio, o proporcionar
sus capacidades a los equipos a través de plataformas automáticas de autoservicio que brindan
entornos similares a la producción, iniciar pruebas automatizadas o realizar implementaciones.

Esto permite que cada equipo de servicio brinde valor de forma independiente al cliente sin
tener que abrir tickets con otros grupos, como IT Operations, QA o Infosec. ** **

HACIENDO TRABAJOS DE ORIENTACIÓN FUNCIONALES

Habiendo recomendado equipos orientados al mercado, vale la pena señalar que es
posible crear organizaciones efectivas de alta velocidad con orientación funcional. Cruzar-
Los equipos funcionales y orientados al mercado son una forma de lograr un flujo rápido y confiabilidad, pero
No son el único camino. También podemos lograr los resultados deseados de DevOps a través de
orientación funcional, siempre y cuando todos en el flujo de valor vean al cliente y
resultados organizacionales como un objetivo compartido, independientemente de dónde residan en el
organización.

Figura 12: *orientación funcional frente a mercado*

Izquierda: orientación funcional: todo el trabajo fluye a través de operaciones de TI centralizadas; Derecha: mercado orientación: todos los equipos de productos pueden implementar sus componentes de autoservicio sueltos en producción. (Fuente: Humble, Molesky y O'Reilly, Lean Enterprise *, edición Kindle, 4523 y 4592.)*

Por ejemplo, alto rendimiento con operaciones centralizadas y orientadas a la funcionalidad.
grupo es posible, siempre y cuando los equipos de servicio obtengan lo que necesitan de las operaciones de manera confiable y rápidamente (idealmente bajo demanda) y viceversa. Muchos de los DevOps más admirados
Las organizaciones conservan la orientación funcional de las operaciones, incluidos Etsy, Google y GitHub.

Lo que estas organizaciones tienen en común es una cultura de alta confianza que permite a todos
departamentos para trabajar juntos de manera efectiva, donde todo el trabajo tiene prioridad transparente y

hay suficiente holgura en el sistema para permitir que el trabajo de alta prioridad se complete rápidamente. Esto es, en parte, habilitado por plataformas automáticas de autoservicio que incorporan calidad en el productos que todos están construyendo.

En el movimiento de manufactura esbelta de la década de 1980, muchos investigadores quedaron perplejos por La orientación funcional de Toyota, que estaba en desacuerdo con la mejor práctica de tener cross-equipos funcionales, orientados al mercado. Estaban tan perplejos que se llamó "el segundo Toyota paradoja."

Como Mike Rother escribió en *Toyota Kata* : "Por tentador que parezca, uno no puede reorganizarse su camino hacia la mejora continua y la adaptabilidad. Lo decisivo no es la forma de la organización, pero cómo las personas actúan y reaccionan. Las raíces del éxito de Toyota no radican en su estructuras organizativas, pero en el desarrollo de capacidades y hábitos en su gente. Sorprende mucha gente, de hecho, para encontrar que Toyota está en gran medida organizada en un tradicional, funcional-estilo departamento ". Es este desarrollo de hábitos y capacidades en las personas y el mano de obra que es el foco de nuestras próximas secciones.

PRUEBAS, OPERACIONES Y SEGURIDAD COMO TODOS TRABAJO CADA DÍA

En las organizaciones de alto rendimiento, todos los miembros del equipo comparten un objetivo común: la calidad, la disponibilidad y la seguridad no son responsabilidad de los departamentos individuales, pero son parte del trabajo de todos, todos los días.

Esto significa que el problema más urgente del día puede ser trabajar o implementar un característica del cliente o reparación de un incidente de producción de Gravedad 1. Alternativamente, el día puede requieren revisar el cambio de un compañero ingeniero, aplicando parches de seguridad de emergencia para servidores de producción o mejoras para que los compañeros ingenieros sean más productivos.

Reflexionando sobre objetivos compartidos entre Desarrollo y Operaciones, Jody Mulkey, CTO en Ticketmaster dijo: "Durante casi 25 años, utilicé una metáfora del fútbol americano para describir Dev y Ops. Sabes, Ops es defensa, quien evita que el otro equipo anote, y Dev es ofensiva, tratando de marcar goles. Y un día, me di cuenta de cuán defectuosa era esta metáfora, porque nunca todos juegan en el campo al mismo tiempo. En realidad no están en el mismo ¡equipo!"

Él continuó: "La analogía que uso ahora es que Ops son los linieros ofensivos, y Dev son las posiciones de 'habilidad' (como el mariscal de campo y los receptores abiertos) cuyo trabajo es mover el bola abajo en el campo: el trabajo de Ops es ayudar a garantizar que Dev tenga suficiente tiempo para ejecutar las jugadas ".

Un ejemplo sorprendente de cómo el dolor compartido puede reforzar los objetivos compartidos es cuando Facebook fue experimentando un enorme crecimiento en 2009. Experimentaban problemas significativos relacionado con las implementaciones de código, aunque no todos los problemas causaron problemas que afectan al cliente, hay Fue crónico de extinción de incendios y largas horas. Pedro Canahuati, su director de producción. ingeniería, describió una reunión llena de ingenieros de operaciones donde alguien preguntó que todos las personas que no trabajan en un incidente cierran sus computadoras portátiles y nadie puede hacerlo.

Una de las cosas más importantes que hicieron para ayudar a cambiar los resultados de las implementaciones era hacer que todos los ingenieros, gerentes de ingeniería y arquitectos de Facebook rotaran

servicio de guardia por los servicios que construyeron. Al hacer esto, todos los que trabajaron en el servicio retroalimentación visceral experimentada sobre las decisiones arquitectónicas y de codificación aguas arriba que hecho, lo que tuvo un enorme impacto positivo en los resultados posteriores.

PERMITA QUE CADA MIEMBRO DEL EQUIPO SEA GENERALISTA

En casos extremos de una organización de Operaciones orientada funcionalmente, tenemos departamentos de especialistas, como administradores de red, administradores de almacenamiento, etc. Cuando los departamentos se especializan en *exceso* , provoca la *siloización* , que el Dr. Spear describe como cuando los departamentos "operan más como estados soberanos". Cualquier actividad operacional compleja entonces requiere múltiples transferencias y colas entre las diferentes áreas de la infraestructura,

lo que lleva a tiempos de entrega más largos (por ejemplo, porque cada cambio de red debe ser realizado por alguien en el departamento de redes).

Debido a que confiamos en un número cada vez mayor de tecnologías, debemos tener ingenieros que se han especializado y logrado el dominio en las áreas tecnológicas que necesitamos. Sin embargo, nos no quieren crear especialistas que estén "congelados en el tiempo", que solo comprendan y puedan contribuir a esa área del flujo de valor.

Una contramedida es permitir y alentar a cada miembro del equipo a ser generalista. Nosotros haga esto proporcionando oportunidades para que los ingenieros aprendan todas las habilidades necesarias para construir y ejecutar los sistemas de los que son responsables y rotar regularmente a las personas diferentes roles El término *ingeniero de pila completa* ahora se usa comúnmente (a veces como un rico fuente de parodia) para describir a los generalistas que están familiarizados, al menos tienen un nivel general de comprensión: con toda la pila de aplicaciones (por ejemplo, código de aplicación, bases de datos, sistemas operativos, redes, nube).

Tabla 2: *Especialistas vs. Generalistas vs. Personal "en forma de E" (experiencia, pericia, exploración y ejecución)*

(Fuente: Scott Prugh, "Entrega continua", ScaledAgileFramework.com, 14 de febrero de 2013, http://scaledagileframework.com/continuous-delivery/.)

Scott Prugh escribe que CSG International ha sufrido una transformación que trae La mayoría de los recursos necesarios para compilar y ejecutar el producto en un equipo, incluido el análisis, arquitectura, desarrollo, prueba y operaciones. "Por entrenamiento cruzado y crecimiento

habilidades de ingeniería, los generalistas pueden hacer mucho más trabajo que sus especialistas contrapartes, y también mejora nuestro flujo general de trabajo al eliminar colas y esperar hora." Este enfoque está en desacuerdo con las prácticas de contratación tradicionales, pero, como explica Prugh, vale la pena "Los gerentes tradicionales a menudo se opondrán a contratar ingenieros con generalistas conjuntos de habilidades, argumentando que son más caros y que 'puedo contratar dos servidores

administradores para cada ingeniero de operaciones multidisciplinar ". Sin embargo, el negocio
Los beneficios de permitir un flujo más rápido son abrumadores. Además, como señala Prugh,
“[Invertir] en capacitación cruzada es lo correcto para el crecimiento profesional [de los empleados] y hace que
el trabajo de todos es más divertido ".

Cuando valoramos a las personas simplemente por sus habilidades o desempeño existentes en su rol actual
en lugar de por su capacidad de adquirir y desplegar nuevas habilidades, nosotros (a menudo inadvertidamente)
Reforzar lo que la Dra. Carol Dweck describe como la *mentalidad fija* , donde las personas ven sus
inteligencia y habilidades como "donaciones" estáticas que no se pueden cambiar de manera significativa.

En cambio, queremos fomentar el aprendizaje, ayudar a las personas a superar la ansiedad de aprendizaje, ayudar
asegúrese de que las personas tengan habilidades relevantes y una hoja de ruta profesional definida, y así sucesivamente. Por
Al hacer esto, ayudamos a fomentar una *mentalidad de crecimiento* en nuestros ingenieros, después de todo, un aprendizaje
La organización requiere personas que estén dispuestas a aprender. Al alentar a todos a aprender, como
además de brindar capacitación y apoyo, creamos los más sostenibles y menos costosos
forma de crear grandeza en nuestros equipos, invirtiendo en el desarrollo de las personas que
ya tengo.

Como Jason Cox, Director de Ingeniería de Sistemas en Disney, describió: "Dentro de las operaciones,
tuvimos que cambiar nuestras prácticas de contratación. Buscamos personas que tuvieran 'curiosidad, coraje,
y franqueza, 'que no solo eran capaces de ser generalistas sino también renegados ... Queremos
para promover la interrupción positiva para que nuestro negocio no se atasque y pueda pasar a la
futuro." Como veremos en la siguiente sección, cómo financiamos a nuestros equipos también afecta nuestros resultados.

FONDO NO PROYECTOS, SINO SERVICIOS Y PRODUCTOS

Otra forma de permitir resultados de alto rendimiento es crear equipos de servicio estables con
financiación continua para ejecutar su propia estrategia y hoja de ruta de iniciativas. Estos equipos
tienen los ingenieros dedicados necesarios para cumplir con los compromisos concretos contraídos a nivel interno
y clientes externos, como características, historias y tareas.

Compare esto con el modelo más tradicional donde los equipos de Desarrollo y Prueba son
asignado a un "proyecto" y luego reasignado a otro proyecto tan pronto como el proyecto esté
completado y se agota el financiamiento. Esto conduce a todo tipo de resultados no deseados, incluidos
los desarrolladores no pueden ver las consecuencias a largo plazo de las decisiones que toman (un formulario
de retroalimentación) y un modelo de financiación que solo valora y paga las primeras etapas de la
ciclo de vida del software, que, trágicamente, es también la parte menos costosa para productos exitosos
o servicios.††

Nuestro objetivo con un modelo de financiación basado en productos es valorar el logro de la organización
y resultados del cliente, como ingresos, valor de por vida del cliente o adopción del cliente
tasa, idealmente con el mínimo de salida (por ejemplo, cantidad de esfuerzo o tiempo, líneas de código).
Compare esto con la forma en que se miden los proyectos, por ejemplo, si se completó
dentro del presupuesto prometido, el tiempo y el alcance.

DISEÑO DE LÍMITES DE EQUIPO DE ACUERDO CON LEY DE CONWAY

A medida que las organizaciones crecen, uno de los mayores desafíos es mantenerse eficaz comunicación y coordinación entre personas y equipos. Con demasiada frecuencia, cuando las personas y los equipos residen en un piso diferente, en un edificio diferente o en una zona horaria diferente, crear y mantener un entendimiento compartido y una confianza mutua se vuelve más difícil, impidiendo una colaboración efectiva. La colaboración también se ve obstaculizada cuando la primaria Los mecanismos de comunicación son tickets de trabajo y solicitudes de cambio, o peor, cuando los equipos están separados por límites contractuales, como cuando el trabajo lo realiza un subcontratado equipo.

Como vimos en el ejemplo de Etsy Sprouter al comienzo de este capítulo, la forma en que Organizar equipos puede generar malos resultados, un efecto secundario de la Ley de Conway. Éstos incluyen dividir equipos por función (por ejemplo, colocando desarrolladores y probadores en diferentes ubicaciones o subcontratando probadores por completo) o por capa arquitectónica (por ejemplo, aplicación, base de datos).

Estas configuraciones requieren una comunicación y coordinación significativas entre los equipos, pero todavía resulta en una gran cantidad de retrabajo, desacuerdos sobre las especificaciones, pobre traspasos, y personas sentadas ociosas esperando a alguien más.

Idealmente, nuestra arquitectura de software debería permitir que los equipos pequeños sean independientes productivo, suficientemente desacoplado entre sí para poder trabajar sin Comunicación y coordinación excesiva o innecesaria.

CREAR ARQUITECTURAS SOLAMENTE ACOPLADAS PARA HABILITAR LA PRODUCTIVIDAD Y LA SEGURIDAD DEL DESARROLLADOR

Cuando tenemos una arquitectura estrechamente acoplada, pequeños cambios pueden resultar en una gran escala fallas Como resultado, cualquier persona que trabaje en una parte del sistema debe coordinar constantemente con cualquier otra persona que trabaje en otra parte del sistema que puedan afectar, incluyendo navegando por procesos complejos y burocráticos de gestión del cambio.

Además, para probar que todo el sistema funciona en conjunto, se requieren cambios integrales con los cambios de cientos, o incluso miles, de otros desarrolladores, que pueden, en a su vez, dependen de decenas, cientos o miles de sistemas interconectados. Las pruebas se realizan en entornos de pruebas de integración escasos, que a menudo requieren semanas para obtener y configurar El resultado no es solo largos plazos de entrega para los cambios (generalmente medidos en semanas o meses), pero también baja productividad del desarrollador y malos resultados de implementación.

Por el contrario, cuando tenemos una arquitectura que permite a pequeños equipos de desarrolladores Implementamos, probamos e implementamos código de forma independiente en la producción de forma segura y rápida, podemos aumentar y mantener la productividad del desarrollador y mejorar los resultados de la implementación. Estas las características se pueden encontrar en *arquitecturas orientadas a servicios* (SOA) descritas por primera vez en Década de 1990, en la que los servicios se pueden probar y desplegar de forma independiente. Una característica clave de las SOA es que están compuestos de servicios *poco acoplados* con *contextos limitados* .

Tener una arquitectura débilmente acoplada significa que los servicios pueden actualizarse en producción independientemente, sin tener que actualizar otros servicios. Los servicios deben estar desconectados de

otros servicios y, igual de importante, de bases de datos compartidas (aunque pueden compartir un *servicio de* base de datos , siempre que no tengan ningún esquema común).

Los contextos limitados se describen en el libro *Domain Driven Design* de Eric J. Evans. los La idea es que los desarrolladores puedan entender y actualizar el código de un servicio sin saber nada sobre los aspectos internos de sus servicios pares. Los servicios interactúan con sus pares estrictamente a través de API y, por lo tanto, no comparten estructuras de datos, esquemas de bases de datos, u otras representaciones internas de objetos. Los contextos limitados aseguran que los servicios sean compartimentados y tienen interfaces bien definidas, lo que también permite pruebas más fáciles.

Randy Shoup, ex Director de Ingeniería de Google App Engine, observó que "Organizaciones con este tipo de arquitecturas orientadas a servicios, como Google y Amazon, tiene una increíble flexibilidad y escalabilidad. Estas organizaciones tienen decenas de miles de desarrolladores donde los equipos pequeños aún pueden ser increíblemente productivos ".

MANTENGA LOS TAMAÑOS DE EQUIPO PEQUEÑOS (LA REGLA DEL “EQUIPO DE DOS PIZZAS”)

La Ley de Conway nos ayuda a diseñar los límites de nuestro equipo en el contexto deseado patrones de comunicación, pero también nos alienta a mantener pequeños los tamaños de nuestro equipo, reduciendo la cantidad de comunicación entre equipos y nos anima a mantener el alcance de cada El dominio del equipo es pequeño y acotado.

Como parte de su iniciativa de transformación lejos de una base de código monolítico en 2002, Amazon usó la regla de las *dos pizzas* para mantener pequeños los tamaños de los equipos: un equipo tan grande como se pueda alimentar con dos pizzas, generalmente de cinco a diez personas.

Este límite de tamaño tiene cuatro efectos importantes:

1. Asegura que el equipo tenga una comprensión clara y compartida del sistema en el que están trabajando en. A medida que los equipos crecen, la cantidad de comunicación requerida para que todos sepan lo que sucede en escalas de forma combinatoria.

2. Limita la tasa de crecimiento del producto o servicio en el que se trabaja. Al limitar el tamaño del equipo, limitamos la velocidad a la que su sistema puede evolucionar. Esto también ayuda a garantizar El equipo mantiene una comprensión compartida del sistema.

3. Descentraliza el poder y permite la autonomía. Cada equipo de dos pizzas (2PT) es como lo más autónomo posible El líder del equipo, trabajando con el equipo ejecutivo, decide sobre la métrica comercial clave de la que es responsable el equipo, conocida como la función de acondicionamiento físico, que se convierte en el criterio general de evaluación para los experimentos del equipo. El equipo es entonces capaz de actuar de forma autónoma para maximizar esa métrica. [§§]

4. Liderar un 2PT es una forma para que los empleados obtengan cierta experiencia de liderazgo en un entorno donde el fracaso no tiene consecuencias catastróficas. Un esencial El elemento de la estrategia de Amazon fue el vínculo entre la estructura organizativa de un 2PT y el enfoque arquitectónico de una arquitectura orientada a servicios.

El CTO de Amazon Werner Vogels explicó las ventajas de esta estructura a Larry Dignan de *Línea* de *base* en 2005. Dignan escribe:

> "Los equipos pequeños son rápidos ... y no se atascan en la llamada administración ... cada uno grupo asignado a un negocio en particular es completamente responsable de ello ... El equipo

> mira el arreglo, lo diseña, lo construye, lo implementa y monitorea su uso continuo. Esta De esta manera, los programadores y arquitectos de tecnología obtienen retroalimentación directa de la empresa personas que usan su código o aplicaciones, en reuniones regulares e informales conversaciones ".

Otro ejemplo de cómo la arquitectura puede mejorar profundamente la productividad es la API Programa de habilitación en Target, Inc.

> Target es el sexto minorista más grande de los EE. UU. Y gasta más de $ 1 mil millones en tecnología anualmente. Heather Mickman, directora de desarrollo de Target, describió el principios de su viaje DevOps: "En los viejos tiempos, solía tomar diez diferentes equipos para aprovisionar un servidor en Target, y cuando las cosas se rompieron, tendíamos a detenernos haciendo cambios para evitar más problemas, lo que, por supuesto, empeora todo ".
>
> Las dificultades asociadas con obtener entornos y realizar implementaciones creó dificultades significativas para los equipos de desarrollo, al igual que obtener acceso a los datos que necesitaban. Como describió Mickman:
>
> > El problema era que gran parte de nuestros datos principales, como la información sobre el inventario, los precios y las tiendas estaban encerrados en sistemas heredados y mainframes. Nosotros a menudo tenía múltiples fuentes de verdades de datos, especialmente entre el comercio electrónico y nuestro tiendas físicas, que eran propiedad de diferentes equipos, con diferentes datos estructuras y diferentes prioridades .... El resultado fue que si un nuevo desarrollo equipo quería construir algo para nuestros invitados, tomaría de tres a seis meses para construir las integraciones para obtener los datos que necesitaban. Peor aún, sería tomar otros tres a seis meses para hacer la prueba manual para asegurarse de que no rompió nada crítico, debido a la cantidad personalizada de punto a punto integraciones que tuvimos en un sistema muy bien acoplado. Tener que gestionar el interacciones con los veinte a treinta equipos diferentes, junto con todos sus dependencias, se requieren muchos gerentes de proyecto, debido a todas las coordinación y traspasos. Significaba que el desarrollo estaba gastando todo su tiempo esperando en colas, en lugar de entregar resultados y hacer cosas.
>
> Este largo tiempo de espera para recuperar y crear datos en sus sistemas de registro fue poner en peligro objetivos comerciales importantes, como la integración de la cadena de suministro operaciones de las tiendas físicas de Target y su sitio de comercio electrónico, que ahora requerían

conseguir inventario para tiendas y hogares de clientes. Esto empujó la cadena de suministro de Target mucho más allá de lo que fue diseñado, que era simplemente para facilitar el movimiento de bienes de vendedores a centros de distribución y tiendas.

En un intento por resolver el problema de los datos, en 2012 Mickman lideró la habilitación de API equipo para permitir que los equipos de desarrollo "entreguen nuevas capacidades en días en lugar de meses." Querían que cualquier equipo de ingeniería dentro de Target pudiera obtener y almacenar los datos que necesitaban, como información sobre sus productos o sus tiendas, incluidas las horas de operación, la ubicación, si hubo Starbucks en el sitio, y así adelante.

Las limitaciones de tiempo jugaron un papel importante en la selección del equipo. Mickman explicó que:

> Debido a que nuestro equipo también necesitaba entregar capacidades en días, no meses, yo necesitábamos un equipo que pudiera hacer el trabajo, no dárselo a los contratistas; queríamos personas con habilidades ingeniosas de ingeniería, no personas que supieran manejar contratos Y para asegurarnos de que nuestro trabajo no estaba en la cola, necesitábamos poseer toda la pila, lo que significa que asumimos los requisitos de Ops como bueno ... Trajimos muchas herramientas nuevas para apoyar la integración continua y entrega continua Y porque sabíamos que si teníamos éxito, lo haríamos tenemos que escalar con un crecimiento extremadamente alto, trajimos nuevas herramientas como el Base de datos Cassandra y agente de mensajes Kafka. Cuando pedimos permiso, nos dijeron que no, pero lo hicimos de todos modos, porque sabíamos que lo necesitaba.

En los siguientes dos años, el equipo de API Enablement permitió cincuenta y tres nuevos capacidades comerciales, incluido el envío a la tienda y el registro de regalos, así como su integraciones con Instacart y Pinterest. Como describió Mickman, "Trabajar con Pinterest de repente se volvió muy fácil, porque les proporcionamos nuestras API ".

En 2014, el equipo de API Enablement atendió más de 1.500 millones de llamadas API por mes. Por 2015, esto había crecido a diecisiete mil millones de llamadas por mes que abarca noventa diferentes APIs. Para soportar esta capacidad, rutinariamente realizaron ochenta implementaciones por semana.

Estos cambios han creado importantes beneficios comerciales para Target: ventas digitales aumentó 42% durante la temporada de vacaciones de 2014 y aumentó otro 32% en el segundo trimestre. Durante el fin de semana del Black Friday de 2015, se registraron más de 280 mil pedidos de recogida en la tienda creado. Para 2015, su objetivo es permitir que 450 de sus 1,800 tiendas puedan cumplir pedidos de comercio electrónico, por encima de cien.

"El equipo de API Enablement muestra lo que puede hacer un equipo de agentes de cambio apasionados hacer ", dice Mickman. "Y nos ayuda a prepararnos para la siguiente etapa, que es expandir DevOps en toda la organización tecnológica ".

CONCLUSIÓN

A través de los estudios de caso de Etsy y Target, podemos ver cómo la arquitectura y la organización El diseño puede mejorar drásticamente nuestros resultados. Hecho incorrectamente, la Ley de Conway Asegurar que la organización cree malos resultados, previniendo la seguridad y la agilidad. Hecho bueno, la organización permite a los desarrolladores desarrollar, probar y desplegar valor para el cliente.

† Entre muchas cosas, un ORM abstrae una base de datos, lo que permite a los desarrolladores hacer consultas y manipulación de datos como si fueran simplemente otro objeto en el lenguaje de programación. Los ORM populares incluyen Hibernate para Java, SQLAlchemy para Python y ActiveRecord para Ruby on Rails.

‡ Sprouter fue una de las muchas tecnologías utilizadas en el desarrollo y la producción que Etsy eliminó como parte de su transformación.

§ Sin embargo, como se explicará más adelante, organizaciones igualmente importantes como Etsy y GitHub tienen orientación funcional.

¶ Adrian Cockcroft comentó: "Para las compañías que ahora están saliendo de contratos de outsourcing de TI de cinco años, es como si se hubieran congelado a tiempo, durante un de los tiempos más disruptivos en tecnología ". En otras palabras, la externalización de TI es una táctica utilizada para controlar los costos a través de la estasis ejecutada contractualmente, con precios fijos firmes que programan reducciones anuales de costos. Sin embargo, a menudo resulta en que las organizaciones no pueden responder a los cambios de negocios y Necesidades tecnológicas.

** Para el resto de estos libros, utilizaremos *equipos de servicio de manera* intercambiable con *equipos de características* , *equipos de productos* , *equipos de desarrollo* y *equipos de entrega* . La intención es especificar el equipo principalmente desarrollando, probando y asegurando el código para que el valor se entregue al cliente.

†† Como John Lauderbach, actualmente vicepresidente de tecnología de la información en los supermercados Roche Bros., bromeó: "Cada nueva aplicación es como un cachorro gratis. No es el costo de capital inicial que lo mata ... Es el mantenimiento y soporte continuos ".

‡‡ Estas propiedades también se encuentran en los "microservicios", que se basan en los principios de SOA. Un conjunto popular de patrones para la arquitectura web moderna basado en estos principios es la "aplicación de 12 factores".

§§ En la cultura de Netflix, uno de los siete valores clave es "altamente alineado, vagamente acoplado".

8 Resultados por Integrando Operaciones en el Trabajo diario de Desarrollo

Cómo ser genial

Nuestro objetivo es permitir resultados orientados al mercado donde muchos equipos pequeños puedan entregar valor de forma rápida e independiente al cliente. Esto puede ser un desafío a alcanzar cuando las operaciones están centralizadas y funcionalmente orientado, teniendo que atender las necesidades de muchos equipos de desarrollo diferentes con necesidades potencialmente muy diferentes. El resultado a menudo puede ser largo plomo tiempos para el trabajo necesario de Ops, repriorización y escalado constante, y

pobres resultados de despliegue.
Podemos crear resultados más orientados al mercado integrando mejor Ops
capacidades en equipos de desarrollo, lo que hace que sean más eficientes y productivos. En
En este capítulo, exploraremos muchas formas de lograr esto, tanto en el
nivel organizacional y a través de rituales diarios. Al hacer esto, Ops puede
Mejorar significativamente la productividad de los equipos de desarrollo en todo
organización, así como permitir una mejor colaboración y organización
resultados.

En Big Fish Games, que desarrolla y admite cientos de dispositivos móviles y
miles de juegos de PC y obtuvieron más de $ 266 millones en ingresos en 2013,
El vicepresidente de operaciones de TI, Paul Farrall, estaba a cargo de las operaciones centralizadas.
organización. Fue responsable de apoyar muchos negocios diferentes
unidades que tenían mucha autonomía.

Cada una de estas unidades de negocios tenía equipos de desarrollo dedicados que a menudo
eligió tecnologías muy diferentes. Cuando estos grupos querían desplegarse
nueva funcionalidad, tendrían que competir por un grupo común de escasos
Recursos de operaciones. Además, todos estaban luchando con una prueba poco confiable
y entornos de integración, así como una versión extremadamente engorrosa
procesos.

Farrall pensó que la mejor manera de resolver este problema era incrustando Ops
experiencia en equipos de desarrollo. Él observó: "Cuando los equipos de desarrollo tenían
problemas con las pruebas o la implementación, necesitaban más que solo tecnología
o ambientes. Lo que también necesitaban era ayuda y entrenamiento. Al principio, nosotros
ingenieros y arquitectos de Ops integrados en cada uno de los equipos de desarrollo, pero hay
simplemente no había suficientes ingenieros de operaciones para cubrir tantos equipos. Éramos
capaz de ayudar a más equipos con lo que llamamos un modelo de *enlace de Ops* y con
menos gente."

Farrall definió dos tipos de enlaces de operaciones: el gerente de relaciones comerciales
y el ingeniero de lanzamiento dedicado. Los gerentes de relaciones comerciales
trabajó con gestión de productos, propietarios de línea de negocio, proyecto
gestión, gestión de desarrollo y desarrolladores. Se volvieron íntimamente
familiarizado con los conductores de negocios del grupo de productos y las hojas de ruta de productos, actuó

como defensores de los propietarios de productos dentro de Operaciones, y ayudaron a sus los equipos de productos navegan por el panorama de Operaciones para priorizar y agilizar las solicitudes de trabajo.

Del mismo modo, el ingeniero de lanzamiento dedicado se familiarizó íntimamente con el Desarrollo de productos y problemas de control de calidad, y les ayudó a obtener lo que necesario de la organización Ops para lograr sus objetivos. Estaban familiarizados con las típicas solicitudes de Dev y QA para Ops, y a menudo ejecutaba el Necesitaban trabajar ellos mismos. Según sea necesario, también atraerían a personas dedicadas ingenieros técnicos de operaciones (p. ej., DBA, Infosec, ingenieros de almacenamiento, redes ingenieros), y ayudan a determinar qué herramientas de autoservicio funcionan todas las operaciones El grupo debe priorizar la construcción.

Al hacer esto, Farrall pudo ayudar a los equipos de desarrollo en toda la organización ser más productivo y lograr sus objetivos de equipo. Además, él ayudó a los equipos a priorizar alrededor de sus restricciones globales de operaciones, reduciendo el Número de sorpresas descubiertas a mitad del proyecto y, en última instancia, aumentando el rendimiento general del proyecto.

Farrall señala que ambas relaciones de trabajo con Operaciones y código la velocidad de liberación mejoró notablemente como resultado de los cambios. Él concluye: “El modelo de enlace de Ops nos permitió integrar las operaciones de TI experiencia en los equipos de desarrollo y producto sin agregar nuevos empleados ".

La transformación DevOps en Big Fish Games muestra cómo un sistema centralizado El equipo de operaciones pudo lograr los resultados típicamente asociados con equipos orientados al mercado. Podemos emplear las tres estrategias generales siguientes:

Cree capacidades de autoservicio para permitir a los desarrolladores en los equipos de servicio ser productivo

Incruste a los ingenieros de Ops en los equipos de servicio.

Asigne enlaces de Ops a los equipos de servicio cuando incrustar Ops no sea posible.

Por último, describimos cómo los ingenieros de Ops pueden integrarse en el equipo de desarrollo

rituales utilizados en su trabajo diario, incluidas las paradas diarias, la planificación y retrospectivas

CREA SERVICIOS COMPARTIDOS PARA AUMENTAR DESARROLLADOR PRODUCTIVIDAD

Una forma de permitir resultados orientados al mercado es que las Operaciones creen un conjunto de plataformas centralizadas y servicios de herramientas que cualquier equipo de desarrollo puede usar para ser más productivo, como obtener entornos de producción, tuberías de implementación, herramientas de prueba automatizadas, telemetría de producción paneles de instrumentos, y así sucesivamente.† Al hacer esto, permitimos que los equipos de desarrollo gasten más funcionalidad de creación de tiempo para su cliente, en lugar de obtener todos los infraestructura requerida para entregar y soportar esa característica en la producción.

Todas las plataformas y servicios que brindamos deberían (idealmente) ser automatizados y disponible bajo demanda, sin requerir que un desarrollador abra un ticket y esperar a que alguien realice el trabajo manualmente. Esto asegura que las operaciones no se convierte en un cuello de botella para sus clientes (por ejemplo, "Recibimos su solicitud de trabajo, y tomará seis semanas configurar manualmente esas pruebas ambientes "). ‡

Al hacer esto, permitimos que los equipos de productos obtengan lo que necesitan, cuando lo necesita, así como también reduce la necesidad de comunicaciones y coordinación. Como Damon Edwards observó: "Sin estas plataformas de operaciones de autoservicio, la nube es simplemente Hosting 2.0 caro ”.

En casi todos los casos, no exigiremos que los equipos internos usen estos plataformas y servicios: estos equipos de plataforma tendrán que ganarse y satisfacer a sus clientes internos, a veces incluso compitiendo con externos vendedores. Al crear este mercado interno efectivo de capacidades, nosotros ayudar a garantizar que las plataformas y servicios que creamos sean los más fáciles y La opción más atractiva disponible (el camino de menor resistencia).

Por ejemplo, podemos crear una plataforma que proporcione una versión compartida repositorio de control con bibliotecas de seguridad previamente bendecidas, una tubería de implementación

que ejecuta automáticamente la calidad del código y las herramientas de escaneo de seguridad, que implementa nuestras aplicaciones en *entornos conocidos y buenos* que ya tienen herramientas de monitoreo de producción instaladas en ellos. Idealmente, hacemos mucho la vida más fácil para los equipos de desarrollo que decidirán abrumadoramente que usar nuestro la plataforma es el medio más fácil, seguro y seguro para obtener su aplicaciones en producción.

Construimos en estas plataformas la experiencia acumulativa y colectiva de todos en la organización, incluidos QA, Operaciones e Infosec, que Ayuda a crear un sistema de trabajo cada vez más seguro. Esto aumenta el desarrollador productividad y facilita que los equipos de productos aprovechen lo común procesos, como realizar pruebas automatizadas y satisfacer la seguridad y requisitos de conformidad.

Crear y mantener estas plataformas y herramientas es un producto real desarrollo: los clientes de nuestra plataforma no son nuestros clientes externos pero nuestros equipos internos de desarrollo. Como crear un gran producto, crear un gran Las plataformas que todos aman no suceden por accidente. Un interno El equipo de plataforma con poca atención al cliente probablemente creará herramientas que todos odiará y abandonará rápidamente por otras alternativas, ya sea por otra equipo de plataforma interna o un proveedor externo.

Dianne Marsh, directora de herramientas de ingeniería en Netflix, afirma que ella El estatuto del equipo es "apoyar la innovación y la velocidad de nuestros equipos de ingeniería. No construimos, horneamos ni implementamos nada para estos equipos, ni gestionamos sus configuraciones En cambio, creamos herramientas para permitir el autoservicio. Está bien para que las personas dependan de nuestras herramientas, pero es importante que no volverse dependiente de nosotros ".

A menudo, estos equipos de plataforma proporcionan otros servicios para ayudar a sus clientes. aprender su tecnología, migrar de otras tecnologías e incluso proporcionar Coaching y consultoría para ayudar a elevar el estado de la práctica dentro del organización. Estos servicios compartidos también facilitan la estandarización, que permitir a los ingenieros ser productivos rápidamente, incluso si cambian entre equipos Por ejemplo, si cada equipo de producto elige una cadena de herramientas diferente, los ingenieros pueden tener que aprender un conjunto completamente nuevo de tecnologías para hacer su trabajo, colocando los objetivos del equipo por delante de los objetivos globales.

En organizaciones donde los equipos solo pueden usar herramientas aprobadas, podemos comenzar por eliminar este requisito para algunos equipos, como el equipo de transformación,

para que podamos experimentar y descubrir qué capacidades hacen esos equipos
más productivo.

Los equipos internos de servicios compartidos deben buscar continuamente
cadenas de herramientas que se están adoptando ampliamente en la organización, decidiendo qué
los que tienen sentido son apoyados centralmente y disponibles para todos.
En general, tomar algo que ya está funcionando en alguna parte y
expandir su uso es mucho más probable que tenga éxito que construir estos
capacidades desde cero.§

INGENIEROS DE OPS EMPOTRADOS EN NUESTRO SERVICIO EQUIPOS

Otra forma en que podemos permitir resultados más orientados al mercado es habilitar
los equipos de productos se volverán más autosuficientes incorporando Operaciones
ingenieros dentro de ellos, reduciendo así su dependencia de centralizado
Operaciones Estos equipos de productos también pueden ser completamente responsables de
prestación de servicios y soporte de servicios.

Al incorporar ingenieros de operaciones en los equipos de desarrollo, sus prioridades son
impulsado casi por completo por los objetivos de los equipos de productos en los que están integrados
en - en lugar de que Ops se centre internamente en resolver sus propios problemas. Como un
Como resultado, los ingenieros de Ops se conectan más estrechamente con sus
Clientes externos. Además, los equipos de productos a menudo tienen el presupuesto.
para financiar la contratación de estos ingenieros de Ops, aunque entrevistando y contratando
las decisiones probablemente se tomarán del grupo de Operaciones centralizado, para
Garantizar la coherencia y la calidad del personal.

Jason Cox dijo: "En muchas partes de Disney hemos incorporado Ops (sistema
ingenieros) dentro de los equipos de productos en nuestras unidades de negocio, junto con
Desarrollo, prueba e incluso seguridad de la información. Ha cambiado totalmente el
dinámica de cómo trabajamos. Como ingenieros de operaciones, creamos las herramientas y
capacidades que transforman la forma en que trabajan las personas, e incluso la forma en que
pensar. En las operaciones tradicionales, simplemente conducíamos el tren que alguien más construyó.
Pero en la Ingeniería de Operaciones moderna, no solo ayudamos a construir el tren, sino que también
también los puentes sobre los que ruedan los trenes ".

Para nuevos proyectos de desarrollo grandes, inicialmente podemos incorporar ingenieros de Ops
en esos equipos. Su trabajo puede incluir ayudar a decidir qué construir y
cómo construirlo, influyendo en la arquitectura del producto, ayudando a influir
opciones tecnológicas internas y externas, que ayudan a crear nuevas capacidades en

nuestras plataformas internas, y tal vez incluso generando nuevas operaciones

capacidades. Después de que el producto se lanza a producción, Ops incrustado
Los ingenieros pueden ayudar con las responsabilidades de producción del equipo de desarrollo.

Participarán en todos los rituales del equipo de desarrollo, como la planificación de reuniones,
Standups diarios y demostraciones donde el equipo muestra nuevas características
y decide cuáles enviar. Como la necesidad de conocimiento de Ops y
las capacidades disminuyen, los ingenieros de Ops pueden hacer la transición a diferentes proyectos o
compromisos, siguiendo el patrón general que la composición dentro de
los equipos de productos cambian a lo largo de su ciclo de vida.

Este paradigma tiene otra ventaja importante: emparejar Dev y Ops
ingenieros juntos es una forma extremadamente eficiente de realizar operaciones de entrenamiento cruzado
conocimiento y experiencia en un equipo de servicio. También puede tener el poderoso
beneficio de transformar el conocimiento de operaciones en código automatizado que puede
Ser mucho más confiable y ampliamente reutilizado.

ASIGNAR UN ENLACE OPS A CADA SERVICIO EQUIPO

Por una variedad de razones, como el costo y la escasez, es posible que no podamos
incrustar a los ingenieros de Ops en cada equipo de producto. Sin embargo, podemos obtener muchos de
los mismos beneficios al asignar un enlace designado para cada equipo de producto.

En Etsy, este modelo se llama "operaciones designadas". Sus operaciones centralizadas
group continúa administrando todos los entornos, no solo la producción
entornos, pero también entornos de preproducción, para ayudar a garantizar que
permanece consistente El ingeniero designado de Ops es responsable de
comprensión:

- Cuál es la funcionalidad del nuevo producto y por qué lo estamos construyendo
- Cómo funciona en lo que respecta a la operabilidad, escalabilidad y observabilidad
  (se recomienda encarecidamente la diagramación)
- Cómo monitorear y recopilar métricas para garantizar el progreso, el éxito o
  falla de la funcionalidad

Cualquier desviación de arquitecturas y patrones anteriores, y el justificaciones para ellos

Cualquier necesidad adicional de infraestructura y cómo afectará el uso capacidad de infraestructura

Planes de lanzamiento de funciones

Además, al igual que en el modelo Ops incorporado, este enlace asiste a la standups de equipo, integrando sus necesidades en la hoja de ruta de Operaciones y realizar cualquier tarea necesaria. Confiamos en estos enlaces para escalar cualquier problema de contención de recursos o priorización. Al hacer esto, identificamos cualquier conflictos de recursos o tiempo que deben evaluarse y priorizarse en el contexto de objetivos organizacionales más amplios.

Asignar enlaces Ops nos permite apoyar más equipos de productos que el modelo Ops integrado. Nuestro objetivo es garantizar que Ops no sea una restricción para Los equipos de producto. Si descubrimos que los enlaces Ops se estiran demasiado, evitando que los equipos de producto logren sus objetivos, entonces probablemente necesita reducir la cantidad de equipos que cada enlace admite o incrustar temporalmente un ingeniero de operaciones en equipos específicos.

INTEGRAR OPS EN RITUALES DE DEV

Cuando los ingenieros de Ops están integrados o asignados como enlaces en nuestro producto equipos, podemos integrarlos en los rituales de nuestro equipo de desarrollo. En esta sección, nuestro El objetivo es ayudar a los ingenieros de Ops y otros no desarrolladores a comprender mejor la cultura de desarrollo existente e integrarlos de manera proactiva en todos aspectos de planificación y trabajo diario. Como resultado, Operaciones está mejor capacitada para planificar e irradiar cualquier conocimiento necesario a los equipos de productos, influyendo trabajar mucho antes de que entre en producción. Las siguientes secciones describen Algunos de los rituales estándar utilizados por los equipos de desarrollo que utilizan ágil métodos y cómo integraríamos a los ingenieros de Ops en ellos. De ninguna manera Las prácticas ágiles son un requisito previo para este paso: como ingenieros de Ops, nuestro objetivo es

para descubrir qué rituales siguen los equipos de productos, integrarlos y agregar valor a ellos.¶

Como observó Ernest Mueller: "Creo que DevOps funciona mucho mejor si
Los equipos de operaciones adoptan los mismos rituales ágiles que los equipos de desarrollo han utilizado:
hemos tenido éxitos fantásticos resolviendo muchos problemas asociados con Ops
puntos débiles, además de integrarse mejor con los equipos de desarrollo ".

INVITE A OPS A NUESTROS STANDUPS DE DEV

Uno de los rituales de desarrollo popularizados por Scrum es el standup diario, un rápido
reunión donde todos en el equipo se reúnen y presentan a cada uno
otras tres cosas: lo que se hizo ayer, lo que se hará hoy,
y lo que te impide hacer tu trabajo. ** **

Page 136

El propósito de esta ceremonia es irradiar información a todo el equipo.
y entender el trabajo que se está haciendo y se va a hacer. Por
haciendo que los miembros del equipo se presenten esta información, aprendemos sobre
cualquier tarea que esté experimentando obstáculos y descubra formas de ayudar a cada uno
otros mueven nuestro trabajo hacia la finalización. Además, al tener gerentes
En la actualidad, podemos resolver rápidamente la priorización y los conflictos de recursos.

Un problema común es que esta información está compartimentada dentro de
Equipo de desarrollo. Al hacer que asistan los ingenieros de Ops, las operaciones pueden ganar un
conocimiento de las actividades del equipo de desarrollo, lo que permite una mejor planificación
y preparación, por ejemplo, si descubrimos que el equipo del producto es
planeando un gran despliegue de funciones en dos semanas, podemos asegurarnos de que
personas y recursos están disponibles para apoyar el lanzamiento. Alternativamente, nosotros
puede resaltar áreas donde se necesita una interacción más cercana o más preparación
(por ejemplo, crear más comprobaciones de supervisión o scripts de automatización). Al hacer esto,
Creamos las condiciones donde las operaciones pueden ayudar a resolver nuestro equipo actual
problemas (por ejemplo, mejorar el rendimiento ajustando la base de datos, en lugar de
código de optimización) o problemas futuros antes de que se conviertan en una crisis (por ejemplo,
creando más entornos de prueba de integración para permitir las pruebas de rendimiento).

INVITE A OPS A NUESTRAS RETROSPECTIVAS DE DEV

Otro ritual ágil generalizado es la retrospectiva. Al final de cada
intervalo de desarrollo, el equipo discute qué fue exitoso, qué podría ser

mejorado, y cómo incorporar los éxitos y mejoras en el futuro
iteraciones o proyectos. El equipo tiene ideas para mejorar las cosas.
y revisa los experimentos de la iteración anterior. éste es uno de
mecanismos primarios donde el aprendizaje organizacional y el desarrollo de
ocurren contramedidas, con el trabajo resultante implementado inmediatamente o
agregado a la cartera de pedidos del equipo.

Tener ingenieros de Ops que asistan a las retrospectivas de nuestro equipo de proyecto significa que pueden
También se benefician de cualquier nuevo aprendizaje. Además, cuando hay un
despliegue o lanzamiento en ese intervalo, las operaciones deben presentar el
resultados y cualquier aprendizaje resultante, creando retroalimentación en el producto
equipo. Al hacer esto, podemos mejorar cómo se planifica el trabajo futuro y
realizado, mejorando nuestros resultados. Ejemplos de retroalimentación que Operaciones
puede traer a una retrospectiva incluyen:

> "Hace dos semanas, encontramos un punto ciego de monitoreo y acordamos cómo
> arreglalo. Funcionó. Tuvimos un incidente el martes pasado y pudimos
> detectarlo y corregirlo rápidamente antes de que los clientes se vean afectados ".

> "El despliegue de la semana pasada fue uno de los más difíciles y largos que hemos tenido
> tenido en más de un año. Aquí hay algunas ideas sobre cómo se puede mejorar ".

> "La campaña de promoción que hicimos la semana pasada fue mucho más difícil que nosotros
> pensé que sería, y probablemente no deberíamos hacer una oferta como esa
> de nuevo. Aquí hay algunas ideas sobre otras ofertas que podemos hacer para lograr nuestro
> metas."

> "Durante la última implementación, el mayor problema que tuvimos fue nuestro firewall
> las reglas ahora tienen miles de líneas, lo que lo hace extremadamente difícil y
> arriesgado de cambiar. Necesitamos rediseñar cómo evitamos las personas no autorizadas.
> tráfico de red."

Los comentarios de Operaciones ayudan a nuestros equipos de productos a ver y mejorar
Comprender el impacto aguas abajo de las decisiones que toman. Cuando hay
resultados negativos, podemos hacer los cambios necesarios para prevenirlos en
el futuro. La retroalimentación de operaciones también probablemente identificará más problemas y
defectos que deben repararse, incluso puede descubrir problemas arquitectónicos más grandes

eso debe ser abordado.

El trabajo adicional identificado durante las retrospectivas del equipo del proyecto cae en la amplia categoría de trabajo de mejora, como la reparación de defectos, refactorización, y automatizar el trabajo manual. Los gerentes de producto y gerentes de proyecto pueden querer diferir o priorizar el trabajo de mejora a favor del cliente caracteristicas.

Sin embargo, debemos recordar a todos que la mejora del trabajo diario es más importante que el trabajo diario en sí, y que todos los equipos deben haber dedicado capacidad para esto (por ejemplo, reservar el 20% de todos los ciclos para el trabajo de mejora, programación de un día por semana o una semana por mes, etc.). Sin hacer esto, la productividad del equipo seguramente se detendrá bajo El peso de su propia deuda técnica y de proceso.

HAGA VISIBLE EL TRABAJO OPS RELEVANTE EN EL KANBAN COMPARTIDO JUNTAS

A menudo, los equipos de desarrollo harán visible su trabajo en un tablero de proyecto o tablero kanban Sin embargo, es mucho menos común que los tableros de trabajo muestren Trabajo de operaciones relevante que debe realizarse para que el aplicación para ejecutarse con éxito en producción, donde el valor para el cliente es En realidad creado. Como resultado, no tenemos conocimiento del trabajo necesario de Operaciones hasta que se convierta en una crisis urgente, poniendo en peligro los plazos o creando un corte de producción.

Debido a que las operaciones son parte del flujo de valor del producto, debemos poner el Trabajo de operaciones que es relevante para la entrega del producto en el kanban compartido tablero. Esto nos permite ver más claramente todo el trabajo requerido para mover nuestro codificar en producción, así como realizar un seguimiento de todo el trabajo de Operaciones requerido para Apoyar el producto. Además, nos permite ver dónde está el trabajo de Ops bloqueado y donde el trabajo necesita escalar, destacando las áreas donde podemos necesita mejorar.

Los tableros Kanban son una herramienta ideal para crear visibilidad, y la visibilidad es la clave. componente para reconocer e integrar adecuadamente el trabajo de Ops en todos los flujos de valor relevantes. Cuando lo hacemos bien, logramos orientarnos al mercado resultados, independientemente de cómo hayamos dibujado nuestros organigramas.

CONCLUSIÓN

A lo largo de este capítulo, exploramos formas de integrar Operaciones en el
trabajo diario de desarrollo, y vimos cómo hacer que nuestro trabajo sea más
visible para las operaciones. Para lograr esto, exploramos tres estrategias generales,
incluida la creación de capacidades de autoservicio para permitir a los desarrolladores en servicio
equipos para ser productivos, integrando ingenieros de Ops en los equipos de servicio,
y asignando enlaces de Ops a los equipos de servicio al incorporar Ops
Los ingenieros no fueron posibles. Por último, describimos cómo los ingenieros de Ops pueden
integrarse con el equipo de desarrollo a través de la inclusión en su trabajo diario, incluyendo
Standups diarios, planificación y retrospectivas.

PARTE II CONCLUSIÓN

En la Parte II: *Dónde comenzar* , exploramos una variedad de formas de pensar
Transformaciones de DevOps, incluido cómo elegir dónde comenzar, relevantes

aspectos de arquitectura y diseño organizacional, y cómo organizar nuestro equipos También exploramos cómo integrar Ops en todos los aspectos de Dev planificación y trabajo diario.

En la Parte III: La primera forma, *las prácticas técnicas de flujo* , ahora comenzaremos para explorar cómo implementar las prácticas técnicas específicas para realizar principios de flujo, que permiten el flujo rápido de trabajo desde el desarrollo hasta Operaciones sin causar caos e interrupción aguas abajo.

† Los términos *plataforma* , *servicio compartido* y *cadena de herramientas* se utilizarán indistintamente en este libro.

‡ Ernest Mueller observó: “En Bazaarvoice, el acuerdo fue que estos equipos de plataforma que fabrican herramientas aceptan requisitos, pero no trabajo de otros equipos ".

§ Después de todo, diseñar un sistema por adelantado para su reutilización es un modo de falla común y costoso de muchas arquitecturas empresariales.

¶ Sin embargo, si descubrimos que toda la organización de Desarrollo simplemente se sienta en sus escritorios todo el día sin hablar con cada uno otro, puede que tengamos que encontrar una forma diferente de involucrarlos, como comprarles el almuerzo, comenzar un club de lectura, turnarse para hacer Presentaciones de "almuerzo y aprendizaje", o conversaciones para descubrir cuáles son los mayores problemas de todos, para que podamos entender descubrimos cómo podemos mejorar sus vidas.

** Scrum es una metodología de desarrollo ágil, descrita como "una estrategia de desarrollo de producto flexible y holística donde un desarrollo el equipo trabaja como una unidad para alcanzar un objetivo común ". Fue descrito por primera vez por completo por Ken Schwaber y Mike Beedle en el libro *Agile Desarrollo de software con Scrum* . En este libro, utilizamos el término "desarrollo ágil" o "desarrollo iterativo" para abarcar Las diversas técnicas utilizadas por metodologías especiales como Agile y Scrum.

Parte

Introducción

En la Parte III, nuestro objetivo es crear las prácticas técnicas.
y arquitectura requerida para habilitar y sostener lo rápido
flujo de trabajo del desarrollo a las operaciones sin
causando caos e interrupción a la producción
Medio ambiente o nuestros clientes. Esto significa que necesitamos
reducir el riesgo asociado con el despliegue y la liberación
cambios en producción. Haremos esto por
implementar un conjunto de prácticas técnicas conocidas como
entrega continua .

La entrega continua incluye la creación de las bases de
nuestra línea de implementación automatizada, asegurando que
tenemos pruebas automatizadas que validan constantemente que somos
en un estado desplegable, haciendo que los desarrolladores integren sus
codificar en el tronco diariamente y diseñar nuestros entornos
y código para habilitar versiones de bajo riesgo. Enfoques primarios
dentro de estos capítulos incluyen:

Creando la base de nuestra tubería de implementación

Permitir pruebas automatizadas rápidas y confiables

Permitir y practicar la integración continua y
pruebas

Automatización, habilitación y arquitectura para bajo riesgo
lanzamientos

La implementación de estas prácticas reduce el tiempo de espera para
obtener entornos de producción, permite continuo
pruebas que brindan a todos comentarios rápidos sobre su trabajo,
permite a pequeños equipos desarrollar de forma segura e independiente,
probar e implementar su código en producción, y hace
implementaciones de producción y libera una parte rutinaria de
trabajo diario.

Además, integrando los objetivos de QA y
Las operaciones en el trabajo diario de todos se reducen
lucha contra incendios, dificultades y trabajo duro, al tiempo que hace que las personas
Más alegría productiva y creciente en el trabajo que hacemos.
No solo mejoramos los resultados, sino que nuestra organización es
mejor capaz de ganar en el mercado.

9 Fundamentos de nuestro despliegue Tubería

Para crear un flujo rápido y confiable de Dev a Ops, debemos asegurarnos de que siempre utilizamos entornos de producción en cada etapa del valor corriente. Además, estos entornos deben crearse de forma automatizada. de manera ideal, a pedido de scripts e información de configuración almacenada en control de versiones, y totalmente autoservicio, sin ningún trabajo manual requerido de Operaciones. Nuestro objetivo es asegurarnos de que podamos recrear el entorno de producción completo basado en lo que hay en el control de versiones.

Con demasiada frecuencia, la única vez que descubrimos cómo funcionan nuestras aplicaciones en cualquier cosa que se parezca a un entorno de producción es durante la producción implementación: demasiado tarde para corregir problemas sin que el cliente sea impactado negativamente. Un ejemplo ilustrativo del espectro de problemas que puede ser causado por aplicaciones y entornos construidos inconsistentemente Programa Enterprise Data Warehouse dirigido por Em Campbell-Pretty en general Empresa australiana de telecomunicaciones en 2009. Campbell-Pretty se convirtió en el gerente general y patrocinador comercial de este programa de $ 200 millones, heredar la responsabilidad de todos los objetivos estratégicos que se basaron en esto plataforma.

En su presentación en la Cumbre Empresarial DevOps 2014, Campbell-Pretty explicó: “En ese momento, había diez flujos de trabajo en progreso, todos utilizando procesos en cascada, y las diez corrientes estaban significativamente atrasadas. Solo una de las diez transmisiones había alcanzado con éxito la Prueba de aceptación del usuario

[UAT] según lo programado, y tardó otros seis meses para que esa transmisión
UAT completa, con la capacidad resultante que está muy por debajo de los negocios
Expectativas. Este bajo rendimiento fue el principal catalizador para el
transformación ágil del departamento ".

Sin embargo, después de usar Agile durante casi un año, experimentaron solo pequeñas
mejoras, aún por debajo de sus resultados comerciales necesarios.
Campbell-Pretty realizó una retrospectiva de todo el programa y preguntó: "Después

Reflexionando sobre todas nuestras experiencias durante el último lanzamiento, ¿cuáles son las cosas que podríamos
¿eso duplicaría nuestra productividad?

A lo largo del proyecto, hubo quejas sobre la "falta de negocios
compromiso." Sin embargo, durante la retrospectiva, "mejorar la disponibilidad de
entornos "estaba en la parte superior de la lista. En retrospectiva, era obvio:
Los equipos de desarrollo necesitaban entornos aprovisionados para comenzar a trabajar,
y a menudo esperaban hasta ocho semanas.

Crearon una nueva integración y crearon un equipo responsable de
"Incorporando calidad en nuestros procesos, en lugar de tratar de inspeccionar la calidad después de
el hecho." Inicialmente estaba compuesto por administradores de bases de datos (DBA) y
especialistas en automatización encargados de automatizar la creación de su entorno
proceso. El equipo rápidamente hizo un descubrimiento sorprendente: solo el 50% de los
el código fuente en sus entornos de desarrollo y prueba coincidía con lo que era
corriendo en producción.

Campbell-Pretty observó: "De repente, entendimos por qué nos encontramos
muchos defectos cada vez que implementamos nuestro código en nuevos entornos. En cada
ambiente, seguimos avanzando, pero los cambios que hicimos no estaban siendo
volver a poner en control de versiones ".

El equipo realizó una ingeniería inversa de todos los cambios que se habían realizado para
los diferentes entornos y ponerlos a todos en control de versiones. Ellos también
automatizó su proceso de creación de entorno para que pudieran repetidamente y
girar correctamente los ambientes.

Campbell-Pretty describió los resultados y señaló que "el tiempo que tomó obtener un
El entorno correcto pasó de ocho semanas a un día. Este fue uno de los

ajustes clave que nos permitieron alcanzar nuestros objetivos con respecto a nuestro liderazgo tiempo, el costo de entrega y la cantidad de defectos escapados que lo convirtieron en producción."

La historia de Campbell-Pretty muestra la variedad de problemas que se pueden rastrear a ambientes construidos inconsistentemente y los cambios no son volver a poner sistemáticamente en el control de versiones.

Durante el resto de este capítulo, discutiremos cómo construir el mecanismos que nos permitirán crear entornos bajo demanda, expandir el uso del control de versiones para todos en el flujo de valor, hacer infraestructura más fácil de reconstruir que reparar, y garantizar que los desarrolladores ejecuten su código en entornos de producción a lo largo de cada etapa del desarrollo de software ciclo vital.

HABILITAR A LA DEMANDA CREACIÓN DE DISPOSITIVO, PRUEBA, Y ENTORNOS DE PRODUCCIÓN

Como se ve en el ejemplo del almacén de datos de la empresa anterior, uno de los principales causas contribuyentes de caótica, disruptiva y, a veces, incluso catastrófica lanzamientos de software, es la primera vez que vemos cómo funciona nuestra aplicación se comporta en un entorno de producción con carga y producción realistas conjuntos de datos es durante el lanzamiento.† En muchos casos, los equipos de desarrollo pueden tener entornos de prueba solicitados en las primeras etapas del proyecto.

Sin embargo, cuando se requieren largos plazos de entrega para que las Operaciones entreguen entornos de prueba, los equipos pueden no recibirlos lo suficientemente pronto como para realizar prueba adecuada Peor aún, los entornos de prueba a menudo están mal configurados o lo son diferente de nuestros entornos de producción que todavía terminamos con grandes problemas de producción a pesar de haber realizado pruebas previas a la implementación.

En este paso, queremos que los desarrolladores ejecuten entornos similares a la producción en sus Estaciones de trabajo propias, creadas bajo demanda y autoservicio. Al hacer esto, los desarrolladores pueden ejecutar y probar su código en entornos de producción como parte de su trabajo diario, proporcionando retroalimentación temprana y constante sobre la calidad su trabajo.

En lugar de simplemente documentar las especificaciones de la producción entorno en un documento o en una página wiki, creamos una compilación común mecanismo que crea todos nuestros entornos, como el desarrollo, prueba y producción. Al hacer esto, cualquiera puede obtener una producción entornos en minutos, sin abrir un ticket, y mucho menos tener que esperar semanas.

Hacer esto requiere definir y automatizar la creación de nuestro bien conocido entornos, que son estables, seguros y en un estado de riesgo reducido, encarnando el conocimiento colectivo de la organización. Todos nuestros requisitos están incrustados, no en documentos o como conocimiento en la cabeza de alguien, pero codificado en nuestro proceso de construcción de entorno automatizado.

En lugar de Operaciones que construyen y configuran manualmente el entorno, nosotros puede usar la automatización para cualquiera o todos los siguientes:

- Copiar un entorno virtualizado (por ejemplo, una imagen de VMware, ejecutar un Script vagabundo, iniciando un archivo de imagen de máquina de Amazon en EC2)
- Crear un proceso de creación de entorno automatizado que comience desde "Metal desnudo" (p. Ej., Instalación de PXE desde una imagen de referencia)

- Uso de herramientas de administración de configuración de "infraestructura como código" (por ejemplo, Marioneta, Chef, Ansible, Sal, CFEngine, etc.)
- Uso de herramientas de configuración del sistema operativo automatizado (p. Ej., Solaris Jumpstart, Red Hat Kickstart, Debian preestablecido)
- Ensamblar un entorno a partir de un conjunto de imágenes virtuales o contenedores (p. ej., Vagabundo, Docker)
- Hacer girar un nuevo entorno en una nube pública (por ejemplo, Amazon Web Servicios, Google App Engine, Microsoft Azure), nube privada u otro PaaS (plataforma como servicio, como OpenStack o Cloud Foundry, etc.).

Debido a que hemos definido cuidadosamente todos los aspectos del entorno antes de tiempo, no solo podemos crear nuevos entornos rápidamente, sino también asegurarnos de que Estos entornos serán estables, confiables, consistentes y seguros. Esta

beneficia a todos.

Las operaciones se benefician de esta capacidad, para crear nuevos entornos rápidamente, porque la automatización del proceso de creación del entorno impone consistencia y reduce el trabajo manual tedioso y propenso a errores. Además, desarrollo se beneficia al poder reproducir todas las partes necesarias de la producción entorno para construir, ejecutar y probar su código en sus estaciones de trabajo. Haciendo esto, permitimos que los desarrolladores encuentren y solucionen muchos problemas, incluso lo antes posible etapas del proyecto, a diferencia de durante las pruebas de integración o peor, en producción.

Al proporcionar a los desarrolladores un entorno que controlan completamente, les permitimos Reproduzca, diagnostique y corrija defectos rápidamente, aislado de forma segura de la producción servicios y otros recursos compartidos. También pueden experimentar con los cambios. a los entornos, así como al código de infraestructura que lo crea (por ejemplo, guiones de gestión de configuración), creando aún más el conocimiento compartido entre Desarrollo y Operaciones.§

CREA NUESTRO ÚNICO REPOSITORIO DE VERDAD PARA TODO EL SISTEMA

En el paso anterior, permitimos la creación a pedido del desarrollo, prueba y entornos de producción. Ahora debemos asegurarnos de que todas las partes de nuestro sistema de software.

Durante décadas, el uso integral del control de versiones se ha convertido cada vez más en un práctica obligatoria de desarrolladores individuales y equipos de desarrollo.¶ A

El sistema de control de versiones registra los cambios en los archivos o conjuntos de archivos almacenados en el sistema. Esto puede ser código fuente, activos u otros documentos que pueden ser parte de un proyecto de desarrollo de software. Hacemos cambios en grupos llamados confirmaciones o revisiones. Cada revisión, junto con metadatos como quién hizo el cambio y cuándo se almacena dentro del sistema de una forma u otra, permitiéndonos comprometer, comparar, fusionar y restaurar revisiones pasadas a objetos al repositorio. También minimiza los riesgos al establecer una forma de revertir objetos en producción a versiones anteriores. (En este libro, los siguientes términos se utilizará indistintamente: registrado en el control de versiones, comprometido en

control de versiones, confirmación de código, confirmación de cambio, confirmación).

Cuando los desarrolladores colocan todos sus archivos de origen de aplicaciones y configuraciones en control de versiones, se convierte en el único repositorio de verdad que contiene el estado previsto preciso del sistema. Sin embargo, porque entregar valor a la el cliente requiere tanto nuestro código como los entornos en los que se ejecuta, necesitamos nuestros entornos en control de versiones también. En otras palabras, el control de versiones es para todos en nuestro flujo de valor, incluidos QA, Operaciones, Infosec, así como desarrolladores Al poner todos los artefactos de producción en control de versiones, nuestro El repositorio de control de versiones nos permite reproducir de forma repetida y confiable componentes de nuestro sistema de software en funcionamiento; esto incluye nuestras aplicaciones y entorno de producción, así como toda nuestra preproducción ambientes.

Para garantizar que podamos restaurar el servicio de producción de forma repetida y previsible (e, idealmente, rápidamente) incluso cuando ocurren eventos catastróficos, debemos registrarnos los siguientes activos para nuestro repositorio de control de versiones compartido:

- Todo el código de aplicación y las dependencias (p. Ej., Bibliotecas, contenido estático, etc.)
- Cualquier script utilizado para crear esquemas de bases de datos, datos de referencia de aplicaciones, etc.
- Todas las herramientas y artefactos de creación de entornos descritos en el artículo anterior. paso (p. ej., imágenes de VMware o AMI, recetas de marionetas o chefs, etc.)
- Cualquier archivo utilizado para crear contenedores (por ejemplo, definición de Docker o Rocket o archivos de composición)
- Todas las pruebas automatizadas compatibles y cualquier script de prueba manual
- Cualquier script que admita el empaquetado de código, la implementación, la migración de la base de datos, y aprovisionamiento ambiental
- Todos los artefactos del proyecto (por ejemplo, documentación de requisitos, implementación procedimientos, notas de lanzamiento, etc.)

- Todos los archivos de configuración en la nube (por ejemplo, plantillas de AWS Cloudformation, Archivos Microsoft Azure Stack DSC, OpenStack HEAT)

Cualquier otra secuencia de comandos o información de configuración requerida para crear infraestructura que admite múltiples servicios (por ejemplo, servicio empresarial buses, sistemas de gestión de bases de datos, archivos de zona DNS, reglas de configuración para firewalls y otros dispositivos de red). ** **

Es posible que tengamos múltiples repositorios para diferentes tipos de objetos y servicios, donde están etiquetados y etiquetados junto con nuestro código fuente. Por ejemplo, podemos almacenar grandes imágenes de máquinas virtuales, archivos ISO, binarios compilados y etc. en repositorios de artefactos (por ejemplo, Nexus, Artifactory). Alternativamente, nosotros puede ponerlos en tiendas de blob (por ejemplo, cubos de Amazon S3) o poner imágenes de Docker en los registros de Docker, y así sucesivamente.

No es suficiente simplemente poder recrear cualquier estado anterior del entorno de producción; También debemos ser capaces de recrear todo el pre procesos de producción y construcción también. En consecuencia, tenemos que poner en control de versiones todo en lo que confían nuestros procesos de compilación, incluido nuestro herramientas (por ejemplo, compiladores, herramientas de prueba) y los entornos de los que dependen.††

En el *Informe de estado de DevOps 2014 de* Puppet Labs , el uso del control de versiones por Ops fue el mayor predictor tanto del rendimiento de TI como de la organización actuación. De hecho, si Ops usó el control de versiones fue un predictor más alto tanto para el desempeño de TI como para el desempeño organizacional que si Dev control de versiones usado.

Los hallazgos del *Informe de estado de DevOps* de Puppet Labs *2014* subrayan el El control crítico de la versión juega un papel en el proceso de desarrollo de software. Nosotros ahora sepa cuándo se registran todos los cambios en la aplicación y el entorno control de versiones, nos permite no solo ver rápidamente todos los cambios que podrían ha contribuido a un problema, pero también proporciona los medios para volver a un estado conocido anterior, en ejecución, lo que nos permite recuperarnos más rápidamente de fallas

Pero, ¿por qué el uso del control de versiones para nuestros entornos predice TI y ¿Es mejor el desempeño organizacional que usar el control de versiones para nuestro código?

Porque en casi todos los casos, hay órdenes de magnitud más configurables configuraciones en nuestro entorno que en nuestro código. En consecuencia, es el entorno que necesita estar más en control de versiones.‡‡

El control de versiones también proporciona un medio de comunicación para todos. trabajando en el flujo de valor: desarrollo, control de calidad, Infosec y Las operaciones capaces de ver los cambios de los demás ayudan a reducir sorpresas, crea

visibilidad en el trabajo del otro, y ayuda a construir y reforzar la confianza. Ver Apéndice 7 .

HAGA LA INFRAESTRUCTURA MÁS FÁCIL PARA RECONSTRUIR QUE REPARAR

Cuando podemos reconstruir y recrear rápidamente nuestras aplicaciones y entornos a pedido, también podemos reconstruirlos rápidamente en lugar de repararlos cuando las cosas van mal. Aunque esto es algo que casi toda la web a gran escala operaciones (es decir, más de mil servidores), también deberíamos adoptar esta práctica incluso si solo tenemos un servidor en producción.

Bill Baker, un distinguido ingeniero de Microsoft, bromeó diciendo que solíamos trata a los servidores como mascotas: "Los nombras y cuando se enferman, cuidas ellos de vuelta a la salud. [Ahora] los servidores son [tratados] como ganado. Los numeras y cuando se enferman, les disparas ".

Al tener sistemas de creación de entorno repetibles, somos capaces de aumentar la capacidad al agregar más servidores en rotación (es decir, horizontal escalada). También evitamos el desastre que inevitablemente resulta cuando debemos restablecer el servicio después de una falla catastrófica de infraestructura irreproducible, creado a través de años de cambios de producción indocumentados y manuales.

Para garantizar la coherencia de nuestros entornos, siempre que realizamos la producción. cambios (cambios de configuración, parches, actualizaciones, etc.), esos cambios necesita ser replicado en todas partes en nuestra producción y preproducción entornos, así como en cualquier entorno recién creado.

En lugar de iniciar sesión manualmente en los servidores y hacer cambios, debemos hacer cambios de una manera que garantiza que todos los cambios se replican en todas partes automáticamente y que todos nuestros cambios se pongan en control de versiones.

Podemos confiar en nuestros sistemas de configuración automatizados para garantizar la coherencia (p. ej., Puppet, Chef, Ansible, Salt, Bosh, etc.), o podemos crear nuevos virtuales máquinas o contenedores de nuestro mecanismo automatizado de construcción y despliegue en producción, destruyendo los viejos o sacándolos de rotación.§§

El último patrón es lo que se conoce como *infraestructura inmutable* , donde los cambios manuales en el entorno de producción ya no están permitidos: la única forma en que se pueden hacer cambios en la producción es poner los cambios en control de versiones y recrea el código y los entornos desde cero. Por Al hacer esto, ninguna variación puede arrastrarse a la producción.

Para evitar variaciones de configuración no controladas, podemos desactivar el control remoto inicia sesión en servidores de producción ¶¶ o rutinariamente mata y reemplaza producción instancias, asegurando que los cambios de producción aplicados manualmente se eliminen. Esta acción motiva a todos a poner sus cambios de la manera correcta a través del control de versiones. Al aplicar tales medidas, estamos sistemáticamente Reducir las formas en que nuestra infraestructura puede derivar de nuestros conocidos y buenos estados (por ejemplo, deriva de configuración, artefactos frágiles, obras de arte, copos de nieve, etc. adelante).

Además, debemos mantener actualizados nuestros entornos de preproducción, específicamente, necesitamos que los desarrolladores sigan funcionando en nuestro entorno más actual. Los desarrolladores a menudo querrán seguir ejecutándose en entornos más antiguos porque temen que las actualizaciones del entorno puedan romper la funcionalidad existente. Sin embargo, nos queremos actualizarlos con frecuencia para que podamos encontrar problemas en la primera parte de El ciclo de vida.***

MODIFICAR NUESTRA DEFINICIÓN DE DESARROLLO "HECHO" PARA INCLUIR EL FUNCIONAMIENTO EN LA PRODUCCIÓN ENTORNOS COMO

Ahora que nuestros entornos se pueden crear a pedido y todo está registrado en el control de versiones, nuestro objetivo es garantizar que estos entornos están siendo utilizados en el trabajo diario de desarrollo. Necesitamos verificar que nuestro la aplicación se ejecuta como se esperaba en un entorno similar a la producción mucho antes de final del proyecto o antes de nuestro primer despliegue de producción.

La mayoría de las metodologías modernas de desarrollo de software prescriben breves y intervalos de desarrollo iterativos, en oposición al enfoque del big bang (por ejemplo, el modelo de cascada). En general, cuanto más largo sea el intervalo entre la implementación, el peores los resultados. Por ejemplo, en la metodología Scrum, un *sprint* es un intervalo de desarrollo de tiempo (generalmente un mes o menos) dentro del cual estamos obligados a terminar, ampliamente definidos como cuando tenemos "trabajando y código potencialmente enviable ".

Nuestro objetivo es garantizar que el desarrollo y el control de calidad se integren habitualmente codifique con entornos de producción a intervalos cada vez más frecuentes

a lo largo del proyecto.ttt Hacemos esto ampliando la definición de "hecho" más allá de la funcionalidad correcta del código (adición en negrita): al final de

En cada intervalo de desarrollo, hemos integrado, probado, trabajando y

código potencialmente enviable, **demostrado en una producción**

medio ambiente .

En otras palabras, solo aceptaremos el trabajo de desarrollo como hecho cuando puede ser construido, implementado y confirmado con éxito que se ejecuta como se esperaba en un entorno de producción, en lugar de simplemente cuando un desarrollador lo cree para hacer, idealmente, se ejecuta bajo una carga de producción con una producción como conjunto de datos, mucho antes del final de un sprint. Esto evita situaciones donde un La función se llama hecho simplemente porque un desarrollador puede ejecutarla con éxito en su computadora portátil pero en ningún otro lugar.

Al hacer que los desarrolladores escriban, prueben y ejecuten su propio código en una producción entorno, la mayoría del trabajo para integrar con éxito nuestro código y entornos ocurren durante nuestro trabajo diario, en lugar de al final de la lanzamiento. Al final de nuestro primer intervalo, nuestra aplicación se puede demostrar para ejecutarse correctamente en un entorno de producción, con el código y El entorno se ha integrado muchas veces, idealmente con todos los pasos automatizados (no se requieren modificaciones manuales).

Mejor aún, al final del proyecto, habremos desplegado con éxito y ejecutar nuestro código en entornos de producción cientos o incluso miles de veces, dándonos la confianza de que la mayoría de nuestros problemas de implementación de producción han sido encontrados y reparados

Idealmente, utilizamos las mismas herramientas, como monitoreo, registro e implementación, en nuestros entornos de preproducción como lo hacemos en producción. Al hacer esto, nosotros tener familiaridad y experiencia que nos ayudará a implementar y ejecutar sin problemas, como así como diagnosticar y reparar nuestro servicio cuando está en producción.

Al permitir que Development and Operations obtenga un dominio compartido de cómo el código y el entorno interactúan, y practican implementaciones temprano y con frecuencia, reducimos significativamente los riesgos de implementación asociados con lanzamientos de código de producción. Esto también nos permite eliminar una clase completa de defectos operacionales y de seguridad y problemas arquitectónicos que generalmente son atrapado demasiado tarde en el proyecto para arreglarlo.

CONCLUSIÓN

El rápido flujo de trabajo desde el desarrollo hasta las operaciones requiere que cualquiera puede obtener entornos de producción bajo demanda. Al permitir que los desarrolladores usar entornos similares a la producción incluso en las primeras etapas de un software proyecto, reducimos significativamente el riesgo de problemas de producción más adelante. Esto es Una de las muchas prácticas que demuestran cómo las operaciones pueden hacer que los desarrolladores Mucho más productivo. Aplicamos la práctica de los desarrolladores que ejecutan su código

en entornos de producción al incorporarlo en la definición de "hecho."

Además, al poner todos los artefactos de producción en control de versiones, tenemos una "fuente única de verdad" que nos permite recrear toda la producción entorno de forma rápida, repetible y documentada, utilizando el mismo las prácticas de desarrollo para Operaciones funcionan como lo hacemos para el trabajo de Desarrollo. Y al hacer que la infraestructura de producción sea más fácil de reconstruir que de reparar, nosotros facilitar y agilizar la resolución de problemas, así como también facilitar ampliar la capacidad

Tener estas prácticas en su lugar establece el escenario para permitir una prueba integral automatización, que se explora en el próximo capítulo.

† En este contexto, el entorno se define como todo en la pila de aplicaciones, excepto la aplicación, incluidas las bases de datos, sistemas operativos, redes, virtualización y todas las configuraciones asociadas.

‡ La mayoría de los desarrolladores quieren probar su código, y a menudo han hecho todo lo posible para obtener entornos de prueba para hacerlo. Se sabe que los desarrolladores reutilizan entornos de prueba antiguos de proyectos anteriores (a menudo años) o preguntan a alguien que tiene un reputación de poder encontrar uno, simplemente no preguntarán de dónde vino, porque, invariablemente, alguien en algún lugar está ahora Falta un servidor.

§ Idealmente, deberíamos encontrar errores antes de las pruebas de integración cuando es demasiado tarde en el ciclo de pruebas para crear comentarios rápidos para desarrolladores Si no podemos hacerlo, es probable que tengamos un problema arquitectónico que debe abordarse. Diseñando nuestros sistemas para comprobabilidad, para incluir la capacidad de descubrir la mayoría de los defectos utilizando un entorno virtual no integrado en una estación de trabajo de desarrollo, es una parte clave de la creación de una arquitectura que admita flujo rápido y retroalimentación.

¶ El primer sistema de control de versiones probablemente estaba ACTUALIZADO en el CDC6600 (1969). Más tarde llegó SCCS (1972), CMS en VMS (1978), RCS (1982), y así sucesivamente.

** Se puede observar que el control de versiones cumple algunas de las construcciones ITIL de la Biblioteca de medios definitiva (DML) y la Configuración Base de datos de gestión (CMDB), haciendo un inventario de todo lo necesario para volver a crear el entorno de producción.

†† En futuros pasos, también realizaremos el check-in para controlar la versión de toda la infraestructura de soporte que construimos, como las suites de prueba automatizadas y nuestra infraestructura de tubería de integración y despliegue continuo.

‡‡ Cualquiera que haya realizado una migración de código para un sistema ERP (por ejemplo, SAP, Oracle Financials, etc.) puede reconocer la siguiente situación: Cuando falla la migración de un código, rara vez se debe a un error de codificación. En cambio, es mucho más probable que la migración haya fallado debido a algunos diferencia en los entornos, como entre Desarrollo y QA o QA y Producción.

§§ En Netflix, la edad promedio de la instancia de Netflix AWS es de veinticuatro días, con un 60% de menos de una semana de antigüedad.

¶¶ O permitirlo solo en emergencias, asegurando que una copia del registro de la consola se envíe automáticamente por correo electrónico al equipo de operaciones.

*** Toda la pila de aplicaciones y el entorno pueden agruparse en contenedores, lo que puede permitir una simplicidad y velocidad en toda la tubería de implementación.

††† El término *integración* tiene muchos usos ligeramente diferentes en Desarrollo y Operaciones. En desarrollo, la integración típicamente se refiere a la integración de código, que es la integración de múltiples ramas de código en el tronco en el control de versiones. En entrega continua y DevOps, las *pruebas de integración se* refieren a las pruebas de la aplicación en un entorno de producción o prueba integrada ambiente.

10 Habilitar rápido y Pruebas automatizadas confiables

En este punto, el desarrollo y el control de calidad están utilizando entornos similares a la producción en su diario trabajo, y estamos integrando y ejecutando con éxito nuestro código en una producción entorno para cada característica que se acepta, con todos los cambios registrados en la versión controlar. Sin embargo, es probable que obtengamos resultados no deseados si encontramos y corregimos errores en un fase de prueba separada, ejecutada por un departamento de control de calidad separado solo después de que todo el desarrollo haya ha sido completado Y, si las pruebas solo se realizan unas pocas veces al año, los desarrolladores aprenden sobre sus errores meses después de que introdujeron el cambio que causó el error. Por entonces, el vínculo entre causa y efecto probablemente se ha desvanecido, resolver el problema requiere lucha contra incendios y arqueología y, lo peor de todo, nuestra capacidad de aprender del error y integrarlo en nuestro trabajo futuro se ve significativamente disminuido.

Las pruebas automatizadas abordan otro problema significativo e inquietante. Gary Gruver observa que "sin pruebas automatizadas, cuanto más código escribimos, más tiempo y

se requiere dinero para probar nuestro código; en la mayoría de los casos, este es un negocio totalmente escalable modelo para cualquier organización tecnológica ".

Aunque Google ahora sin duda ejemplifica una cultura que valora las pruebas automatizadas en escala, este no siempre fue el caso. En 2005, cuando Mike Bland se unió a la organización, la implementación en Google.com a menudo era extremadamente problemática, especialmente para Google Web Equipo de servidor (GWS).

Como explica Bland: "El equipo de GWS se había puesto en una posición a mediados de la década de 2000 donde fue extremadamente difícil hacer cambios en el servidor web, una aplicación C ++ que manejaba todas las solicitudes a la página de inicio de Google y muchas otras páginas web de Google. Tan importante y prominente como lo fue Google.com, estar en el equipo de GWS no fue una tarea glamorosa: a menudo era el vertedero de todos los diferentes equipos que estaban creando varios funcionalidad de búsqueda, todos los cuales estaban desarrollando código independientemente uno del otro. Ellos tuvo problemas como que las compilaciones y las pruebas demoraron demasiado, el código se puso en producción sin ser probado, y los equipos controlan cambios grandes e infrecuentes que entran en conflicto con los de otros equipos ".

Las consecuencias de esto fueron grandes: los resultados de búsqueda podrían tener errores o convertirse en inaceptablemente lento, que afecta a miles de consultas de búsqueda en Google.com. El potencial El resultado no fue solo la pérdida de ingresos, sino también la confianza del cliente.

Bland describe cómo afectó a los desarrolladores que implementaron cambios: "El miedo se convirtió en la mente-asesino. El miedo evitó que los nuevos miembros del equipo cambiaran las cosas porque no lo hicieron Entiende el sistema. Pero el miedo también impidió que las personas experimentadas cambiaran las cosas. porque lo entendieron muy bien ".† Bland era parte del grupo que se determinó para resolver este problema.

El líder del equipo de GWS, Bharat Mediratta, creía que las pruebas automatizadas ayudarían. Como Bland describe: "Crearon una línea dura: no se aceptarían cambios en GWS sin pruebas automatizadas de acompañamiento. Establecieron una construcción continua y la mantuvieron religiosamente paso. Establecieron monitoreo de cobertura de prueba y se aseguraron de que su nivel de cobertura de prueba subió con el tiempo. Redactaron políticas y guías de prueba e insistieron en que los contribuyentes tanto dentro como fuera del equipo los siguen ".

Los resultados fueron sorprendentes. Como señala Bland, "GWS se convirtió rápidamente en uno de los más equipos productivos en la empresa, integrando grandes cantidades de cambios de diferentes equipos cada semana manteniendo un calendario de lanzamiento rápido. Los nuevos miembros del equipo fueron capaz de hacer contribuciones productivas a este complejo sistema rápidamente, gracias a una buena prueba cobertura y código de salud. Finalmente, su política radical permitió el inicio de Google.com página para expandir rápidamente sus capacidades y prosperar en un movimiento increíblemente rápido y panorama tecnológico competitivo ".

Pero GWS seguía siendo un equipo relativamente pequeño en una empresa grande y en crecimiento. El equipo quería expandir estas prácticas en toda la organización. Por lo tanto, el grupo de prueba nació, un grupo informal de ingenieros que querían elevar las pruebas automatizadas prácticas en toda la organización. Durante los siguientes cinco años, ayudaron a replicar esto cultura de pruebas automatizadas en todo Google. [1]

Ahora, cuando cualquier desarrollador de Google confirma el código, se ejecuta automáticamente en un conjunto de cientos de miles de pruebas automatizadas. Si el código pasa, se fusiona automáticamente en el maletero, listo para ser desplegado en producción. Muchas propiedades de Google se crean cada hora o diariamente, luego elija qué compilaciones lanzar; otros adoptan un continuo "Push on Green" Filosofía de entrega.

Las apuestas son más altas que nunca: un solo error de implementación de código en Google puede eliminar cada propiedad, todo al mismo tiempo (como un cambio de infraestructura global o cuando un el defecto se introduce en una biblioteca central de la que depende cada propiedad).

Eran Messeri, ingeniero del grupo de Infraestructura de desarrolladores de Google, señala: "Grande las fallas ocurren ocasionalmente. Recibirás un montón de mensajes instantáneos e ingenieros llamando en tu puerta [Cuando la tubería de implementación se rompe,] necesitamos arreglarlo de inmediato, porque los desarrolladores ya no pueden confirmar el código. En consecuencia, queremos que sea muy fácil. retroceder ".

Lo que permite que este sistema funcione en Google es la profesionalidad de la ingeniería y una gran confianza cultura que asume que todos quieren hacer un buen trabajo, así como la capacidad de detectar y Corrija los problemas rápidamente. Messeri explica: "No hay políticas estrictas en Google, como" Si interrumpe la producción de más de diez proyectos, tiene un SLA para solucionar el problema diez minutos.' En cambio, existe un respeto mutuo entre los equipos y un acuerdo implícito. que todo el mundo haga lo que sea necesario para mantener en funcionamiento la canalización de implementación. Todos sabemos que un día romperé tu proyecto por accidente; al día siguiente, puedes romper el mío ".

Lo que Mike Bland y el equipo de Testing Grouplet lograron ha hecho de Google uno de los organizaciones tecnológicas más productivas del mundo. Para 2013, pruebas automatizadas y La integración continua en Google permitió a más de cuatro mil pequeños equipos trabajar juntos y mantenerse productivo, todos desarrollando, integrando, probando e implementando simultáneamente su código en producción. Todo su código está en un único repositorio compartido, compuesto por

miles de millones de archivos, todos ellos continuamente construidos e integrados, con el 50% de su código siendo cambiado cada mes Algunas otras estadísticas impresionantes sobre su desempeño incluyen:

- 40,000 confirmaciones de código / día
- 50,000 construcciones / día (entre semana, esto puede exceder 90,000)
- 120,000 suites de pruebas automatizadas

75 millones de casos de prueba se ejecutan diariamente

Más de 100 ingenieros trabajando en ingeniería de prueba, integración continua y lanzamiento herramientas de ingeniería para aumentar la productividad del desarrollador (que representan el 0,5% de la I + D personal)

En el resto de este capítulo, veremos las prácticas de integración continua.
requerido para replicar estos resultados.

CONTINUAMENTE CONSTRUYA, PRUEBE E INTEGRE NUESTRO CÓDIGO Y ENTORNOS

Nuestro objetivo es incorporar calidad a nuestro producto, incluso en las primeras etapas, al tener Los desarrolladores crean pruebas automatizadas como parte de su trabajo diario. Esto crea una respuesta rápida bucle que ayuda a los desarrolladores a encontrar problemas temprano y solucionarlos rápidamente, cuando hay pocas limitaciones (p. ej., tiempo, recursos).

En este paso, creamos conjuntos de pruebas automatizadas que aumentan la frecuencia de integración y prueba de nuestro código y nuestros entornos de periódicos a continuos. Hacemos esto por construyendo nuestra tubería de implementación, que realizará la integración de nuestro código y entornos y desencadenan una serie de pruebas cada vez que se introduce un nuevo cambio en la versión controlar. § (Verfigura 13.)

La tubería de despliegue, definida por primera vez por Jez Humble y David Farley en su libro *Entrega continua: lanzamientos confiables de software a través de compilación, prueba e implementación* *La automatización* garantiza que todo el código registrado en el control de versiones se construya automáticamente y probado en un entorno similar a la producción. Al hacer esto, encontramos cualquier compilación, prueba o errores de integración tan pronto como se introduce un cambio, lo que nos permite corregirlos de inmediato. Hecho correctamente, esto nos permite estar siempre seguros de que estamos en una implementación Estado de envío.

Para lograr esto, debemos crear procesos automatizados de compilación y prueba que se ejecutan en ambientes. Esto es crítico por las siguientes razones:

Nuestro proceso de compilación y prueba puede ejecutarse todo el tiempo, independientemente de los hábitos de trabajo de ingenieros individuales

Un proceso de compilación y prueba segregado garantiza que comprendamos todas las dependencias requerido para construir, empaquetar, ejecutar y probar nuestro código (es decir, eliminar el "funcionó en el portátil de desarrollador, pero se rompió en la producción "problema").

Podemos empaquetar nuestra aplicación para permitir la instalación repetible de código y

configuraciones en un entorno (por ejemplo, en Linux RPM, yum, npm; en Windows, OneGet; Alternativamente, se pueden utilizar sistemas de embalaje específicos del marco, como EAR y archivos WAR para Java, gemas para Ruby, etc.).

En lugar de poner nuestro código en paquetes, podemos optar por empaquetar nuestras aplicaciones en contenedores desplegables (por ejemplo, Docker, Rkt, LXD, AMI).

Los entornos se pueden hacer más productivos de una manera consistente y repetible (por ejemplo, los compiladores se eliminan del entorno, los indicadores de depuración son apagado, etc.).

Nuestra tubería de implementación se valida después de cada cambio que nuestro código integra con éxito en un entorno de producción. Se convierte en la plataforma a través de la cual los probadores solicitan y certifica las compilaciones durante las pruebas de aceptación y las pruebas de usabilidad, y se ejecutará automáticamente validaciones de rendimiento y seguridad.

>

Figura 13: *La tubería de implementación (Fuente: Humble y Farley,* Entrega continua *, 3.)*

Además, se usará para compilaciones de autoservicio para UAT (prueba de aceptación del usuario), pruebas de integración y entornos de prueba de seguridad. En pasos futuros, a medida que evolucionamos canalización de implementación, también se utilizará para administrar todas las actividades necesarias para llevar nuestro cambios del control de versiones a la implementación.

Se ha diseñado una variedad de herramientas para proporcionar la funcionalidad de canalización de implementación, muchas de ellos de código abierto (por ejemplo, Jenkins, ThoughtWorks Go, Concourse, Bamboo, Microsoft Team Foundation Server, TeamCity, Gitlab CI, así como soluciones basadas en la nube como Travis CI y Snap). ¶

Comenzamos la canalización de implementación ejecutando la etapa de confirmación, que compila y empaqueta el software, ejecuta pruebas de unidad automatizadas y realiza validaciones adicionales, como estáticas análisis de código, duplicación y análisis de cobertura de prueba, y verificación de estilo. ** Si tiene éxito, esto desencadena la etapa de aceptación, que implementa automáticamente los paquetes creados en el compromete la etapa en un entorno similar a la producción y ejecuta las pruebas de aceptación automatizadas.

Una vez que se aceptan los cambios en el control de versiones, queremos empaquetar nuestro código solo una vez, así que que los mismos paquetes se utilizan para implementar código en toda nuestra línea de implementación. Al hacer esto, el código se implementará en nuestros entornos integrados de prueba y preparación en de la misma manera que se implementa en producción. Esto reduce las variaciones que pueden evitar errores posteriores que son difíciles de diagnosticar (por ejemplo, usar diferentes compiladores, compiladores banderas, versiones de biblioteca o configuraciones).††

El objetivo de la canalización de implementación es proporcionar a todos en la cadena de valor, especialmente desarrolladores, la retroalimentación más rápida posible de que un cambio nos ha sacado de un despliegue estado. Esto podría ser un cambio en nuestro código, en cualquiera de nuestros entornos, en nuestro pruebas, o incluso a la infraestructura de la tubería de implementación (por ejemplo, una configuración de Jenkins ajuste).

Como resultado, nuestra infraestructura de canalización de implementación se vuelve fundamental para nuestra procesos de desarrollo como nuestra infraestructura de control de versiones. Nuestra tubería de despliegue también almacena el historial de cada compilación de código, incluida la información sobre qué pruebas fueron realizado en qué compilación, qué compilaciones se han implementado en qué entorno, y cuáles fueron los resultados de la prueba. En combinación con la información en nuestro control de versiones historial, podemos determinar rápidamente qué causó la ruptura de nuestra tubería de implementación y, probablemente, cómo solucionar el error.

Esta información también nos ayuda a cumplir con los requisitos de evidencia para auditoría y cumplimiento propósitos, con evidencia que se genera automáticamente como parte del trabajo diario.

Ahora que tenemos una infraestructura de canalización de implementación en funcionamiento, debemos crear nuestra Prácticas de *integración continua* , que requieren tres capacidades:

Un conjunto completo y confiable de pruebas automatizadas que validan que estamos en una implementación estado.

Una cultura que "detiene toda la línea de producción" cuando nuestras pruebas de validación fallan.

Desarrolladores que trabajan en pequeños lotes en el tronco en lugar de ramas características de larga duración.

En la siguiente sección, describimos por qué se necesitan pruebas automatizadas rápidas y confiables y cómo para construirlo

CONSTRUIR UNA VALIDACIÓN AUTOMÁTICA RÁPIDA Y CONFIABLE BANCO DE PRUEBAS

En el paso anterior, comenzamos a crear la infraestructura de prueba automatizada que valida que tenemos una *compilación verde* (es decir, lo que sea que esté en el control de versiones está en un compilador y estado desplegable). Para subrayar por qué necesitamos realizar esta integración y pruebas paso continuo, considere lo que sucede cuando solo realizamos esta operación periódicamente, como durante un proceso de construcción nocturno.

Supongamos que tenemos un equipo de diez desarrolladores, y todos verifican su código en la versión control diario, y un desarrollador introduce un cambio que rompe nuestra compilación y prueba nocturna trabajo. En este escenario, cuando descubrimos al día siguiente que ya no tenemos una construcción verde, nuestro equipo de desarrollo tardará minutos, o más horas, en determinar qué El cambio causó el problema, quién lo introdujo y cómo solucionarlo.

Peor aún, suponga que el problema no fue causado por un cambio de código, sino que se debió a una prueba problema de entorno (p. ej., una configuración incorrecta en alguna parte). El desarrollo el equipo puede creer que resolvió el problema porque todas las pruebas unitarias pasan, solo para descubre que las pruebas seguirán fallando más tarde esa noche.

Para complicar aún más el problema, se habrán registrado diez cambios más en la versión control por el equipo ese día. Cada uno de estos cambios tiene el potencial de introducir más errores que podrían romper nuestras pruebas automatizadas, aumentando aún más la dificultad de tener éxito diagnosticando y solucionando el problema.

En resumen, la retroalimentación lenta y periódica mata. Especialmente para equipos de desarrollo más grandes. los problema se vuelve aún más desalentador cuando tenemos decenas, cientos o incluso miles de

otros desarrolladores verifican sus cambios en el control de versiones cada día. El resultado es que nuestras compilaciones y pruebas automatizadas se rompen con frecuencia, y los desarrolladores incluso dejan de verificar sus cambios en el control de versiones ("¿Por qué molestarse, ya que las compilaciones y las pruebas son siempre ¿roto?"). En cambio, esperan para integrar su código al final del proyecto, lo que resulta en todo los resultados no deseados del gran tamaño de lote, integraciones de big bang y producción despliegues‡‡

Para evitar este escenario, necesitamos pruebas automáticas rápidas que se ejecuten dentro de nuestra compilación y prueba entornos cada vez que se introduce un nuevo cambio en el control de versiones. De esta manera nosotros puede encontrar y solucionar cualquier problema de inmediato, como el ejemplo del servidor web de Google demostrado Al hacer esto, nos aseguramos de que nuestros lotes sigan siendo pequeños y, en cualquier momento dado a tiempo, permanecemos en un estado desplegable.

En general, las pruebas automatizadas se dividen en una de las siguientes categorías, de la más rápida a la más lenta:

Pruebas unitarias : por lo general, prueban un único método, clase o función de forma aislada, Brindar al desarrollador la seguridad de que su código funciona según lo diseñado. Para muchos razones, incluida la necesidad de mantener nuestras pruebas rápidas y sin estado, las pruebas unitarias suelen " fuera "de bases de datos y otras dependencias externas (por ejemplo, las funciones se modifican para devolver valores estáticos predefinidos, en lugar de llamar a la base de datos real).§§

Pruebas de aceptación : generalmente prueban la aplicación en su conjunto para garantizar que un mayor nivel de funcionalidad opera según lo diseñado (por ejemplo, la aceptación comercial criterios para una historia de usuario, la corrección de una API) y que los errores de regresión no introducido (es decir, rompimos la funcionalidad que anteriormente funcionaba correctamente). Humble y Farley definen la diferencia entre la unidad y las pruebas de aceptación como "El El objetivo de una prueba unitaria es mostrar que una sola parte de la aplicación hace lo que el programador tiene la intención de ... El objetivo de las pruebas de aceptación es demostrar que nuestro la aplicación hace lo que el cliente quiso, no es que funcione de la manera los programadores piensan que debería ". Después de que una compilación pasa nuestras pruebas unitarias, nuestro despliegue la tubería lo ejecuta contra nuestras pruebas de aceptación. Cualquier compilación que pase nuestras pruebas de aceptación normalmente se pone a disposición para pruebas manuales (por ejemplo, pruebas exploratorias, pruebas de IU, etc.), así como para pruebas de integración.

Pruebas de integración : las **pruebas de** integración son donde nos aseguramos de que nuestra aplicación sea correcta interactúa con otras aplicaciones y servicios de producción, en lugar de llamar a stubbed fuera de las interfaces. Como observan Humble y Farley, "gran parte del trabajo en el SIT El entorno implica la implementación de nuevas versiones de cada una de las aplicaciones hasta que todas cooperar. En esta situación, la prueba de humo suele ser un conjunto completo de aceptación pruebas que se ejecutan contra toda la aplicación ". Las pruebas de integración se realizan en compilaciones

que han pasado nuestras pruebas de unidad y aceptación. Porque las pruebas de integración son a menudo frágil, queremos minimizar el número de pruebas de integración y encontrar tantas de nuestras defectos como sea posible durante la unidad y las pruebas de aceptación. La capacidad de usar virtual o versiones simuladas de servicios remotos cuando la ejecución de pruebas de aceptación se convierte en un Requisito arquitectónico esencial.

Al enfrentar presiones de fecha límite, los desarrolladores pueden dejar de crear pruebas unitarias como parte de su trabajo diario, independientemente de cómo hayamos definido 'hecho'. Para detectar esto, podemos elegir medir y hacer visible nuestra cobertura de prueba (en función del número de clases, líneas de

código, permutaciones, etc.), incluso puede fallar nuestro conjunto de pruebas de validación cuando cae por debajo cierto nivel (p. ej., cuando menos del 80% de nuestras clases tienen pruebas unitarias).¶

Martin Fowler observa que, en general, "una construcción de diez minutos [y un proceso de prueba] es perfectamente dentro de lo razonable ... [Primero] hacemos la compilación y ejecutamos pruebas que son unidades más localizadas pruebas con la base de datos completamente apagada. Tales pruebas pueden ejecutarse muy rápido, manteniéndose dentro de La directriz de diez minutos. Sin embargo, cualquier error que implique interacciones a mayor escala, particularmente aquellos que involucran la base de datos real, no se encontrarán. La construcción de la segunda etapa se ejecuta un conjunto diferente de pruebas [pruebas de aceptación] que llegan a la base de datos real e involucran más comportamiento de extremo a extremo. Esta suite puede tardar un par de horas en ejecutarse ".

TENGA ERRORES TAN PRONTO EN NUESTRAS PRUEBAS AUTOMATIZADAS COMO POSIBLE

Un objetivo de diseño específico de nuestro conjunto de pruebas automatizadas es encontrar errores lo antes posible en las pruebas como posible. Es por eso que ejecutamos pruebas automatizadas de ejecución más rápida (p. Ej., Pruebas unitarias) antes que más lento. ejecutar pruebas automatizadas (por ejemplo, pruebas de aceptación e integración), que se ejecutan antes Cualquier prueba manual.

Otro corolario de este principio es que cualquier error debe encontrarse con el más rápido. categoría de prueba posible. Si la mayoría de nuestros errores se encuentran en nuestra aceptación y pruebas de integración, los comentarios que brindamos a los desarrolladores son mucho más lentos que con pruebas unitarias, y las pruebas de integración requieren el uso de pruebas de integración escasas y complejas entornos, que solo pueden ser utilizados por un equipo a la vez, lo que retrasa aún más los comentarios.

Además, no solo los errores detectados durante las pruebas de integración son difíciles y el tiempo consumir para los desarrolladores reproducir, incluso validar que se ha solucionado es difícil (es decir, un desarrollador crea una solución pero luego necesita esperar cuatro horas para saber si el las pruebas de integración ahora pasan).

Por lo tanto, siempre que encontremos un error con una prueba de aceptación o integración, debemos cree una prueba unitaria que pueda encontrar el error más rápido, antes y más barato. Martin Fowler describió la noción de la "pirámide de prueba ideal", donde podemos capturar la mayoría de nuestros errores al usar nuestras pruebas unitarias. (Verfigura 14.) En contraste, en muchos programas de prueba el lo inverso es cierto, donde la mayor parte de la inversión se realiza en pruebas manuales y de integración.

Figura 14: *Las pirámides de prueba automatizadas ideales y no ideales (Fuente: Martin Fowler, "TestPyramid")*

Si encontramos que la unidad o las pruebas de aceptación son demasiado difíciles y caras de escribir y mantener, es probable que tengamos una arquitectura que esté demasiado unida, donde sea fuerte La separación entre los límites de nuestro módulo ya no existe (o tal vez nunca existió). En En este caso, tendremos que crear un sistema más débilmente acoplado para que los módulos puedan ser probado de forma independiente sin entornos de integración. Conjuntos de pruebas de aceptación incluso para Son posibles las aplicaciones más complejas que se ejecutan en minutos.

ASEGURAR QUE LAS PRUEBAS SE EJECUTEN RÁPIDAMENTE (EN PARALELO, SI ES NECESARIO)

Debido a que queremos que nuestras pruebas se ejecuten rápidamente, necesitamos diseñar nuestras pruebas para que se ejecuten en paralelo, potencialmente en muchos servidores diferentes. También podemos querer ejecutar diferentes categorías de pruebas en paralelo. Por ejemplo, cuando una compilación pasa nuestras pruebas de aceptación, podemos ejecutar nuestro pruebas de rendimiento en paralelo con nuestras pruebas de seguridad, como se muestra en la figura 15. Podemos o es posible que no permita las pruebas exploratorias manuales hasta que la compilación haya pasado todas nuestras pruebas automatizadas —Que permite una retroalimentación más rápida, pero también puede permitir pruebas manuales en compilaciones que eventualmente fallará.

Hacemos que cualquier compilación que pase todas nuestras pruebas automatizadas esté disponible para su uso para exploración pruebas, así como para otras formas de pruebas manuales o intensivas en recursos (como pruebas de rendimiento). Queremos hacer todas estas pruebas con la mayor frecuencia posible y práctica, ya sea continuamente o en un horario.

Figura 15: *Ejecución de pruebas automáticas y manuales en paralelo*
(Fuente: Humble and Farley, Continuous Delivery *, edición Kindle, ubicación 3868.)*

Cualquier probador (que incluye a todos nuestros desarrolladores) debe usar la última compilación que haya pasado todas las pruebas automatizadas, en lugar de esperar a que los desarrolladores marquen una compilación específica como lista Probar. Al hacer esto, nos aseguramos de que las pruebas se realicen lo antes posible en el proceso.

ESCRIBE NUESTRAS PRUEBAS AUTOMATIZADAS ANTES DE ESCRIBIR EL CÓDIGO ("PRUEBA-DESARROLLO IMPULSADO ")

Una de las formas más efectivas de garantizar que tengamos pruebas automáticas confiables es escribir esas pruebas como parte de nuestro trabajo diario, utilizando técnicas como el *desarrollo basado en pruebas* (TDD) y el *desarrollo basado en pruebas de aceptación* (ATDD). Esto es cuando comenzamos cada cambiar al sistema escribiendo primero una prueba automatizada que valide el comportamiento esperado *falla* , y luego escribimos el código para que las pruebas pasen.

Esta técnica fue desarrollada por Kent Beck a fines de la década de 1990 como parte de Extreme Programación, y tiene los siguientes tres pasos:

1. Asegúrese de que las pruebas fallen. "Escriba una prueba para el próximo bit de funcionalidad que desea agregar". Registrarse.
2. Asegúrese de que las pruebas pasen. "Escriba el código funcional hasta que pase la prueba".
3. "Refactorice el código nuevo y el antiguo para hacerlo bien estructurado". Asegúrese de que las pruebas pasen. Regístrese nuevamente.

Estas suites de pruebas automatizadas se registran en el control de versiones junto con nuestro código, que proporciona una especificación actualizada y viva del sistema. Desarrolladores que deseen entender cómo usar el sistema puede ver este conjunto de pruebas para encontrar ejemplos de trabajo sobre cómo usar el API del sistema. ***

AUTOMATIZAR COMO MUCHAS DE NUESTRAS PRUEBAS MANUALES COMO POSIBLE

Nuestro objetivo es encontrar tantos errores de código a través de nuestras suites de pruebas automatizadas, reduciendo nuestro dependencia en pruebas manuales. En su presentación de 2013 en Flowcon titulada "On the Care and

Alimentación de los ciclos de retroalimentación ", observó Elisabeth Hendrickson," aunque las pruebas pueden ser automatizado, la creación de calidad no puede. Tener humanos ejecutando pruebas que deberían ser automatizado es una pérdida de potencial humano ".

Al hacer esto, permitimos que todos nuestros probadores (que, por supuesto, incluyen desarrolladores) trabajen en actividades de alto valor que no pueden automatizarse, como pruebas exploratorias o mejoras El proceso de prueba en sí.

Sin embargo, la simple automatización de todas nuestras pruebas manuales puede crear resultados no deseados. no quiere pruebas automatizadas que no sean confiables o generen falsos positivos (es decir, pruebas que debería haber pasado porque el código es funcionalmente correcto pero falló debido a problemas como como rendimiento lento, causando tiempos de espera, estado de inicio no controlado o estado no deseado debido al uso de stubs de bases de datos o entornos de prueba compartidos).

Las pruebas poco confiables que generan falsos positivos crean problemas importantes: desperdician tiempo valioso (por ejemplo, obligar a los desarrolladores a volver a ejecutar la prueba para determinar si hay en realidad un problema), aumentar el esfuerzo general de ejecutar e interpretar los resultados de nuestras pruebas, y a menudo resultan en desarrolladores estresados que ignoran los resultados de las pruebas por completo o apagan el pruebas automatizadas a favor de centrarse en la creación de código.

El resultado es siempre el mismo: detectamos los problemas más tarde, los problemas son más difíciles para arreglar, y nuestros clientes tienen peores resultados, lo que a su vez crea estrés en todo el flujo de valor.

Para mitigar esto, un pequeño número de pruebas confiables y automatizadas son casi siempre preferible a una gran cantidad de pruebas automáticas manuales o poco confiables. por lo tanto, nosotros enfóquese en automatizar solo las pruebas que realmente validan los objetivos comerciales que estamos intentando conseguir. Si abandonar una prueba da como resultado defectos de producción, debemos agregarla nuevamente a nuestro conjunto de pruebas manuales, con el ideal de eventualmente automatizarlo.

Como Gary Gruver, ex vicepresidente de ingeniería de calidad, ingeniería de lanzamiento y operaciones para Macys.com, describe observa, "Para un gran sitio de comercio electrónico minorista, pasamos de

ejecutar 1.300 pruebas manuales que realizamos cada diez días para ejecutar solo diez pruebas automatizadas en cada confirmación de código: es mucho mejor ejecutar algunas pruebas en las que confiamos que ejecutar pruebas que no son confiables Con el tiempo, crecimos este conjunto de pruebas para tener cientos de miles de pruebas automatizadas ".

En otras palabras, comenzamos con una pequeña cantidad de pruebas automáticas confiables y las agregamos con el tiempo, creando un nivel de seguridad cada vez mayor de que detectaremos rápidamente cualquier cambios en el sistema que nos sacan de un estado desplegable.

PRUEBAS INTEGRALES DE DESEMPEÑO EN NUESTRO SUITE DE PRUEBAS

Con demasiada frecuencia, descubrimos que nuestra aplicación funciona mal durante las pruebas de integración o después de que se haya implementado en producción. Los problemas de rendimiento son a menudo difíciles de resolver.

detectar, como cuando las cosas se ralentizan con el tiempo, pasan desapercibidas hasta que es demasiado tarde (por ejemplo, consultas de bases de datos sin un índice). Y muchos problemas son difíciles de resolver, especialmente cuando son causadas por decisiones arquitectónicas que tomamos o limitaciones imprevistas de nuestra redes, bases de datos, almacenamiento u otros sistemas.

Nuestro objetivo es escribir y ejecutar pruebas de rendimiento automatizadas que validen nuestro rendimiento en toda la pila de aplicaciones (código, base de datos, almacenamiento, red, virtualización, etc.) como parte de la tubería de implementación, por lo que detectamos problemas temprano, cuando las soluciones son más baratas Y más rápido.

Al comprender cómo se comportan nuestras aplicaciones y entornos bajo una producción carga, podemos hacer un trabajo mucho mejor en la planificación de la capacidad, así como detectar condiciones tales como:

> Cuando los tiempos de consulta de nuestra base de datos crecen de forma no lineal (por ejemplo, olvidamos activar la base de datos) indexación, y la carga de la página va de cien minutos a treinta segundos).
>
> Cuando un cambio de código provoca la cantidad de llamadas a la base de datos, uso de almacenamiento o tráfico de red para aumentar diez veces.

Cuando tenemos pruebas de aceptación que se pueden ejecutar en paralelo, podemos usarlas como base de nuestras pruebas de rendimiento. Por ejemplo, supongamos que ejecutamos un sitio de comercio electrónico y tenemos identificó "buscar" y "pagar" como dos operaciones de alto valor que deben funcionar bien bajo carga Para probar esto, podemos ejecutar miles de pruebas de aceptación de búsqueda paralelas simultáneamente con miles de pruebas de pago en paralelo.

Debido a la gran cantidad de cómputo y E / S que se requieren para ejecutar pruebas de rendimiento, crear un entorno de prueba de rendimiento puede ser fácilmente más complejo que crear entorno de producción para la aplicación en sí. Debido a esto, podemos construir nuestro entorno de pruebas de rendimiento al comienzo de cualquier proyecto y asegúrese de que nos dediquemos Cualesquiera recursos sean necesarios para construirlo temprano y correctamente.

Para encontrar problemas de rendimiento temprano, debemos registrar los resultados de rendimiento y evaluar cada rendimiento contra resultados anteriores. Por ejemplo, podríamos fallar las pruebas de rendimiento si el rendimiento se desvía más del 2% de la ejecución anterior.

INTEGRAR REQUISITOS NO FUNCIONALES PRUEBAS EN NUESTROS BANCO DE PRUEBAS

Además de probar que nuestro código funciona como está diseñado y funciona bajo cargas similares a la producción, también queremos validar cualquier otro atributo del sistema que nos importa acerca de. A menudo se denominan requisitos no funcionales, que incluyen disponibilidad, escalabilidad, capacidad, seguridad, etc.

Muchos de estos requisitos se cumplen mediante la configuración correcta de nuestros entornos, así que también debemos crear pruebas automatizadas para validar que nuestros entornos se han construido y configurado correctamente. Por ejemplo, queremos hacer cumplir la coherencia y la corrección de los siguientes, en los que se basan muchos requisitos no funcionales (por ejemplo, seguridad, rendimiento, disponibilidad):

- Aplicaciones de soporte, bases de datos, bibliotecas, etc.
- Intérpretes de idiomas, compiladores, etc.
- Sistemas operativos (por ejemplo, registro de auditoría habilitado, etc.)
- Todas las dependencias

Cuando usamos la infraestructura como herramientas de gestión de configuración de código (por ejemplo, Puppet, Chef, Ansible, Salt, Bosh), podemos usar los mismos marcos de prueba que usamos para probar nuestro código para también compruebe que nuestros entornos están configurados y funcionan correctamente (p. ej., codificación pruebas ambientales en pruebas de pepino o pepinillo).

Además, similar a cómo ejecutamos herramientas de análisis en nuestra aplicación en nuestro despliegue tubería (por ejemplo, análisis de código estático, análisis de cobertura de prueba), debemos ejecutar herramientas que analicen el código que construye nuestros entornos (p. ej., Foodcritic para Chef, puppet-lint para Marioneta). También deberíamos ejecutar cualquier control de seguridad como parte de nuestras pruebas automatizadas. para garantizar que todo esté configurado de forma segura y correcta (p. ej., especificaciones del servidor).

En cualquier momento, nuestras pruebas automatizadas pueden validar que tenemos una construcción ecológica y que Estamos en un estado desplegable. Ahora, debemos crear un cable Andon para que cuando alguien rompe la tubería de implementación, tomamos todos los pasos necesarios para volver a una construcción ecológica estado.

TIRE DE NUESTRO CORDÓN ANDON CUANDO EL DESPLIEGUE ESTANCIAS DE TUBERÍAS

Cuando tenemos una construcción ecológica en nuestra tubería de implementación, tenemos un alto grado de confianza en que nuestro código y entorno funcionarán según lo diseñado cuando implementemos nuestro cambios en producción.

Para mantener nuestro canal de implementación en un estado verde, crearemos un Andon virtual Cordón, similar al físico en el sistema de producción de Toyota. Cuando alguien introduce un cambio que hace que nuestras compilaciones o pruebas automatizadas fallen, no se permite ningún trabajo nuevo para ingresar al sistema hasta que se solucione el problema. Y si alguien necesita ayuda para resolver el problema, pueden traer cualquier ayuda que necesiten, como en el ejemplo de Google en el comienzo de este capítulo.

Cuando nuestra tubería de implementación se rompe, como mínimo, notificamos a todo el equipo de la falla, por lo que cualquiera puede solucionar el problema o revertir la confirmación. Podemos incluso configurar el sistema de control de versiones para evitar más confirmaciones de código hasta la primera etapa (es decir, compilaciones y pruebas unitarias) de la tubería de implementación vuelve a estar en estado verde. Si el el problema se debió a una prueba automatizada que generó un error falso positivo, la prueba ofensiva debe reescribirse o eliminarse.††† Todos los miembros del equipo deben estar capacitados para revertir el compromiso para volver a un estado verde.

Randy Shoup, ex director de ingeniería de Google App Engine, escribió sobre el importancia de devolver el despliegue a un estado verde. "Priorizamos el equipo objetivos sobre objetivos individuales: cada vez que ayudamos a alguien a avanzar en su trabajo, ayudamos todo el equipo Esto se aplica ya sea que ayudemos a alguien a arreglar la compilación o a un automatizado prueba, o incluso realizar una revisión de código para ellos. Y, por supuesto, sabemos que harán el lo mismo para nosotros, cuando necesitamos ayuda. Este sistema funcionó sin mucha formalidad o política: todos sabían que nuestro trabajo no era solo 'escribir código', sino también 'ejecutar un servicio'. Esto es por qué priorizamos todos los problemas de calidad, especialmente los relacionados con la confiabilidad y el escalado, en el nivel más alto, tratándolos como problemas de prioridad 0 'show-stopper'. De un sistema perspectiva, estas prácticas nos impiden deslizarnos hacia atrás ".

Cuando fallan las etapas posteriores de la tubería de implementación, como las pruebas de aceptación o el rendimiento pruebas, en lugar de detener todo el trabajo nuevo, tendremos desarrolladores y probadores de guardia que están responsable de solucionar estos problemas de inmediato. También deberían crear nuevas pruebas que ejecutar en una etapa anterior en la tubería de implementación para detectar cualquier regresión futura. por ejemplo, si descubrimos un defecto en nuestras pruebas de aceptación, deberíamos escribir una prueba unitaria para detectar el problema. Del mismo modo, si descubrimos un defecto en las pruebas exploratorias, deberíamos escribir una unidad o prueba de aceptación.

Para aumentar la visibilidad de las fallas de prueba automatizadas, debemos crear altamente visibles indicadores para que todo el equipo pueda ver cuándo fallan nuestras pruebas de compilación o automatizadas. Muchos equipos han creado luces de construcción altamente visibles que se montan en una pared, lo que indica el estado de compilación actual u otras formas divertidas de decirle al equipo que la compilación está rota, incluyendo lámparas de lava, reproducción de una muestra de voz o canción, klaxons, semáforos, etc.

En muchos sentidos, este paso es más desafiante que crear nuestras compilaciones y servidores de prueba: esas eran actividades puramente técnicas, mientras que este paso requiere cambiar el comportamiento humano e incentivos. Sin embargo, la integración continua y la entrega continua requieren estos cambios, como exploramos en la siguiente sección.

POR QUÉ NECESITAMOS EXTRAER EL CORDÓN ANDON

La consecuencia de no tirar del cable Andon y arreglar inmediatamente cualquier despliegue los problemas de la tubería resultan en un problema demasiado familiar donde se vuelve cada vez más difícil para devolver nuestras aplicaciones y entorno a un estado desplegable. Considera el siguiente situación:

> Alguien registra el código que rompe la compilación o nuestras pruebas automatizadas, pero nadie corrige eso.
>
> Alguien más registra otro cambio en la construcción rota, que tampoco pasa nuestras pruebas automatizadas, pero nadie ve los resultados fallidos de las pruebas que habrían permitido

para ver el nuevo defecto, y mucho menos arreglarlo.

Nuestras pruebas existentes no se ejecutan de manera confiable, por lo que es muy poco probable que creemos nuevas pruebas. (Por qué ¿molestia? Ni siquiera podemos ejecutar las pruebas actuales).

Cuando esto sucede, nuestras implementaciones en cualquier entorno se vuelven tan poco confiables como cuando no tenía pruebas automatizadas o estaba utilizando un método de cascada, donde la mayoría de nuestros Se están descubriendo problemas en la producción. El resultado inevitable de este círculo vicioso es que terminamos donde comenzamos, con una impredecible "fase de estabilización" que lleva semanas o meses en los que todo nuestro equipo está sumido en una crisis, tratando de hacer que todas nuestras pruebas aprobar, tomar atajos debido a las presiones de la fecha límite y aumentar nuestra deuda técnica. ‡‡‡

CONCLUSIÓN

En este capítulo, hemos creado un conjunto completo de pruebas automatizadas para confirmar que tener una construcción verde que todavía está en un estado pasable y desplegable. Hemos organizado nuestra prueba suites y actividades de prueba en una tubería de implementación. También hemos creado lo cultural norma de hacer lo que sea necesario para volver a un estado de construcción verde si alguien introduce un cambio que rompe cualquiera de nuestras pruebas automatizadas.

Al hacer esto, preparamos el escenario para implementar la integración continua, lo que permite muchos equipos pequeños para desarrollar, probar e implementar código de forma independiente y segura en producción, entregando valor a los clientes.

† Bland describió que en Google, una de las consecuencias de tener tantos desarrolladores talentosos fue que creó el "síndrome impostor", un término acuñado por psicólogos para describir informalmente a personas que no pueden internalizar sus logros. Wikipedia afirma que "a pesar de la evidencia externa de su competencia, quienes exhiben el síndrome siguen convencidos de que son fraudes y no merecen el éxito que han logrado. La prueba de éxito es descartados como suerte, oportunidad o como resultado de engañar a otros para que piensen que son más inteligentes y competentes de lo que creen ser ".

‡ Crearon programas de capacitación, publicaron el famoso boletín de *pruebas en el inodoro* (que publicaron en los baños), la hoja de ruta de Test Certified y programa de certificación, y llevó varios días de "arreglo" (es decir, bombardeos de mejora), que ayudaron a los equipos a mejorar sus procesos de prueba automatizados para que podría replicar los sorprendentes resultados que el equipo de GWS pudo lograr.

§ En Desarrollo, *la integración continua a* menudo se refiere a la integración continua de múltiples ramas de código en el tronco y asegurar que pase la unidad pruebas Sin embargo, en el contexto de la entrega continua y DevOps, la integración continua también exige la ejecución en entornos similares a la producción y pasando las pruebas de aceptación e integración. Jez Humble y David Farley desambiguan estos llamando al último CI +. En este libro, *integración continua* siempre se referirá a las prácticas de CI +.

¶ Si creamos contenedores en nuestra canalización de implementación y tenemos una arquitectura como microservicios, podemos permitir que cada desarrollador cree inmutables artefactos donde los desarrolladores ensamblan y ejecutan todos los componentes de servicio en un entorno idéntico a la producción en su estación de trabajo. Esto permite desarrolladores para construir y ejecutar más pruebas en su estación de trabajo en lugar de en servidores de prueba, dándonos comentarios aún más rápidos sobre su trabajo.

** Incluso podemos requerir que estas herramientas se ejecuten antes de que se acepten cambios en el control de versiones (por ejemplo, obtener ganchos de precompromiso). También podemos ejecutar estas herramientas dentro del *entorno de* desarrollo *integrado del desarrollador* (IDE; donde el desarrollador edita, compila y ejecuta código), lo que crea un código aún más rápido Bucle de retroalimentación.

†† También podemos usar contenedores, como Docker, como mecanismo de embalaje. Los contenedores permiten la capacidad de escribir una vez, ejecutar en cualquier lugar. Estos contenedores se crean como parte de nuestro proceso de compilación y se pueden implementar y ejecutar rápidamente en cualquier entorno. Porque el mismo contenedor se ejecuta en cada entorno, ayudamos a reforzar la consistencia de todos nuestros artefactos de construcción.

‡‡ Es exactamente este problema el que condujo al desarrollo de prácticas de integración continua.

§§ Existe una amplia categoría de técnicas arquitectónicas y de prueba utilizadas para manejar los problemas de las pruebas que requieren aportes de puntos de integración externos, incluidos "talones", "simulacros", "virtualización de servicios", etc. Esto se vuelve aún más importante para las pruebas de aceptación e integración, que colocan mucha más dependencia de estados externos.

¶¶ Deberíamos hacer esto solo cuando nuestros equipos ya valoran las pruebas automatizadas: este tipo de métrica es fácilmente elegible por desarrolladores y gerentes.

*** Nachi Nagappan, E. Michael Maximilien y Laurie Williams (de Microsoft Research, IBM Almaden Labs y North Carolina State University, respectivamente) realizó un estudio que mostró que los equipos que utilizan el código TDD produjeron un 60% -90% mejor en términos de densidad de defectos que los equipos que no son TDD, mientras que demorando solo 15% −35% más.

††† Si el proceso para deshacer el código no es bien conocido, una posible contramedida es programar un *par de reversiones programadas* , para que pueda ser mejor documentado

‡‡‡ Esto a veces se denomina *antipatrón de caída de Scrum de agua* , que se refiere a cuando una organización afirma estar utilizando prácticas similares a las de Agile, pero, en realidad, Todas las pruebas y la reparación de defectos se realizan al final del proyecto.

11 Habilitar y practicar Integración continua

En el capítulo anterior, creamos el automatizado
prácticas de prueba para garantizar que los desarrolladores se vuelvan rápidos
retroalimentación sobre la calidad de su trabajo. Esto se vuelve parejo
más importante a medida que aumentamos el número de desarrolladores
y la cantidad de ramas en las que trabajan en la versión
controlar.

La capacidad de "ramificarse" en los sistemas de control de versiones era
creado principalmente para permitir a los desarrolladores trabajar en
diferentes partes del sistema de software en paralelo, sin
el riesgo de que los desarrolladores individuales verifiquen cambios que
podría desestabilizar o introducir errores en el tronco
(a veces también llamado maestro o línea principal).†

Sin embargo, a los desarrolladores más largos se les permite trabajar en
sus ramas aisladas, cuanto más difícil se vuelve

para integrar y fusionar los cambios de todos en
maletero. De hecho, integrar esos cambios se convierte en
exponencialmente más difícil a medida que aumentamos el número
de ramas y la cantidad de cambios en cada código
rama.

Los problemas de integración resultan en una cantidad significativa de
volver a trabajar para volver a un estado desplegable, que incluye

cambios conflictivos que deben fusionarse manualmente o
fusiones que rompen nuestras pruebas automáticas o manuales,
por lo general requiere que varios desarrolladores tengan éxito
resolver. Y porque la integración ha sido tradicionalmente
hecho al final del proyecto, cuando lleva mucho más tiempo
luego planeado, a menudo nos vemos obligados a cortar esquinas para hacer
La fecha de lanzamiento.

Esto provoca otra espiral descendente: al fusionar
el código es doloroso, tendemos a hacerlo con menos frecuencia, haciendo futuro
se fusiona aún peor. La integración continua fue diseñada
para resolver este problema haciendo que la fusión en el tronco sea
parte del trabajo diario de todos.

La sorprendente variedad de problemas que continúan
la integración resuelve, así como las soluciones mismas,
se ejemplifican en la experiencia de Gary Gruver como el
director de ingeniería para el firmware LaserJet de HP
división, que construye el firmware que ejecuta todos sus
escáneres, impresoras y dispositivos multifunción.

El equipo estaba formado por cuatrocientos desarrolladores.
distribuido en los Estados Unidos, Brasil e India. A pesar de la
tamaño de su equipo, se movían demasiado lento. por
años, no pudieron ofrecer nuevas funciones tan rápido
como el negocio lo necesitaba.

Gruver describió el problema de esta manera: "El marketing sería
ven a nosotros con un millón de ideas para deslumbrar a nuestro cliente,
y simplemente les decíamos: 'Fuera de tu lista, elige los dos
cosas que le gustaría obtener en los próximos seis a doce meses ".

Solo estaban completando dos versiones de firmware por
año, con la mayor parte de su tiempo dedicado a portar código

para apoyar nuevos productos. Gruver estimó que solo el 5%
dedicaron su tiempo a crear nuevas funciones, el resto de
el tiempo dedicado al trabajo no productivo asociado
con su deuda técnica, como la gestión de múltiples
ramas de código y pruebas manuales, como se muestra a continuación:

> 20% en planificación detallada (su bajo rendimiento y
> los altos plazos de entrega se atribuyeron erróneamente a defectuosos
> estimación, y con la esperanza de obtener una mejor respuesta, ellos
> se les pidió que estimaran el trabajo con mayor detalle).
>
> 25% de código de portabilidad gastado, todo mantenido en separado
> ramas de código
>
> 10% gastado integrando su código entre desarrolladores

ramas

15% gastado completando pruebas manuales

Gruver y su equipo crearon el objetivo de aumentar el tiempo.
gastado en innovación y nueva funcionalidad por un factor de
diez. El equipo esperaba que este objetivo se pudiera lograr a través de:

Integración continua y desarrollo basado en troncales

Inversión significativa en automatización de pruebas

Creación de un simulador de hardware para que las pruebas puedan ser
correr en una plataforma virtual

La reproducción de fallas de prueba en el desarrollador
estaciones de trabajo

Página 171

Una nueva arquitectura para soportar la ejecución de todas las impresoras
una construcción y lanzamiento común

Antes de esto, cada línea de productos requeriría un nuevo código
rama, con cada modelo con una compilación de firmware única
con capacidades definidas en tiempo de compilación. ‡ El nuevo
arquitectura tendría todos los desarrolladores que trabajan en una
base de código común, con una única versión de firmware
compatible con todos los modelos LaserJet construidos fuera del maletero, con
capacidades de la impresora que se establecen en tiempo de ejecución en un

Archivo de configuración XML.

Cuatro años después, tenían una base de código compatible con todos veinticuatro líneas de productos HP LaserJet en desarrollo en el maletero Gruver admite desarrollo basado en troncales requiere un gran cambio de mentalidad. Los ingenieros pensaron que el tronco el desarrollo basado nunca funcionaría, pero una vez que comenzaron, no podían imaginar volver nunca más. Sobre el años hemos tenido varios ingenieros dejar HP, y ellos me llamaría para contarme qué tan atrasado desarrollo estaba en sus nuevas empresas, señalando cuán difícil es ser efectivo y lanzar un buen código cuando no hay retroalimentación que la integración continua les da.

Sin embargo, el desarrollo basado en troncales requería que construir pruebas automatizadas más efectivas. Gruver observó, "Sin pruebas automatizadas, la integración continua es la forma más rápida de obtener un montón de basura que nunca compila o se ejecuta correctamente ". Al principio, un completo El ciclo de prueba manual requirió seis semanas.

Para que todas las compilaciones de firmware se prueben automáticamente, invirtieron mucho en sus simuladores de impresoras y creó una granja de prueba en seis semanas, en pocos años dos mil simuladores de impresoras corrieron en seis bastidores de servidores que cargarían las compilaciones de firmware desde su tubería de despliegue. Su integración continua (CI)

el sistema ejecutó todo su conjunto de unidades automatizadas,
aceptación y pruebas de integración en compilaciones desde troncal,
tal como se describe en el capítulo anterior. Además,
crearon una cultura que detuvo todo el trabajo en cualquier momento
el desarrollador rompió la tubería de implementación, asegurando que
los desarrolladores rápidamente volvieron a poner el sistema en verde
estado.

Las pruebas automatizadas crearon comentarios rápidos que permitieron
desarrolladores para confirmar rápidamente que su código comprometido
En realidad funcionó. Las pruebas unitarias se ejecutarían en su
estaciones de trabajo en minutos, tres niveles de automatizado
las pruebas se ejecutarían en cada confirmación, así como cada dos
y cuatro horas La prueba de regresión completa final sería
corre cada veinticuatro horas. Durante este proceso, ellos:

Redujo la construcción a una construcción por día, eventualmente
haciendo de diez a quince construcciones por día

Pasé de alrededor de veinte confirmaciones por día
realizado por un "jefe de construcción" a más de cien
confirmaciones por día realizadas por desarrolladores individuales

Permitió a los desarrolladores cambiar o agregar líneas de 75k a 100k
de código cada día

Reducción de los tiempos de prueba de regresión de seis semanas a uno.

día

Este nivel de productividad nunca podría haber sido
compatible antes de adoptar la integración continua,
cuando simplemente crea una construcción verde requiere días de
heroica Los beneficios comerciales resultantes fueron
asombroso:

- Tiempo dedicado a impulsar la innovación y escribir nuevos
  las características aumentaron del 5% del tiempo de desarrollador al 40%.

- Los costos generales de desarrollo se redujeron en
  aproximadamente 40%.

- Los programas en desarrollo se incrementaron en
  alrededor del 140%.

- Los costos de desarrollo por programa disminuyeron en
  78%

Lo que muestra la experiencia de Gruver es que, después
uso integral del control de versiones, continuo
la integración es una de las prácticas más críticas que
permitir el flujo rápido de trabajo en nuestro flujo de valor, permitiendo
muchos equipos de desarrollo para desarrollar, probar,
y entregar valor. Sin embargo, integración continua
sigue siendo una práctica controvertida. El resto de esto
el capítulo describe las prácticas necesarias para implementar
integración continua, así como cómo superar
objeciones comunes

DESARROLLO DE LOTE PEQUEÑO Y QUÉ SUCEDE CUANDO NOS COMPROMETEMOS CÓDIGO PARA TRONCAR CON FRECUENCIA

Como se describe en los capítulos anteriores, cada vez que cambia se introducen en el control de versiones que causan nuestra la tubería de implementación fallará, rápidamente pululamos problema para solucionarlo, devolviendo nuestra tubería de implementación en un estado verde Sin embargo, se producen problemas importantes cuando los desarrolladores trabajan en sucursales privadas de larga duración (también conocido como "ramas de características"), solo fusionándose en el tronco esporádicamente, lo que resulta en un gran tamaño de lote de cambios Como se describe en el ejemplo de HP LaserJet, qué Los resultados son un caos significativo y un nuevo trabajo para obtener su código en un estado liberable.

Jeff Atwood, fundador del sitio Stack Overflow y autor del blog *Coding Horror* , observa que mientras hay muchas estrategias de ramificación, todas se pueden poner en el siguiente espectro:

> **Optimice la productividad individual:** cada persona soltera en el proyecto trabaja en su propio sucursal privada. Todos trabajan independientemente y nadie puede interrumpir el trabajo de nadie más; sin embargo, fusionarse se convierte en una pesadilla. Colaboración se vuelve casi cómicamente difícil: el de cada persona el trabajo tiene que fusionarse minuciosamente con todos trabajo de otra persona para ver incluso la parte más pequeña de la Sistema completo.

> **Optimice la productividad del equipo:** todos trabajan
> En la misma zona común. No hay ramas
> solo una línea recta de desarrollo larga e ininterrumpida.
> No hay nada que entender, por lo que los compromisos son
> simple, pero cada confirmación puede romper todo el proyecto
> y detener todo el progreso.

La observación de Atwood es absolutamente correcta.
precisamente, el esfuerzo requerido para fusionar con éxito
ramas de nuevo juntas aumenta exponencialmente como el
aumento de número de sucursales. El problema no es solo
en la reelaboración que crea este "infierno de fusión", pero también en el
comentarios retrasados que recibimos de nuestro despliegue
tubería. Por ejemplo, en lugar de pruebas de rendimiento
contra un sistema totalmente integrado
continuamente, es probable que suceda solo al final de nuestro
proceso.

Además, a medida que aumentamos la tasa de producción de código
A medida que agregamos más desarrolladores, aumentamos la probabilidad
que cualquier cambio impactará a alguien más y
aumentar el número de desarrolladores que se verán afectados
cuando alguien rompe la tubería de implementación.

Aquí hay un último efecto secundario preocupante del gran tamaño de lote
fusiones: cuando la fusión es difícil, nos volvemos menos capaces
y motivado para mejorar y refactorizar nuestro código, porque

Es más probable que las refactorizaciones causen reprocesos para todos
más. Cuando esto sucede, somos más reacios a
modificar el código que tiene dependencias en todo el
codebase, que es (trágicamente) donde podemos tener el
pagos más altos

Así es como Ward Cunningham, desarrollador del primer
wiki, describió por primera vez la deuda técnica: cuando no lo hacemos
refactorizar agresivamente nuestra base de código, se vuelve más
difícil de hacer cambios y mantener con el tiempo,
ralentizando la velocidad a la que podemos agregar nuevas funciones.
Resolver este problema fue una de las razones principales
detrás de la creación de integración continua y troncal
prácticas de desarrollo basadas, para optimizar para el equipo
productividad sobre productividad individual.

ADOPTAR BASADO EN TRONCO PRÁCTICAS DE DESARROLLO

Nuestra contramedida para las fusiones de lotes grandes es
instituir integración continua y basada en troncales
prácticas de desarrollo, donde todos los desarrolladores se registran
su código para troncalizar al menos una vez al día. Código de comprobación
en esto frecuentemente reduce nuestro tamaño de lote al trabajo
realizado por todo nuestro equipo de desarrolladores en un solo día.
Los desarrolladores más frecuentemente registran su código para
tronco, cuanto menor es el tamaño del lote y más cerca estamos de

El ideal teórico del flujo de una sola pieza.

El código frecuente se compromete a troncal significa que podemos ejecutar todos pruebas automatizadas en nuestro sistema de software en su conjunto y recibir alertas cuando un cambio rompe alguna otra parte de la aplicación o interfiere con el trabajo de otro desarrollador. Y porque podemos detectar problemas de fusión cuando son pequeños, podemos corregirlos más rápido.

Incluso podemos configurar nuestra tubería de implementación para rechazar cualquier confirmación (por ejemplo, cambios de código o entorno) que sacarnos de un estado desplegable. Este método se llama *commits cerrados* , donde la tubería de implementación primero confirma que el cambio enviado se realizará correctamente fusionar, construir como se esperaba y pasar todos los automatizados pruebas antes de fusionarse realmente en el tronco. Si no, el se notificará al desarrollador, permitiendo que las correcciones sean hecho sin afectar a nadie más en el flujo de valor.

La disciplina de los compromisos diarios de código también nos obliga a dividir nuestro trabajo en trozos más pequeños mientras todavía manteniendo el tronco en un estado funcional y liberable. Y versión el control se convierte en un mecanismo integral de cómo el equipo se comunican entre sí, todos tienen una mejor comprensión compartida del sistema, conoce el estado de la tubería de implementación, y pueden ayudarse mutuamente cuando se rompe Como resultado, logramos mayor calidad y

tiempos de entrega más rápidos.

Una vez implementadas estas prácticas, ahora podemos modificar nuevamente nuestra definición de "hecho" (adición en negrita): "En el Al final de cada intervalo de desarrollo, debemos tener integrado, probado, en funcionamiento y potencialmente enviable código, demostrado en un entorno de producción, **creado desde el tronco usando un proceso de un clic, y validado con pruebas automatizadas.** "

Adherirse a esta definición revisada de hecho nos ayuda Garantizar aún más la capacidad de prueba y la capacidad de implementación en curso del código que estamos produciendo. Al mantener nuestro código en un estado desplegable, somos capaces de eliminar lo común

práctica de tener una prueba separada y una fase de estabilización Al final del proyecto.

> Ernest Mueller, quien ayudó a diseñar DevOps transformación en National Instruments, luego ayudó transformar los procesos de desarrollo y lanzamiento en Bazaarvoice en 2012. Suministros de Bazaarvoice contenido generado por el cliente (p. ej., reseñas, calificaciones) para miles de minoristas, como Best Buy, Nike, y Walmart

En ese momento, Bazaarvoice tenía $ 120 millones en ingresos y se estaba preparando para una salida a bolsa. § El el negocio fue impulsado principalmente por el Bazaarvoice Aplicación de conversaciones, un Java monolítico aplicación compuesta por casi cinco millones de líneas de código que data de 2006, que abarca quince años mil archivos El servicio se ejecutó en 1.200 servidores. en cuatro centros de datos y múltiples servicios en la nube proveedores.

Parcialmente como resultado de cambiar a un Agile proceso de desarrollo y de dos semanas intervalos de desarrollo, hubo una tremenda deseo de aumentar la frecuencia de liberación de su calendario actual de lanzamiento de producción de diez semanas. También habían comenzado a desacoplar partes de sus aplicación monolítica, descomponiéndola en microservicios

Su primer intento de un calendario de lanzamiento de dos semanas fue en enero de 2012. Mueller observó: "No lo hizo ir bien. Causó un caos masivo, con cuarenta y cuatro Incidentes de producción presentados por nuestros clientes. los La reacción principal de la gerencia fue básicamente "No volvamos a hacer eso nunca más".

Mueller se hizo cargo de los procesos de lanzamiento en breve luego, con el objetivo de hacer lanzamientos quincenales

sin causar tiempo de inactividad del cliente. El negocio
objetivos para liberar más frecuentemente incluidos
permitiendo pruebas A / B más rápidas (descritas en el próximo
capítulos) y aumentar el flujo de características en
producción. Mueller identificó tres problemas centrales:

> La falta de automatización de prueba hizo que cualquier nivel de prueba
> durante los intervalos de dos semanas inadecuados para
> prevenir fallas a gran escala.
>
> La estrategia de ramificación de control de versiones permitida
> desarrolladores para registrar el nuevo código hasta el
> lanzamiento de la producción.
>
> Los equipos que ejecutan microservicios también fueron
> realizar lanzamientos independientes, que fueron
> a menudo causando problemas durante el lanzamiento del monolito
> o viceversa.

Mueller concluyó que el monolítico
Proceso de implementación de la aplicación de conversaciones
necesitaba ser estabilizado, lo que requería continuo
integración. En las seis semanas que siguieron,
los desarrolladores dejaron de hacer trabajos de características para enfocarse

en lugar de escribir suites de pruebas automatizadas,
incluyendo pruebas unitarias en JUnit, pruebas de regresión en
Selenio, y obtener una tubería de implementación
corriendo en TeamCity. "Al ejecutar estas pruebas todos los

tiempo, sentimos que podríamos hacer cambios con cierto nivel de seguridad. Y lo más importante, nosotros podría encontrar de inmediato cuando alguien rompió algo, en lugar de descubrirlo solo después de está en producción ".

También cambiaron a una versión de tronco / rama modelo, donde cada dos semanas creaban un nuevo rama de lanzamiento dedicada, sin nuevos compromisos permitido a esa rama a menos que hubiera una emergencia: todos los cambios se resolverían un proceso de cierre de sesión, ya sea por boleto o por equipo a través de su wiki interno. Esa rama iría a través de un proceso de control de calidad, que luego sería promovido a producción.

Las mejoras en la previsibilidad y la calidad de los lanzamientos fueron sorprendentes:

> Lanzamiento de enero de 2012: cuarenta y cuatro clientes incidentes (comienza el esfuerzo de integración continua)
>
> Versión del 6 de marzo de 2012: cinco días tarde, cinco incidentes de clientes
>
> Versión del 22 de marzo de 2012: a tiempo, un cliente incidente

> Versión del 5 de abril de 2012: a tiempo, cero clientes
> incidentes

Mueller describió además el éxito de este esfuerzo.
fue:

> Tuvimos tanto éxito con lanzamientos cada
> dos semanas, fuimos a lanzamientos semanales,
> que casi no requiere cambios de la
> equipos de ingenieria. Porque lanzamientos
> se volvió tan rutinario, era tan simple como
> duplicando el número de lanzamientos en el
> calendario y lanzamiento cuando el calendario
> nos dijo que lo hiciéramos. En serio, era casi un no
> evento. La mayoría de los cambios requeridos fueron
> en nuestros equipos de servicio al cliente y marketing,
> quien tuvo que cambiar sus procesos, como
> cambiando el horario de su semanal
> correos electrónicos de clientes para asegurarse de que los clientes
> Sabía que vendrían cambios de características.
> Después de eso, comenzamos a trabajar hacia nuestro próximo
> objetivos, que eventualmente llevaron a acelerar
> nuestros tiempos de prueba de más de tres horas a menos
> de una hora, reduciendo el número de
> entornos de cuatro a tres (Dev, Test,
> Producción, eliminación de etapas) y movimiento
> a un modelo completo de entrega continua donde
> habilite implementaciones rápidas con un solo clic.

CONCLUSIÓN

El desarrollo basado en troncales es probablemente el más
práctica controvertida discutida en este libro. Muchos
los ingenieros no creerán que es posible, incluso aquellos
que prefieren trabajar sin interrupciones en una sucursal privada
sin tener que lidiar con otros desarrolladores. Sin embargo,
los datos del *Informe de estado de DevOps 2015* de Puppet Labs
está claro: el desarrollo basado en troncales predice mayor
rendimiento y mejor estabilidad, y trabajo aún mayor
satisfacción y tasas más bajas de agotamiento.

Si bien convencer a los desarrolladores puede ser difícil al principio,
una vez que vean los beneficios extraordinarios, probablemente
convertirse en conversos de por vida, como el HP LaserJet y
Los ejemplos de Bazaarvoice ilustran. Integración continua
las prácticas preparan el escenario para el siguiente paso, que es
Automatizando el proceso de implementación y habilitando
liberaciones de riesgo.

† La ramificación en el control de versiones se ha usado de muchas maneras, pero generalmente se usa para dividir el trabajo entre los miembros del equipo por lanzamiento, promoción, tarea, componente, plataformas tecnológicas, etc. adelante.

‡ Se utilizaron indicadores de compilación (#define y #ifdef) para habilitar / deshabilitar la ejecución del código en presencia de copiadoras, tamaño de papel compatible, etc.

§ El lanzamiento de la producción se retrasó debido a su (exitosa) OPI.

12 Automatizar y habilitar Lanzamientos de bajo riesgo

Chuck Rossi es el director de ingeniería de lanzamiento en Facebook. Una de sus responsabilidades es supervisando el empuje diario del código. En 2012, Rossi describió su proceso de la siguiente manera: "Comenzando alrededor de la 1 p.m., cambio al 'modo de operaciones' y trabajo con mi equipo para prepararme para lanza los cambios que se enviarán a Facebook.com ese día. Esto es lo más estresante parte del trabajo y realmente depende en gran medida del juicio y la experiencia pasada de mi equipo. Nosotros trabaje para asegurarse de que todos los que tienen cambios están siendo contabilizados y estén activamente probando y apoyando sus cambios ".

Justo antes del impulso de producción, todos los desarrolladores que tengan cambios deben estar presentes. y verifique en su canal de chat IRC: cualquier desarrollador que no esté presente tendrá sus cambios eliminado automáticamente del paquete de implementación. Rossi continuó: "Si todo parece bueno y nuestros paneles de prueba y pruebas canarias† son verdes, presionamos el gran botón rojo y toda la flota de servidores de Facebook.com recibe el nuevo código. Dentro de veinte minutos, miles y miles de máquinas tienen un nuevo código sin impacto visible para el personas que usan el sitio ". ‡

Más tarde ese año, Rossi duplicó su frecuencia de lanzamiento de software a dos veces al día. Él explicó que el segundo impulso de código le dio a los ingenieros que no están en la costa oeste de los Estados Unidos la capacidad de "moverse y enviar tan rápido como cualquier otro ingeniero en la empresa ", y también les dio a todos un segundo oportunidad cada día para enviar código y lanzar funciones.

Figura 16: *Número de desarrolladores que implementan por semana en Facebook*
(Fuente: Chuck Rossi, "Enviar temprano y enviar dos veces más seguido")

Kent Beck, el creador de la metodología Extreme Programming, una de las principales
Los defensores de Test Driven Development, y el entrenador técnico en Facebook, además

comentarios sobre su estrategia de lanzamiento de código en un artículo publicado en su página de Facebook:
"Chuck Rossi hizo la observación de que parece haber un número fijo de cambios
Facebook puede manejar en una sola implementación. Si queremos más cambios, necesitamos más
despliegues Esto ha llevado a un aumento constante en el ritmo de implementación en los últimos cinco años,
desde implementaciones semanales a diarias hasta tres veces diarias de nuestro código PHP y de seis a cuatro para
ciclos de dos semanas para implementar nuestras aplicaciones móviles. Esta mejora ha sido impulsada
principalmente por el equipo de ingeniería de lanzamiento ".

Al usar la integración continua y hacer que la implementación del código sea un proceso de bajo riesgo, Facebook
ha permitido que el despliegue de código sea parte del trabajo diario de todos y sostenga al desarrollador
productividad. Esto requiere que la implementación del código sea automatizada, repetible y
previsible. En las prácticas descritas en el libro hasta ahora, a pesar de que nuestro código y
los entornos se han probado juntos, lo más probable es que no estemos implementando la producción
muy a menudo porque las implementaciones son manuales, lentas, dolorosas, tediosas y con errores.
propensos, y a menudo implican un traspaso inconveniente y poco confiable entre
Desarrollo y operaciones.

Y debido a que es doloroso, tendemos a hacerlo cada vez con menos frecuencia, lo que resulta en otra
reforzando la espiral descendente. Al diferir los despliegues de producción, acumulamos
diferencias más grandes entre el código que se implementará y lo que se está ejecutando en producción,
aumentando el tamaño del lote de implementación. A medida que aumenta el tamaño del lote de implementación, también lo hace el riesgo de
resultados inesperados asociados con el cambio, así como la dificultad para solucionarlos.

En este capítulo, reducimos la fricción asociada con los despliegues de producción, asegurando
que pueden realizarse con frecuencia y facilidad, ya sea por Operaciones o Desarrollo. Nosotros
haga esto ampliando nuestro canal de implementación.

En lugar de simplemente integrar continuamente nuestro código en un entorno similar a la producción, nosotros
permitirá la promoción a la producción de cualquier compilación que pase nuestra prueba automatizada y
proceso de validación, ya sea a pedido (es decir, con solo presionar un botón) o automáticamente (es decir,
cualquier compilación que pase todas las pruebas se implementa automáticamente).

Debido a la cantidad de prácticas presentadas, se proporcionan notas de pie de página extensas

numerosos ejemplos e información adicional, sin interrumpir la presentación de conceptos en el capitulo.

AUTOMATIZAR NUESTRO PROCESO DE DESPLIEGUE

Lograr resultados como los de Facebook requiere que tengamos un mecanismo automatizado que despliega nuestro código en producción. Especialmente si tenemos un proceso de implementación que tiene existió durante años, necesitamos documentar completamente los pasos en el proceso de implementación, como en un ejercicio de mapeo de flujo de valor, que podemos armar en un taller o documento incrementalmente (p. ej., en una wiki).

Una vez que tenemos el proceso documentado, nuestro objetivo es simplificar y automatizar la mayor cantidad de pasos manuales como sea posible, tales como:

- Empaquetar el código de manera adecuada para la implementación
- Crear imágenes o contenedores de máquinas virtuales preconfigurados

- Automatizar la implementación y configuración de middleware
- Copiar paquetes o archivos en servidores de producción
- Reiniciar servidores, aplicaciones o servicios
- Generando archivos de configuración a partir de plantillas
- Ejecución de pruebas de humo automáticas para asegurarse de que el sistema funciona correctamente configurado
- Ejecución de procedimientos de prueba
- Scripting y automatización de migraciones de bases de datos

Siempre que sea posible, rediseñaremos para eliminar los pasos, particularmente aquellos que llevan mucho tiempo hora de completar. También queremos no solo reducir nuestros plazos de entrega sino también la cantidad de transferencias tanto como sea posible para reducir errores y pérdida de conocimiento.

Hacer que los desarrolladores se centren en automatizar y optimizar el proceso de implementación puede llevar a mejoras significativas en el flujo de implementación, como garantizar que la aplicación pequeña los cambios de configuración ya no necesitan nuevas implementaciones o nuevos entornos.

Sin embargo, esto requiere que el Desarrollo trabaje estrechamente con las Operaciones para garantizar que todos Las herramientas y los procesos que creamos conjuntamente se pueden utilizar en sentido descendente, en lugar de alienar Operaciones o reinvención de la rueda.

Muchas herramientas que brindan integración y pruebas continuas también admiten la capacidad de ampliar la canalización de implementación para que las compilaciones validadas puedan promoverse a producción, típicamente después de que se realizan las pruebas de aceptación de producción (por ejemplo, Jenkins Build Pipeline plugin, ThoughtWorks Go.cd y Snap CI, Microsoft Visual Studio Team Services, y Concurso Pivotal).

Los requisitos para nuestra canalización de implementación incluyen:

Implementar de la misma manera en todos los entornos: al usar la misma implementación mecanismo para cada entorno (p. ej., desarrollo, prueba y producción), nuestro Es probable que las implementaciones de producción sean mucho más exitosas, ya que sabemos que tiene se ha realizado con éxito muchas veces ya anteriormente en la tubería.

Prueba de humo de nuestras implementaciones: durante el proceso de implementación, debemos probar que podemos conectar a cualquier sistema de soporte (por ejemplo, bases de datos, buses de mensajes, externos servicios) y ejecutar una transacción de prueba única a través del sistema para garantizar que nuestro sistema está funcionando según lo diseñado. Si alguna de estas pruebas falla, deberíamos fallar la implementación.

Asegúrese de mantener entornos consistentes: en los pasos anteriores, creamos un proceso de compilación de entorno de un solo paso para que el desarrollo, prueba y producción Los entornos tenían un mecanismo de construcción común. Debemos asegurarnos continuamente de que estos Los entornos permanecen sincronizados.

Por supuesto, cuando ocurre algún problema durante la implementación, tiramos del cable de Andon y pulula el problema hasta que se resuelva, tal como lo hacemos cuando nuestro despliegue

la tubería falla en cualquiera de los pasos anteriores.

CSG International dirige una de las mayores operaciones de impresión de facturas en los Estados Unidos. Scott Prugh, su arquitecto principal y vicepresidente de desarrollo, en un esfuerzo por mejorar el La previsibilidad y la confiabilidad de sus lanzamientos de software duplicaron su frecuencia de lanzamiento de dos por año a cuatro por año (reduciendo a la mitad su intervalo de implementación de veintiocho semanas a catorce semanas).

Aunque los equipos de desarrollo estaban utilizando la integración continua para desplegar sus código en entornos de prueba diariamente, los lanzamientos de producción fueron realizados por El equipo de operaciones. Prugh observó: “Era como si tuviéramos un 'equipo de práctica' que Practicado diariamente (o incluso con más frecuencia) en entornos de prueba de bajo riesgo, perfeccionando sus procesos y herramientas. Pero nuestro "equipo de juego" de producción tuvo muy pocos intentos de práctica, solo dos veces al año. Peor aún, estaban practicando en la producción de alto riesgo. entornos, que a menudo eran muy diferentes a los entornos de preproducción

con diferentes restricciones: a los entornos de desarrollo les faltaban muchos activos de producción como seguridad, cortafuegos, equilibradores de carga y una SAN ".

Para resolver este problema, crearon un Equipo de Operaciones Compartidas (SOT) que fue responsable de gestionar todos los entornos (desarrollo, prueba, producción) realizar implementaciones diarias en esos entornos de desarrollo y prueba, también como hacer implementaciones de producción y lanzamientos cada catorce semanas. Porque el SOT estaba haciendo implementaciones todos los días, cualquier problema que encontraron que era si no se fija, simplemente ocurriría nuevamente al día siguiente. Esto creó tremenda motivación para automatizar pasos manuales tediosos o propensos a errores y para solucionar cualquier problema que potencialmente podría suceder nuevamente. Debido a que los despliegues se realizaron casi cien veces antes del lanzamiento de producción, se encontraron la mayoría de los problemas y arreglado mucho antes de eso.

Hacer esto reveló problemas que anteriormente solo experimentaban las operaciones equipo, que entonces eran problemas para toda la cadena de valor para resolver. El diario las implementaciones permitieron comentarios diarios sobre qué prácticas funcionaron y cuáles no.

También se centraron en hacer que todos sus entornos se vean lo más similares posible, incluidos los derechos de acceso de seguridad restringidos y equilibradores de carga. Prugh escribe: "Nosotros Hicimos entornos de no producción lo más parecidos posible a la producción, y nosotros buscó emular las restricciones de producción de la mayor manera posible. Temprano la exposición a entornos de clase de producción alteró los diseños de la arquitectura para hacerlos más amigables en estos entornos restringidos o diferentes. Todos se ponen más inteligente desde este enfoque ".

Prugh también observa:

"Hemos experimentado muchos casos donde los cambios en los esquemas de la base de datos son 1) entregado a un equipo de DBA para que 'vayan y lo resuelvan' o 2) pruebas automatizadas que se ejecutan en conjuntos de datos poco realistas (es decir, "100 de MB frente a 100 de GB"), que condujo a fallas de producción. En nuestra antigua forma de trabajar, esto se convertiría en una noche culpa al juego entre equipos que intentan resolver el desastre. Creamos un desarrollo y el proceso de implementación que eliminó la necesidad de traspasos a DBA por cross-

Page 188

capacitar a los desarrolladores, automatizar los cambios de esquema y ejecutarlos diariamente. Nosotros creó pruebas de carga realistas contra datos de clientes desinfectados, ejecutando idealmente migraciones todos los días. Al hacer esto, ejecutamos nuestro servicio cientos de veces con escenarios realistas antes de ver el tráfico de producción real ". §

Sus resultados fueron asombrosos. Al hacer implementaciones diarias y duplicar el frecuencia de lanzamientos de producción, el número de incidentes de producción se redujo en 91%, MTTR bajó un 80% y el tiempo de implementación requerido para el el servicio para funcionar en producción en un "estado totalmente libre" pasó de catorce días a uno día.

Prugh informó que los despliegues se volvieron tan rutinarios que el equipo de Ops estaba jugando videojuegos al final del primer día. Además de las implementaciones que van más sin problemas para Dev y Ops, en el 50% de los casos el cliente recibió el valor en la mitad del tiempo, subrayando cómo las implementaciones más frecuentes pueden ser buenas para Desarrollo, QA, Operaciones y el cliente.

Figura 17: *Las implementaciones diarias y la frecuencia de lanzamiento cada vez mayor disminuyeron el número de incidentes de producción y MTTR (Fuente: "DOES15 - Scott Prugh y Erica Morrison - Conway & Taylor Meet the Strangler (v2.0) ", video de YouTube, 29:39, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015 https://www.youtube.com/watch?v=tKdIHCL0DUg.)*

HABILITAR DESPLIEGUE AUTOMATIZADO AUTOMATIZADO

Considere la siguiente cita de Tim Tischler, Director de Automatización de Operaciones en Nike, Inc., que describe la experiencia común de una generación de desarrolladores: "Como un desarrollador, nunca ha habido un punto más satisfactorio en mi carrera que cuando escribí el código, cuando presioné el botón para implementarlo, cuando pude ver las métricas de producción confirmar que realmente funcionó en producción, y cuando podría arreglarlo yo mismo si no funcionara ".

La capacidad de los desarrolladores de autodesplegar código en producción, para ver rápidamente clientes satisfechos cuando su función funciona y para solucionar rápidamente cualquier problema sin tener que abrir un ticket with Operations ha disminuido en la última década, en parte como resultado de la necesidad de control y supervisión, tal vez impulsado por los requisitos de seguridad y cumplimiento.

La práctica común resultante es que Operations realice implementaciones de código, porque La separación de funciones es una práctica ampliamente aceptada para reducir el riesgo de interrupciones de producción. y fraude Sin embargo, para lograr los resultados de DevOps, nuestro objetivo es cambiar nuestra dependencia a otros

mecanismos de control que pueden mitigar estos riesgos de manera igual o incluso más efectiva, como a través de pruebas automatizadas, implementación automatizada y revisión por pares de cambios.

El *Informe del Estado de DevOps 2013 de* Puppet Labs , que encuestó a más de cuatro mil

profesionales de la tecnología, encontraron que no había una diferencia estadísticamente significativa en el cambiar las tasas de éxito entre organizaciones donde el desarrollo implementó código y aquellas donde Operaciones desplegó el código.

En otras palabras, cuando hay objetivos compartidos que abarcan Desarrollo y Operaciones, y hay transparencia, responsabilidad y responsabilidad por los resultados de implementación, no importa quién realiza el despliegue. De hecho, incluso podemos tener otros roles, como como probadores o gerentes de proyecto, capaces de implementar en ciertos entornos para que puedan obtener su propio trabajo realizado rápidamente, como configurar demostraciones de características específicas en pruebas o Ambientes UAT.

Para habilitar mejor el flujo rápido, queremos un proceso de promoción de código que pueda realizar Desarrollo u Operaciones, idealmente sin pasos manuales o traspasos. Esta afecta los siguientes pasos:

Compilación: nuestra canalización de implementación debe crear paquetes a partir del control de versiones que puedan ser desplegado en cualquier entorno, incluida la producción.

Prueba: cualquier persona debería poder ejecutar cualquiera o todo nuestro conjunto de pruebas automatizadas en su estación de trabajo o en nuestros sistemas de prueba.

Implementación: cualquiera debería poder implementar estos paquetes en cualquier entorno donde tienen acceso, ejecutado ejecutando scripts que también se registran en el control de versiones.

Estas son las prácticas que permiten que las implementaciones se realicen con éxito, independientemente de quien realiza el despliegue.

INTEGRAR EL DESPLIEGUE DEL CÓDIGO EN LA TUBERÍA DE DESPLIEGUE

Una vez que el proceso de implementación del código está automatizado, podemos hacer que sea parte de la implementación tubería. En consecuencia, nuestra automatización de implementación debe proporcionar lo siguiente capacidades:

Asegúrese de que los paquetes creados durante el proceso de integración continua sean adecuados para despliegue en producción

Muestra la preparación de los entornos de producción de un vistazo

Proporcione un método de autoservicio con botón pulsador para cualquier versión adecuada del paquete código para ser implementado en producción

Registre automáticamente, para fines de auditoría y cumplimiento, qué comandos fueron ejecutar en qué máquinas cuándo, quién lo autorizó y cuál fue el resultado

Realice una prueba de humo para asegurarse de que el sistema funciona correctamente y la configuración la configuración, incluidos los elementos como las cadenas de conexión de la base de datos, es correcta

> Proporcione comentarios rápidos para el implementador para que puedan determinar rápidamente si la implementación fue exitosa (por ejemplo, si la implementación fue exitosa, es la aplicación funcionando como se espera en la producción, etc.)

Nuestro objetivo es garantizar que las implementaciones sean rápidas; no queremos tener que esperar horas para determinar si nuestro despliegue de código tuvo éxito o falló y luego necesitará horas para implementar cualquier corrección de código necesaria. Ahora que tenemos tecnologías como los contenedores, es posible complete incluso las implementaciones más complejas en segundos o minutos. En Puppet Labs ' *2014 Informe del estado de DevOps* , los datos mostraron que los de alto rendimiento tenían plazos de implementación medido en minutos u horas, mientras que los de menor rendimiento tuvieron tiempos de implementación medido en meses.

Figura 18: Los de *alto rendimiento tuvieron tiempos de entrega de implementación mucho más rápidos y un tiempo mucho más rápido para restaurar el servicio de producción después de los incidentes (Fuente: Puppet Labs,* Informe del estado de DevOps 2014 *).*

Al desarrollar esta capacidad, ahora tenemos un botón de "código de implementación" que nos permite promueva rápidamente cambios en nuestro código y nuestros entornos en producción a través de nuestro tubería de despliegue.

> A diferencia de Facebook, donde las implementaciones son administradas por ingenieros de lanzamiento, en Etsy las implementaciones son realizadas por cualquier persona que quiera realizar una implementación, como Desarrollo, Operaciones o Infosec. El proceso de implementación en Etsy se ha convertido tan seguro y rutinario que los nuevos ingenieros realizarán una implementación de producción en su primer día en el trabajo, ¡al igual que los miembros de la junta de Etsy e incluso perros!
>
> Como Noah Sussman, un arquitecto de pruebas en Etsy, escribió: “Para cuando las 8am se mueven en un día hábil normal, aproximadamente 15 personas y perros comienzan a hacer cola, todos ellos esperando desplegar colectivamente hasta 25 conjuntos de cambios antes de que termine el día ".

Los ingenieros que desean implementar su código primero van a una sala de chat, donde los ingenieros
agregarse a la cola de implementación, ver la actividad de implementación en progreso, ver
quién más está en la cola, difunde sus actividades y recibe ayuda de otros ingenieros
cuando lo necesitan Cuando es el turno de un ingeniero para implementar, se les notifica en el chat
habitación.

El objetivo en Etsy ha sido hacer que sea fácil y seguro implementarlo en producción con
menor número de pasos y la menor cantidad de ceremonia. Probablemente antes del
el desarrollador incluso registra el código, ejecutará en su estación de trabajo todas las 4.500 pruebas unitarias,
que lleva menos de un minuto. Todas las llamadas a sistemas externos, como bases de datos,
han sido apagados

Después de verificar sus cambios en el tronco en el control de versiones, más de siete mil
Las pruebas troncales automatizadas se ejecutan instantáneamente en sus servidores de integración continua (CI).
Sussman escribe: "A través de prueba y error, nos hemos conformado con unos 11 minutos como el
más tiempo que las pruebas automatizadas pueden ejecutarse durante un empuje. Eso deja tiempo para volver a correr
las pruebas una vez durante una implementación [si alguien rompe algo y necesita arreglarlo
], sin ir demasiado lejos del límite de tiempo de 20 minutos ".

Si todas las pruebas se ejecutaran secuencialmente, Sussman afirma que "las 7,000 pruebas troncales
demora aproximadamente media hora en ejecutarse. Entonces dividimos estas pruebas en subconjuntos, y
distribúyalos en las 10 máquinas de nuestro clúster Jenkins [CI] ....
conjunto de pruebas y ejecutar muchas pruebas en paralelo, nos da el tiempo de ejecución deseado de 11 minutos ".

Figura 19: *La consola Deployinator en Etsy (Fuente: Erik Kastner, "Quantum of Deployment" CodeasCraft.com, 20 de mayo de 2010, https://codeascraft.com/2010/05/20/quantum-of-despliegue / .)*

Las siguientes pruebas para ejecutar son las *pruebas de humo* , que son pruebas de nivel del sistema que ejecutan cURL

para ejecutar casos de prueba PHPUnit. Después de estas pruebas, se ejecutan las pruebas funcionales, que ejecutan pruebas de GUI de extremo a extremo en un servidor en vivo; este servidor es su Entorno de control de calidad o entorno de preparación (apodado "Princess"), que en realidad es un servidor de producción que se ha sacado de rotación, asegurando que coincida exactamente El entorno de producción.

Una vez que es el turno de despliegue de un ingeniero, Erik Kastner escribe: "vas a Deployinator [una herramienta desarrollada internamente, vea la figura 19] y presione el botón para obtener el control de calidad. Desde allí visita a Princess ... Luego, cuando está listo para salir en vivo, presionas el botón "Prod" y pronto su código estará en vivo, y todos en IRC [canal de chat] saben quién presionó qué código, completo con un enlace a la diferencia. Para cualquiera que no esté en IRC, está el correo electrónico que todos obtienen con la misma información ".

En 2009, el proceso de implementación en Etsy fue causa de estrés y miedo. Para 2011, se se había convertido en una operación de rutina, pasando de veinticinco a cincuenta veces por día, ayudando Los ingenieros obtienen su código rápidamente en la producción, entregando valor a sus clientes.

DESPLIEGUE DESPLIEGUE DE LIBERACIONES

En el lanzamiento tradicional de un proyecto de software, los lanzamientos son impulsados por nuestro lanzamiento de marketing fecha. La noche anterior, implementamos nuestro software completo (o tan cerca de completar como podríamos entrar) en producción. A la mañana siguiente, anunciamos nuestras nuevas capacidades a mundo, comience a tomar pedidos, entregue la nueva funcionalidad al cliente, etc.

Sin embargo, con demasiada frecuencia las cosas no salen según el plan. Podemos experimentar la producción cargas que nunca probamos o diseñamos, lo que hace que nuestro servicio falle espectacularmente, tanto para nuestros clientes y nuestra organización. Peor aún, restaurar el servicio puede requerir un doloroso proceso de reversión o una operación de *reparación* igualmente arriesgada , donde realizamos cambios directamente en producción, todo esto puede ser una experiencia verdaderamente miserable para los trabajadores. Cuando todo es finalmente trabajando, todos dan un suspiro de alivio, agradecidos de que los despliegues de producción y los lanzamientos no ocurren más a menudo.

Por supuesto, sabemos que debemos implementar más frecuentemente para lograr nuestro objetivo resultado del flujo suave y rápido, no con menos frecuencia. Para habilitar esto, necesitamos desacoplar nuestras implementaciones de producción a partir de nuestros lanzamientos de funciones. En la práctica, los términos *despliegue* y la *liberación a* menudo se usan indistintamente. Sin embargo, son dos acciones distintas que sirven dos propósitos muy diferentes:

La implementación es la instalación de una versión específica de software en un entorno determinado (por ejemplo, implementar código en un entorno de prueba de integración o implementar código en producción). Específicamente, un despliegue puede o no estar asociado con un lanzamiento de un

Característica a los clientes.

El lanzamiento es cuando ponemos una función (o conjunto de funciones) a disposición de todos nuestros clientes o un segmento de clientes (por ejemplo, permitimos que la función sea utilizada por el 5% de nuestros clientes base). Nuestro código y entornos deben estar diseñados de tal manera que el lanzamiento de funcionalidad no requiere cambiar nuestro código de aplicación.¶

En otras palabras, cuando combinamos la implementación y el lanzamiento, hace que sea difícil crear responsabilidad por resultados exitosos: desacoplar estas dos actividades nos permite empoderar a Desarrollo y Operaciones para ser responsables del éxito de implementaciones frecuentes, al tiempo que permite a los propietarios de productos ser responsables del éxito resultados comerciales del lanzamiento (es decir, estaba construyendo y lanzando la característica que vale nuestra hora).

Página 193

Las prácticas descritas hasta ahora en este libro aseguran que lo estamos haciendo rápido y frecuente implementaciones de producción a lo largo del desarrollo de características, con el objetivo de reducir riesgo e impacto de los errores de implementación. El riesgo restante es el riesgo de liberación, que es si Las características que ponemos en producción logran los resultados deseados para el cliente y el negocio.

Si tenemos plazos de implementación extremadamente largos, esto determina con qué frecuencia podemos lanzar nuevas funciones al mercado. Sin embargo, a medida que podamos implementar en demanda, qué tan rápido exponemos las nuevas funcionalidades a los clientes se convierte en un negocio y decisión de marketing, no una decisión técnica. Hay dos amplias categorías de lanzamiento patrones que podemos usar:

Patrones de lanzamiento basados en el entorno: aquí es donde tenemos dos o más entornos en los que implementamos, pero solo un entorno recibe clientes en vivo tráfico (por ejemplo, configurando nuestros equilibradores de carga). El nuevo código se implementa en un estado no activo entorno, y la versión se realiza moviendo el tráfico a este entorno. Estas son patrones extremadamente poderosos, porque generalmente requieren poco o ningún cambio en nuestro aplicaciones. Estos patrones incluyen *implementaciones azul-verde* , *lanzamientos canarios* y *sistemas inmunes en racimo* , todo lo cual se discutirá en breve.

Patrones de lanzamiento basados en aplicaciones: aquí es donde modificamos nuestra aplicación para que podemos liberar selectivamente y exponer funcionalidades de aplicaciones específicas por pequeños Cambios de configuración. Por ejemplo, podemos implementar indicadores de características que progresivamente exponer nuevas funcionalidades en producción al equipo de desarrollo, todo interno empleados, el 1% de nuestros clientes, o cuando estamos seguros de que la versión funcionará según lo diseñado, toda nuestra base de clientes. Como se discutió anteriormente, esto permite una técnica llamado lanzamiento oscuro, donde presentamos toda la funcionalidad que se lanzará en producción y pruébelo con el tráfico de producción antes de nuestro lanzamiento. Por ejemplo, podemos probar de forma invisible nuestra nueva funcionalidad con tráfico de producción durante semanas antes de nuestro lanzamiento para

exponer problemas para que puedan solucionarse antes de nuestro lanzamiento real.

PATRONES DE LIBERACIÓN BASADOS EN EL MEDIO AMBIENTE

El desacoplamiento de las implementaciones de nuestras versiones cambia drásticamente la forma en que trabajamos. Nosotros no ya tiene que realizar despliegues en medio de la noche o los fines de semana para bajar el riesgo de impactar negativamente a los clientes. En cambio, podemos hacer implementaciones durante el proceso típico horario comercial, lo que permite que Ops finalmente tenga un horario laboral normal, como todos los demás.

Esta sección se centra en los patrones de lanzamiento basados en el entorno, que no requieren cambios en código de aplicación Hacemos esto al tener múltiples entornos para implementar, pero solo uno de ellos recibe tráfico de clientes en vivo. Al hacer esto, podemos disminuir significativamente el riesgo asociado con lanzamientos de producción y reduce el tiempo de implementación.

El patrón de implementación azul-verde

El más simple de los tres patrones se llama despliegue azul-verde. En este patrón, tenemos Dos ambientes de producción: azul y verde. En cualquier momento, solo uno de estos está sirviendo tráfico de clientes: en la figura 20, el entorno verde está vivo.

Página 194

Figura 20: *Patrones de implementación azul-verde (Fuente: Humble and North,* Entrega continua *, 261.)*

Para lanzar una nueva versión de nuestro servicio, implementamos en el entorno inactivo donde puede realizar nuestras pruebas sin interrumpir la experiencia del usuario. Cuando tenemos confianza que todo funciona según lo diseñado, ejecutamos nuestro lanzamiento dirigiendo el tráfico al ambiente azul Por lo tanto, el azul se vuelve vivo y el verde se vuelve puesta en escena. Retroceder es se realiza enviando el tráfico de clientes de vuelta al entorno verde. ** **

El patrón de implementación azul-verde es simple y es extremadamente fácil de adaptar sistemas existentes También tiene beneficios increíbles, como permitir que el equipo realice implementaciones durante el horario comercial normal y realizar cambios simples (por ejemplo, cambiar la configuración de un enrutador, cambiar un enlace simbólico) durante las horas de menor actividad. Esto solo puede mejorar drásticamente las condiciones de trabajo para el equipo que realiza la implementación.

Manejo de cambios en la base de datos

Tener dos versiones de nuestra aplicación en producción crea problemas cuando dependen

en una base de datos común; cuando la implementación requiere cambios en el esquema de la base de datos o agregando, modificando o eliminando tablas o columnas, la base de datos no puede soportar ambos versiones de nuestra aplicación. Hay dos enfoques generales para resolver este problema:

Cree dos bases de datos (es decir, una base de datos azul y verde): cada versión: azul (antiguo) y verde (nuevo): la aplicación tiene su propia base de datos. Durante el lanzamiento, nosotros ponga la base de datos azul en modo de solo lectura, realice una copia de seguridad de la misma, restaure en el base de datos verde, y finalmente cambiar el tráfico al entorno verde. El problema con Este patrón es que si necesitamos volver a la versión azul, potencialmente podemos perder transacciones si no las migramos manualmente desde la versión verde primero.

Desacoplar los cambios de la base de datos de los cambios de la aplicación: en lugar de admitir dos bases de datos, desacoplamos el lanzamiento de los cambios de la base de datos del lanzamiento de la aplicación cambia haciendo dos cosas: Primero, solo hacemos cambios aditivos en nuestra base de datos, nunca mutamos los objetos existentes de la base de datos, y segundo, no hacemos supuestos en nuestra aplicación sobre qué versión de la base de datos estará en producción. Esta es muy diferente de cómo nos han entrenado tradicionalmente para pensar en bases de datos, donde evitamos duplicar datos. El proceso de desacoplamiento de la base de datos cambia de IMVU (entre otros) utilizó los cambios de la aplicación alrededor de 2009, lo que les permitió hacer cincuenta implementaciones por día, algunas de las cuales requirieron cambios en la base de datos.††

Dan North y Dave Farley, coautores de *Continuous Delivery* , estaban trabajando en un proyecto para Dixons Retail, un gran minorista británico que involucra miles de puntos de venta

Página 195

(POS) sistemas que residían en cientos de tiendas minoristas y operaban bajo un Número de diferentes marcas de clientes.

Aunque las implementaciones de color verde azulado se asocian principalmente con servicios web en línea, North y Farley usaron este patrón para reducir significativamente el riesgo y el cambio tiempos para actualizaciones de POS.

Tradicionalmente, la actualización de los sistemas de punto de venta es un gran proyecto de cascada: el punto de venta los clientes y el servidor centralizado se actualizan al mismo tiempo, lo que requiere tiempo de inactividad extenso (a menudo un fin de semana completo), así como una red significativa ancho de banda para lanzar el nuevo software de cliente a todas las tiendas minoristas. Cuando las cosas no vaya completamente de acuerdo al plan, puede ser increíblemente perjudicial para las operaciones de la tienda.

Para esta actualización, no había suficiente ancho de banda de red para actualizar todos los puntos de venta sistemas simultáneamente, lo que hizo imposible la estrategia tradicional. Para resolver esto

problema, utilizaron la estrategia azul-verde y crearon dos versiones de producción de el software del servidor centralizado, lo que les permite admitir simultáneamente el antiguo y Nuevas versiones de los clientes POS.

Después de hacer esto, semanas antes de la actualización de POS planificada, comenzaron a enviar nuevas versiones de instaladores de software POS para clientes en las tiendas minoristas a lo largo de la lenta enlaces de red, implementando el nuevo software en los sistemas POS en un estado inactivo. Mientras tanto, la versión anterior seguía funcionando normalmente.

Cuando todos los clientes POS tenían todo preparado para la actualización (el cliente actualizado y el servidor se habían probado juntos con éxito, y se había creado un nuevo software de cliente desplegado a todos los clientes), los gerentes de las tiendas estaban facultados para decidir cuándo lanza la nueva versión.

Dependiendo de sus necesidades comerciales, algunos gerentes querían usar las nuevas funciones inmediatamente y liberado de inmediato, mientras que otros querían esperar. En cualquier caso, ya sea lanzando funciones de inmediato o esperando, fue significativamente mejor para el gerentes que tener el departamento de TI centralizado elegir por ellos cuando el se produciría la liberación.

El resultado fue una liberación significativamente más suave y rápida, mayor satisfacción de los gerentes de la tienda, y mucho menos interrupción en las operaciones de la tienda. Además, esto la aplicación de implementaciones de color verde azulado a aplicaciones de PC de cliente grueso demuestra cómo los patrones DevOps se pueden aplicar universalmente a diferentes tecnologías, a menudo en formas muy sorprendentes pero con los mismos resultados fantásticos.

Los patrones de liberación del sistema inmunitario de Canarias y del grupo

El patrón de liberación azul-verde es fácil de implementar y puede aumentar drásticamente seguridad de las versiones de software. Hay variantes de este patrón que pueden mejorar aún más tiempos de entrega de seguridad y despliegue utilizando la automatización, pero con la posible compensación de Complejidad adicional.

El patrón de lanzamiento canario automatiza el proceso de lanzamiento de la promoción a sucesivamente entornos más grandes y críticos a medida que confirmamos que el código funciona según lo diseñado.

El término *liberación canaria* proviene de la tradición de los mineros de carbón que traen canarios enjaulados en minas para proporcionar detección temprana de niveles tóxicos de monóxido de carbono. Si hubiera demasiado

Página 196

mucho gas en la cueva, mataría a los canarios antes de matar a los mineros, alertándolos de evacuar.

En este patrón, cuando realizamos un lanzamiento, monitoreamos cómo funciona el software en cada El entorno está funcionando. Cuando algo parece estar yendo mal, retrocedemos; de lo contrario, implementamos en el siguiente entorno.‡‡

La Figura 21 muestra los grupos de entornos que Facebook creó para admitir esta versión

modelo:

Grupo A1: servidores de producción que solo sirven a empleados internos

Grupo A2: servidores de producción que solo atienden a un pequeño porcentaje de clientes y son desplegado cuando se cumplen ciertos criterios de aceptación (ya sea automático o manual)

Grupo A3: el resto de los servidores de producción, que se implementan después del software ejecutarse en el clúster A2 cumple con ciertos criterios de aceptación

Figura 21: *El patrón de liberación canaria (Fuente: Humble and Farley,* Continuous Delivery *, 263.)*

El sistema inmune del racimo se expande sobre el patrón de liberación canaria al vincular nuestro sistema de monitoreo de producción con nuestro proceso de lanzamiento y al automatizar la reversión de código cuando el rendimiento del sistema de producción de cara al usuario se desvía fuera de un rango esperado predefinido, como cuando las tasas de conversión para nuevos usuarios caen por debajo nuestras normas históricas del 15% al 20%.

Hay dos beneficios significativos para este tipo de salvaguardia. Primero, protegemos contra defectos que son difíciles de encontrar a través de pruebas automatizadas, como un cambio de página web que representa algún elemento crítico de la página invisible (por ejemplo, cambio de CSS). Segundo, reducimos el tiempo requerido para detectar y responder al rendimiento degradado creado por nuestro cambio. §§

PATRONES BASADOS EN LA APLICACIÓN PARA HABILITAR LANZAMIENTOS MÁS SEGUROS

En la sección anterior, creamos patrones basados en el entorno que nos permitieron desacoplar nuestras implementaciones de nuestras versiones mediante el uso de múltiples entornos y por cambiar entre qué entorno estaba vivo, que se puede implementar por completo en el nivel de infraestructura

En esta sección, describimos los patrones de lanzamiento basados en aplicaciones que podemos implementar en nuestro código, lo que permite una mayor flexibilidad en la forma en que lanzamos nuevas funciones de forma segura a nuestro

cliente, a menudo por función. Porque los patrones de lanzamiento basados en aplicaciones son implementado en la aplicación, estos requieren la participación del desarrollo.

Implementar alternar funciones

La forma principal en que habilitamos los patrones de lanzamiento basados en aplicaciones es implementando la función alterna, que nos proporciona el mecanismo para habilitar y deshabilitar selectivamente las funciones sin requerir una implementación de código de producción. Alternar funciones también puede controlar qué las funciones son visibles y están disponibles para segmentos de usuarios específicos (por ejemplo, empleados internos, segmentos de clientes).

Los conmutadores de funciones generalmente se implementan al ajustar la lógica de la aplicación o los elementos de la IU una declaración condicional, donde la característica está habilitada o deshabilitada en función de una configuración configuración almacenada en algún lugar. Esto puede ser tan simple como un archivo de configuración de la aplicación (por ejemplo, archivos de configuración en JSON, XML), o puede ser a través de un servicio de directorio o incluso una web servicio específicamente diseñado para administrar la alternancia de funciones.¶¶

Las funciones de alternar también nos permiten hacer lo siguiente:

Retroceda fácilmente: las características que crean problemas o interrupciones en la producción pueden ser deshabilitado de forma rápida y segura simplemente cambiando la configuración de alternancia de funciones. Esto es especialmente valioso cuando los despliegues son poco frecuentes, apagando uno en particular Las características de las partes interesadas suelen ser mucho más fáciles que revertir una versión completa.

Gravemente degradar el rendimiento: cuando nuestro servicio experimenta un nivel extremadamente alto cargas que normalmente requieren que aumentemos la capacidad o, lo que es peor, arriesguemos tener nuestro falla en la producción del servicio, podemos usar alternar funciones para reducir la calidad del servicio. En en otras palabras, podemos aumentar la cantidad de usuarios que atendemos reduciendo el nivel de funcionalidad entregada (por ejemplo, reducir el número de clientes que pueden acceder a un determinado , deshabilite las funciones intensivas de CPU, como recomendaciones, etc.).

Aumentar nuestra capacidad de recuperación a través de una arquitectura orientada a servicios: si tenemos un característica que depende de otro servicio que aún no está completo, aún podemos implementar nuestro característica en producción pero ocultarlo detrás de una función alternar. Cuando ese servicio finalmente está disponible, podemos activar la función. Del mismo modo, cuando un servicio confiamos en falla, podemos desactivar la función para evitar llamadas al servicio descendente mientras manteniendo el resto de la aplicación en ejecución.

Para garantizar que encontremos errores en las funciones envueltas en alternancia de funciones, nuestro automatizado Las pruebas de aceptación deben ejecutarse con todas las funciones activadas. (También debemos probar que nuestro ¡La funcionalidad de alternar funciones también funciona correctamente!)

Los conmutadores de funciones permiten el desacoplamiento de implementaciones de código y lanzamientos de funciones, más adelante en el libro que utilizamos alterna las funciones para permitir el desarrollo basado en hipótesis y las pruebas A / B, fomentando nuestra capacidad para lograr los resultados comerciales deseados.

Realizar lanzamientos oscuros

Los conmutadores de funciones nos permiten implementar funciones en producción sin hacerlas accesibles para los usuarios, lo que permite una técnica conocida como *lanzamiento oscuro* . Aquí es donde desplegamos todos los funcionalidad en producción y luego realizar pruebas de esa funcionalidad mientras todavía está

invisible para los clientes Para cambios grandes o riesgosos, a menudo hacemos esto durante semanas antes de la lanzamiento de producción, lo que nos permite probar de forma segura con las cargas de producción previstas.

Por ejemplo, supongamos que lanzamos una nueva característica que presenta un riesgo de liberación significativo, como como nuevas funciones de búsqueda, procesos de creación de cuentas o nuevas consultas de bases de datos. Después de todo el código está en producción, manteniendo la nueva función deshabilitada, podemos modificar el código de sesión del usuario para hacer llamadas a nuevas funciones: en lugar de mostrar los resultados al usuario, simplemente registramos o descartar los resultados.

Por ejemplo, es posible que el 1% de nuestros usuarios en línea realice llamadas invisibles a una nueva función programado para ser lanzado para ver cómo se comporta nuestra nueva función bajo carga. Después de que encontremos y solucionamos cualquier problema, aumentamos progresivamente la carga simulada aumentando la frecuencia y número de usuarios que ejercen la nueva funcionalidad. Al hacer esto, somos capaces de simule cargas similares a la producción, dándonos la confianza de que nuestro servicio funcionará necesita.

Además, cuando lanzamos una función, podemos implementarla progresivamente a pequeña segmentos de clientes, deteniendo el lanzamiento si se encuentran problemas. De esa manera, nosotros minimizar el número de clientes que reciben una función solo para que se la quiten porque encontramos un defecto o no podemos mantener el rendimiento requerido.

En 2009, cuando John Allspaw era vicepresidente de operaciones en Flickr, le escribió a Yahoo! equipo de gestión ejecutiva sobre su proceso de lanzamiento oscuro, diciendo que "aumenta *la* confianza *de todos* casi hasta el punto de la apatía, en lo que respecta al miedo a los problemas relacionados con la carga preocupado. No tengo idea de cuántas implementaciones de código se hicieron para la producción en cualquier día dado en los últimos 5 años ... porque en su mayor parte no me importa, porque esos cambios hecho en producción tiene una probabilidad tan baja de causar problemas. Cuando han causado problemas, todos los miembros del personal de Flickr pueden encontrar en una página web *cuando* se realizó el cambio, *quién* hizo el cambio, y exactamente (línea por línea) *cuál* fue el cambio ". ***

Más tarde, cuando hemos construido una telemetría de producción adecuada en nuestra aplicación y entornos, también podemos habilitar ciclos de retroalimentación más rápidos para validar nuestro negocio supuestos y resultados inmediatamente después de implementar la función en producción.

Al hacer esto, ya no esperamos hasta un lanzamiento de Big Bang para probar si los clientes quieren usa la funcionalidad que construimos. En cambio, cuando anunciamos y lanzamos nuestro gran característica, ya hemos probado nuestras hipótesis de negocio y hemos realizado innumerables experimentos para continuamente refinar nuestro producto con clientes reales, lo que nos ayuda a validar que el Las características lograrán los resultados deseados para el cliente.

> Durante casi una década, Facebook ha sido uno de los sitios de Internet más visitados sitios, según lo medido por las páginas vistas y los usuarios únicos del sitio. En 2008, había terminado setenta millones de usuarios activos diarios, lo que creó un desafío para el equipo que estaba desarrollando la nueva funcionalidad de Facebook Chat. †††
>
> Eugene Letuchy, un ingeniero en el equipo de Chat, escribió acerca de cómo el número de Los usuarios concurrentes presentaron un gran desafío de ingeniería de software: "La mayoría

La operación intensiva en recursos realizada en un sistema de chat no está enviando mensajes. Eso

es más bien mantener a cada usuario en línea al tanto de los estados en línea de inactividad
amigos, para que las conversaciones puedan comenzar ".

La implementación de esta característica computacionalmente intensiva fue una de las mayores técnicas
empresas en Facebook y tardó casi un año en completarse. Parte de la
La complejidad del proyecto se debió a la gran variedad de tecnologías necesarias para
lograr el rendimiento deseado, incluidos C ++, JavaScript y PHP, así como
su primer uso de Erlang en su infraestructura de back-end.

A lo largo de todo el año, el equipo de Chat verificó su código
en el control de versiones, donde se implementaría en producción al menos una vez por
día. Al principio, la funcionalidad de Chat solo era visible para el equipo de Chat. Más tarde fue
hecho visible para todos los empleados internos, pero estaba completamente oculto de los externos
Usuarios de Facebook a través de Gatekeeper, el servicio de alternancia de funciones de Facebook.

Como parte de su proceso de lanzamiento oscuro, cada sesión de usuario de Facebook, que se ejecuta
JavaScript en el navegador del usuario tenía un arnés de prueba cargado: la interfaz de usuario del chat
los elementos estaban ocultos, pero el cliente del navegador enviaría un chat de prueba invisible
mensajes al servicio de chat de back-end que ya estaba en producción, permitiéndoles
para simular cargas de producción en todo el proyecto, permitiéndoles encontrar
y solucione problemas de rendimiento mucho antes del lanzamiento del cliente.

Al hacer esto, cada usuario de Facebook era parte de un programa de prueba de carga masiva,
lo que permitió al equipo ganar confianza en que sus sistemas podrían manejar realistas
cargas similares a la producción. El lanzamiento y lanzamiento de Chat solo requirió dos pasos:
modificando la configuración de Gatekeeper para que la función de Chat sea visible para
una parte de los usuarios externos y hacer que los usuarios de Facebook carguen JavaScript nuevo
código que procesó la interfaz de usuario de Chat y deshabilitó el arnés de prueba invisible. Si algo
salió mal, los dos pasos se revertirían. Cuando el día del lanzamiento de Facebook
El chat llegó, fue sorprendentemente exitoso y sin incidentes, parecía escalar
sin esfuerzo de cero a setenta millones de usuarios durante la noche. Durante el lanzamiento, ellos
incrementalmente habilitó la funcionalidad de chat a segmentos cada vez más grandes del cliente
población: primero a todos los empleados internos de Facebook, luego al 1% del cliente
población, luego al 5%, y así sucesivamente. Como Letuchy escribió: "El secreto para ir de
de cero a setenta millones de usuarios de la noche a la mañana es evitar hacerlo todo de una sola vez ".

ENCUESTA DE ENTREGA CONTINUA Y CONTINUA IMPLEMENTACIÓN EN LA PRÁCTICA

En *Entrega continua,* Jez Humble y David Farley definen el término *continuo entrega.* El término *despliegue continuo* fue mencionado por primera vez por Tim Fitz en su publicación de blog "Implementación continua en IMVU: hacer lo imposible cincuenta veces al día". Sin embargo, en 2015, durante la construcción del *Manual de DevOps* , Jez Humble comentó: "En el En los últimos cinco años, ha habido confusión sobre los términos de entrega continua versus despliegue continuo y, de hecho, mi propio pensamiento y definiciones han cambiado desde Nosotros escribimos el libro. Cada organización debe crear sus variaciones, en función de lo que necesitar. Lo clave que debería importarnos no es la forma, sino los resultados: implementaciones deberían ser eventos de bajo riesgo y pulsadores que podemos realizar a pedido ".

Página 200

Sus definiciones actualizadas de entrega continua y despliegue continuo son las siguientes:

> Cuando todos los desarrolladores están trabajando en pequeños lotes en el tronco, o todos están trabajando tronco en ramas características de corta duración que se fusionan con el tronco regularmente, y cuando el tronco siempre se mantiene en un estado liberable, y cuando podemos liberarlo a pedido en el presionando un botón durante el horario comercial normal, estamos haciendo una entrega continua. Los desarrolladores obtienen comentarios rápidos cuando presentan errores de regresión, que incluyen defectos, problemas de rendimiento, problemas de seguridad, problemas de usabilidad, etc. Cuando estos problemas son encontrado, se reparan de inmediato para que el tronco siempre sea desplegable.
>
> Además de lo anterior, cuando estamos implementando buenas compilaciones en producción en un de forma regular a través del autoservicio (implementado por Dev o por Ops), que generalmente significa que estamos implementando en producción al menos una vez al día por desarrollador, o tal vez incluso implementando automáticamente cada cambio que un desarrollador comete, esto es cuando Estamos participando en un despliegue continuo.

Definido de esta manera, la entrega continua es el requisito previo para la implementación continua, solo Como la integración continua es un requisito previo para la entrega continua. Despliegue continuo es probable que se aplique en el contexto de los servicios web que se entregan en línea. Sin embargo, la entrega continua es aplicable en casi todos los contextos donde deseamos implementaciones y lanzamientos que tienen alta calidad, plazos de entrega rápidos y tienen un alto riesgo predecible y bajo resultados, incluidos los sistemas integrados, los productos COTS y las aplicaciones móviles.

En Amazon y Google, la mayoría de los equipos practican la entrega continua, aunque algunos realizan despliegue continuo, por lo tanto, existe una variación considerable entre los equipos en cuanto a cómo con frecuencia implementan código y cómo se realizan las implementaciones. Los equipos tienen poder elegir cómo implementar en función de los riesgos que gestionan. Por ejemplo, el Google El equipo de App Engine a menudo se implementa una vez al día, mientras que la propiedad de Búsqueda de Google se implementa varias veces por semana

Del mismo modo, la mayoría de los estudios de casos presentados en este libro también son entrega continua, como el software integrado que se ejecuta en impresoras HP LaserJet, la impresión de facturas CSG operaciones que se ejecutan en veinte plataformas tecnológicas, incluido un mainframe COBOL aplicación, Facebook y Etsy. Estos mismos patrones se pueden usar para el software que se ejecuta en

teléfonos móviles, estaciones de control en tierra que controlan satélites, etc.

CONCLUSIÓN

Como han demostrado los ejemplos de Facebook, Etsy y CSG, los lanzamientos y las implementaciones no tienen que ser asuntos de alto riesgo y gran drama que requieren decenas o cientos de ingenieros para trabajar todo el día para completar. En cambio, pueden hacerse completamente rutinarios y parte de El trabajo diario de todos.

Al hacer esto, podemos reducir nuestros plazos de implementación de meses a minutos, lo que permite nuestras organizaciones para entregar rápidamente valor a nuestros clientes sin causar caos y ruptura. Además, al hacer que Dev y Ops trabajen juntos, finalmente podemos hacer Las operaciones funcionan de manera humana.

† Una prueba de lanzamiento canario es cuando el software se implementa en un pequeño grupo de servidores de producción para asegurarse de que nada terrible les suceda con un cliente en vivo tráfico.

‡ La base de código front-end de Facebook está escrita principalmente en PHP. En 2010, para aumentar el rendimiento del sitio, el código PHP se convirtió en C ++ por su Compilador HipHop desarrollado internamente, que luego se compiló en un ejecutable de 1.5 GB. Este archivo se copió en servidores de producción usando

BitTorrent, que permite completar la operación de copia en quince minutos.

§ En sus experimentos, descubrieron que los equipos SOT tenían éxito, independientemente de si eran gestionados por Desarrollo u Operaciones, siempre que los equipos contaban con las personas adecuadas y estaban dedicados al éxito de SOT.

La Operación Escudo del Desierto puede servir como una metáfora efectiva. A partir del 7 de agosto de 1990, miles de hombres y materiales fueron desplegados de manera segura en más de cuatro meses en el teatro de producción, que culminó en un lanzamiento único, multidisciplinario y altamente coordinado.

** Otras formas en que podemos implementar el patrón azul-verde incluyen la configuración de múltiples servidores web Apache / NGINX para escuchar en diferentes entornos físicos o virtuales interfaces; empleando múltiples raíces virtuales en servidores Windows IIS vinculados a diferentes puertos; usando diferentes directorios para cada versión del sistema, con un enlace simbólico que determina cuál está vivo (por ejemplo, como lo hace Capistrano para Ruby on Rails); ejecutar múltiples versiones de servicios o middleware simultáneamente, con cada escucha en diferentes puertos; usando dos centros de datos diferentes y cambiando el tráfico entre los centros de datos, en lugar de usarlos simplemente como repuestos en caliente o en caliente para fines de recuperación ante desastres (incidentalmente, mediante el uso rutinario de ambos entornos, nos aseguramos continuamente de que nuestro el proceso de recuperación de desastres funciona según lo diseñado); o usando diferentes zonas de disponibilidad en la nube.

†† Este patrón también se conoce comúnmente como el patrón de expansión / contrato, que Timothy Fitz describió cuando dijo: "No cambiamos (mutamos) objetos de base de datos, como columnas o tablas. En cambio, primero nos expandimos, agregando nuevos objetos, luego, más tarde, contraemos eliminando los viejos ". Además, cada vez más, existen tecnologías que permiten la virtualización, el control de versiones, el etiquetado y la reversión de bases de datos, como Redgate, Delphix, DBMaestro y Datical, así como herramientas de código abierto, como DBDeploy, que hacen que los cambios en la base de datos sean mucho más seguros y rápidos.

‡‡ Tenga en cuenta que las versiones canarias requieren tener varias versiones de nuestro software ejecutándose en producción simultáneamente. Sin embargo, porque cada adicional la versión que tenemos en producción crea una complejidad adicional para administrar, debemos mantener el número de versiones al mínimo. Esto puede requerir el uso del patrón de base de datos expand / contract descrito anteriormente.

§§ El sistema inmunitario del grupo fue documentado por primera vez por Eric Ries mientras trabajaba en IMVU. Esta funcionalidad también es compatible con Etsy en su API de funciones biblioteca, así como por Netflix.

¶¶ Un ejemplo sofisticado de dicho servicio es el Gatekeeper de Facebook, un servicio desarrollado internamente que selecciona dinámicamente qué características son visibles a usuarios específicos basados en información demográfica como ubicación, tipo de navegador y datos de perfil de usuario (edad, género, etc.). Por ejemplo, un particular La función podría configurarse de modo que solo sea accesible para los empleados internos, el 10% de su base de usuarios o solo los usuarios entre las edades de veinticinco y treinta y cinco. Otros ejemplos incluyen la API de Etsy Feature y la biblioteca Netflix Archaius.

*** De manera similar, como describió Chuck Rossi, Director de Release Engineering en Facebook, "Todo el código que admite todas las funciones que planeamos lanzar los próximos seis meses ya se han implementado en nuestros servidores de producción. Todo lo que tenemos que hacer es encenderlo ".

††† En 2015, Facebook tenía más de mil millones de usuarios activos, con un crecimiento del 17% respecto al año anterior.

‡‡‡ Este problema tiene una característica computacional de peor caso de O (n3). En otras palabras, el tiempo de cálculo aumenta exponencialmente como la función de cantidad de usuarios en línea, el tamaño de sus listas de amigos y la frecuencia del cambio de estado en línea / fuera de línea.

13 Arquitecto de bajo riesgo Lanzamientos

Casi todos los conocidos ejemplares de DevOps han tenido casi muerte experiencias debido a problemas arquitectónicos, como en las historias presentado sobre LinkedIn, Google, eBay, Amazon y Etsy. En cada caso, pudieron migrar con éxito a un lugar más adecuado arquitectura que abordó sus problemas actuales y organizacionales necesidades.

Este es el principio de la arquitectura evolutiva: observa Jez Humble esa arquitectura de "cualquier producto u organización exitosa necesariamente evolucionar a lo largo de su ciclo de vida ". Antes de su mandato en Google, Randy Shoup se desempeñó como ingeniero jefe y arquitecto distinguido en eBay de 2004 a 2011. Observa que "eBay y Google están cada uno en su quinta reescritura completa de su arquitectura de arriba a abajo ".

Él reflexiona: "Mirando hacia atrás con 20/20 en retrospectiva, algo de tecnología [y opciones arquitectónicas] parecen proféticas y otras parecen miopes. Cada La decisión probablemente sirvió mejor a los objetivos de la organización en ese momento. Si habíamos intentado implementar el equivalente de micro servicios de 1995 la puerta, probablemente habríamos fallado, colapsando bajo nuestro propio peso y probablemente llevar a toda la compañía con nosotros ". †

El desafío es cómo seguir migrando desde la arquitectura que tenemos que La arquitectura que necesitamos. En el caso de eBay, cuando necesitaban volver a arquitecto, primero harían un pequeño proyecto piloto para probarse a sí mismos que entendieron el problema lo suficientemente bien como para incluso emprender el esfuerzo. Por ejemplo, cuando el equipo de Shoup planeaba mudar ciertos partes del sitio para Java de pila completa en 2006, buscaron el área eso les daría la mayor ganancia por su dinero al ordenar el sitio páginas por ingresos producidos. Eligieron las áreas de mayores ingresos, detenerse cuando no hubo suficiente retorno comercial para justificar la esfuerzo.

Lo que el equipo de Shoup hizo en eBay es un ejemplo de libro de texto de evolución diseñar, utilizando una técnica llamada la *aplicación estrangulador* patronaje en lugar de "eliminar y reemplazar" los servicios antiguos con arquitecturas que ya no es compatible con nuestros objetivos organizacionales, ponemos los existentes funcionalidad detrás de una API y evite realizar más cambios en ella. Todos Luego se implementa una nueva funcionalidad en los nuevos servicios que utilizan nueva arquitectura deseada, haciendo llamadas al viejo sistema cuando sea necesario.

El patrón de aplicación de estrangulador es especialmente útil para ayudar a migrar

porciones de una aplicación monolítica o servicios estrechamente acoplados a uno eso está más vagamente acoplado. Con demasiada frecuencia, nos encontramos trabajando dentro de una arquitectura que se ha acoplado demasiado y también interconectados, a menudo creados hace años (o décadas).

Las consecuencias de arquitecturas demasiado ajustadas son fáciles de detectar: cada vez que intentamos ingresar código en el tronco o liberar código en producción, corremos el riesgo de crear fallas globales (por ejemplo, rompemos a todos pruebas y funcionalidad de los demás, o todo el sitio se cae). Para evitar esto, cada pequeño cambio requiere enormes cantidades de comunicación y coordinación durante días o semanas, así como aprobaciones de cualquier grupo eso podría verse afectado potencialmente. Las implementaciones se vuelven problemáticas a medida que bueno, la cantidad de cambios que se agrupan para cada la implementación crece, lo que complica aún más la integración y el esfuerzo de prueba, y aumentando la ya alta probabilidad de que algo salga mal.

Incluso implementar pequeños cambios puede requerir la coordinación con cientos (o incluso miles) de otros desarrolladores, y cualquiera de ellos puede crear una falla catastrófica, que posiblemente requiera semanas para encontrar y solucionar el problema. (Esto da como resultado otro síntoma: "Mis desarrolladores gastan solo el 15% de su tiempo codifica; el resto de su tiempo lo dedica a reuniones ")

Todo esto contribuye a un sistema de trabajo extremadamente inseguro, donde los cambios tienen consecuencias aparentemente desconocidas y catastróficas. Eso También a menudo contribuye al temor de integrar e implementar nuestro código, y la espiral descendente autorreforzada de desplegarse con menos frecuencia.

Desde una perspectiva de arquitectura empresarial, esta espiral descendente es la consecuencia de la segunda ley de termodinámica arquitectónica, especialmente en organizaciones grandes y complejas. Charles Betz, autor de *Arquitectura y patrones para la gestión de servicios de TI, recursos*

204 de 1189.

Planificación y gobernanza: hacer zapatos para los niños del zapatero , observa, "[los propietarios de proyectos de TI] no son responsables de su contribuciones a la entropía general del sistema ". En otras palabras, reduciendo nuestra

complejidad general y aumento de la productividad de todos nuestros
Los equipos de desarrollo rara vez son el objetivo de un proyecto individual.

En este capítulo, describiremos los pasos que podemos seguir para revertir
espiral descendente, revise los principales arquetipos arquitectónicos, examine el
atributos de arquitecturas que permiten la productividad del desarrollador, la capacidad de prueba,
implementabilidad y seguridad, así como evaluar estrategias que nos permitan
migre de forma segura desde cualquier arquitectura actual que tengamos a una que
permite mejor el logro de nuestros objetivos organizacionales.

UNA ARQUITECTURA QUE PERMITE PRODUCTIVIDAD, TESTABILIDAD Y SEGURIDAD

En contraste con una arquitectura estrechamente acoplada que puede impedir que todos
productividad y capacidad para realizar cambios de forma segura, un acoplamiento flexible
arquitectura con interfaces bien definidas que imponen cómo los módulos
conectarse entre sí promueve la productividad y la seguridad. Permite
equipos pequeños, productivos, de dos pizzas que pueden hacer pequeños cambios
que se puede implementar de forma segura e independiente. Y porque cada servicio
también tiene una API bien definida, permite una prueba más fácil de los servicios y la
creación de contratos y SLA entre equipos.

Figura 22: *Almacén de datos de la nube de Google (Fuente: Shoup, "Del Monolito a Micro servicios ")*

Como describe Randy Shoup, "Este tipo de arquitectura ha servido a Google extremadamente bien: para un servicio como Gmail, hay otras cinco o seis capas de servicios debajo de él, cada uno muy enfocado en una función muy específica. Cada servicio es apoyado por un pequeño equipo, que lo construye y ejecuta su funcionalidad, con cada grupo potencialmente haciendo tecnología diferente opciones Otro ejemplo es el servicio Google Cloud Datastore, que es uno de los servicios NoSQL más grandes del mundo y, sin embargo, es compatible por un equipo de solo unas ocho personas, en gran parte porque se basa en capas sobre capas de servicios confiables construidos uno sobre el otro ".

Este tipo de arquitectura orientada al servicio permite que pequeños equipos trabajen en Unidades de desarrollo más pequeñas y simples que cada equipo puede implementar de forma independiente, rápida y segura. Shoup señala: "Organizaciones con Este tipo de arquitecturas, como Google y Amazon, muestran cómo puede afectar las estructuras organizacionales, [creando] flexibilidad y escalabilidad Ambas son organizaciones con decenas de miles de desarrolladores, donde los equipos pequeños aún pueden ser increíblemente productivos ".

ARQUITECTOS ARQUITECTONICOS: MONOLITOS VS. MICROSERVICIOS

En algún momento de su historia, la mayoría de las organizaciones de DevOps fueron obstaculizadas por arquitecturas monolíticas estrechamente acopladas que, aunque extremadamente exitosos en ayudarlos a lograr un producto / ajuste en el mercado, ponerlos en riesgo de fracaso organizacional una vez que tuvieron que operar a escala (por ejemplo, eBay Aplicación monolítica de C ++ en 2001, OBIDOS monolíticos de Amazon aplicación en 2001, el front-end monolítico de Rails de Twitter en 2009, y Aplicación monolítica Leo de LinkedIn en 2011). En cada uno de estos casos, pudieron rediseñar sus sistemas y preparar el escenario no solo para sobrevivir, pero también para prosperar y ganar en el mercado.

Las arquitecturas monolíticas no son inherentemente malas, de hecho, a menudo lo son La mejor opción para una organización al principio del ciclo de vida de un producto. Como Randy Shoup observa: “No existe una arquitectura perfecta para todos los productos. y todas las escalas. Cualquier arquitectura cumple con un conjunto particular de objetivos o rango de requisitos y limitaciones, como tiempo de comercialización, facilidad de desarrollo funcionalidad, escala, etc. La funcionalidad de cualquier producto o servicio es casi seguro que evolucione con el tiempo; no debería sorprendernos que nuestro las necesidades arquitectónicas también cambiarán. Lo que funciona a escala 1x rara vez funciona a escala 10x o 100x ".

Tabla 3: *arquetipos arquitectónicos*

(Fuente: Shoup, "Del monolito a los microservicios")

Los principales arquetipos arquitectónicos se muestran en la tabla 3, cada fila indica una necesidad evolutiva diferente de una organización, con cada

columna que proporciona los pros y los contras de cada uno de los diferentes arquetipos. Como la tabla muestra una arquitectura monolítica que admite un inicio (por ejemplo, creación rápida de prototipos de nuevas características y posibles pivotes o grandes cambios en estrategias) es muy diferente de una arquitectura que necesita cientos de equipos de desarrolladores, cada uno de los cuales debe poder entregar valor al cliente. Al apoyar arquitecturas evolutivas, podemos asegurarnos de que nuestra arquitectura siempre satisfaga las necesidades actuales de la organización.

Una de las transformaciones de arquitectura más estudiadas ocurrió en Amazonas. En una entrevista con el ganador del premio ACM Turing y Miembro técnico de Microsoft Jim Gray, CTO de Amazon Werner Vogels explica que Amazon.com comenzó en 1996 como un "monolítico aplicación, ejecutándose en un servidor web, hablando con una base de datos en el back end Esta aplicación, llamada Obidos, evolucionó para contener todos los lógica de negocios, toda la lógica de visualización y toda la funcionalidad que Amazon finalmente se hizo famoso por: similitudes, recomendaciones, Listmania, reseñas, etc. "

Con el paso del tiempo, Obidos se volvió demasiado enredado, con un intercambio complejo relaciones que significan que las piezas individuales no se pueden escalar como necesario. Vogels le dice a Gray que esto significa "muchas cosas que usted quisiera ver que suceda en un buen entorno de software no se pudo hacer más; había muchas piezas complejas de software combinado en un solo sistema. No pudo evolucionar nunca más."

Describiendo el proceso de pensamiento detrás del nuevo deseo arquitectura, le dice a Gray, "Pasamos por un período de graves introspección y concluyó que una arquitectura orientada a servicios nos daría el nivel de aislamiento que nos permitiría construir

muchos componentes de software de forma rápida e independiente ".

Vogels señala: "El gran cambio arquitectónico que tuvo Amazon en los últimos cinco años [desde 2001-2005] fue pasar de un monolito de dos niveles a servicios descentralizados totalmente distribuidos plataforma que sirve para muchas aplicaciones diferentes. Mucha innovación fue necesario para que esto sucediera, ya que fuimos uno de los primeros en adopta este enfoque ". Las lecciones de la experiencia de Vogel en Amazon que son importantes para nuestra comprensión de la arquitectura los turnos incluyen lo siguiente:

> **Lección 1:** Cuando se aplica rigurosamente, la orientación estricta al servicio es un excelente técnica para lograr el aislamiento; logras un nivel de propiedad y control que no se había visto antes.

Página 209

> **Lección 2:** Prohibir el acceso directo a la base de datos por parte de los clientes realizando mejoras de escala y confiabilidad a su servicio estado posible sin involucrar a sus clientes.

> **Lección 3: El** desarrollo y el proceso operativo benefician enormemente de cambiar a orientación de servicio. El modelo de servicios tiene ha sido un facilitador clave en la creación de equipos que pueden innovar rápidamente con un fuerte enfoque en el cliente. Cada servicio tiene un equipo asociado con él, y ese equipo es completamente responsable del servicio: desde el alcance de la funcionalidad hasta la arquitectura, construcción y operando

La medida en que la aplicación de estas lecciones mejora al desarrollador La productividad y la fiabilidad son impresionantes. En 2011, Amazon fue realizando aproximadamente quince mil implementaciones por día. Para 2015, realizaban casi 136,000 implementaciones por día.

UTILICE EL PATRÓN DE APLICACIÓN DE ESTRANGULADOR EVOLUCIONAR SEGURAMENTE NUESTRA EMPRESA ARQUITECTURA

El término *aplicación estrangulador* fue acuñado por Martin Fowler en 2004 después de que se inspiró al ver enormes enredaderas estranguladoras durante un viaje a Australia, escribiendo: "Sembran en las ramas superiores de una higuera y Gradualmente, descienda por el árbol hasta que arraigue en el suelo. Terminado muchos años crecen en formas fantásticas y hermosas, mientras tanto estrangulando y matando al árbol que era su anfitrión ".

Si hemos determinado que nuestra arquitectura actual es demasiado estricta, acoplados, podemos comenzar a desacoplar de forma segura partes de la funcionalidad de Nuestra arquitectura existente. Al hacer esto, habilitamos equipos que apoyan funcionalidad desacoplada para desarrollar, probar e implementar de forma independiente codifique en producción con autonomía y seguridad, y reduzca la arquitectura entropía

Como se describió anteriormente, el patrón de aplicación del estrangulador implica colocar funcionalidad existente detrás de una API, donde permanece sin cambios, y implementando una nueva funcionalidad usando nuestra arquitectura deseada, haciendo

Página 210

llama al sistema anterior cuando es necesario. Cuando implementamos estrangulador aplicaciones, buscamos acceder a todos los servicios a través de API versionadas, también llamados *servicios versionados* o *servicios inmutables* .

Las API versionadas nos permiten modificar el servicio sin afectar el llamadas, lo que permite que el sistema se acople más libremente, si es necesario Para modificar los argumentos, creamos una nueva versión de API y migramos equipos quienes dependen de nuestro servicio para la nueva versión. Después de todo, no somos lograr nuestros objetivos de reestructuración si permitimos que nuestro nuevo estrangulador aplicación para acoplarse estrechamente a otros servicios (por ejemplo, conectarse directamente a la base de datos de otro servicio).

Si los servicios que llamamos no tienen API bien definidas, deberíamos construir

ellos o al menos esconden la complejidad de comunicarse con tales sistemas dentro de una biblioteca de cliente que tiene una API limpiamente definida.

Al desacoplar repetidamente la funcionalidad de nuestro existente estrechamente acoplado sistema, trasladamos nuestro trabajo a un ecosistema seguro y vibrante donde los desarrolladores pueden ser mucho más productivos dando como resultado la aplicación heredada Encogimiento en la funcionalidad. Incluso podría desaparecer por completo como todos los La funcionalidad necesaria migra a nuestra nueva arquitectura.

Al crear aplicaciones estranguladoras, evitamos simplemente reproducir las existentes funcionalidad en alguna nueva arquitectura o tecnología, a menudo, nuestra Los procesos comerciales son mucho más complejos de lo necesario debido a la idiosincrasias de los sistemas existentes, que terminaremos replicando. (Al investigar al usuario, a menudo podemos rediseñar el proceso para que podamos puede diseñar un medio mucho más simple y simplificado para lograr el objetivo comercial.) ‡

Una observación de Martin Fowler subraya este riesgo: "Gran parte de mi La carrera ha implicado reescrituras de sistemas críticos. Pensarías que tal Lo fácil es hacer que el nuevo haga lo que hizo el anterior. Sin embargo ellos siempre son mucho más complejos de lo que parecen y desbordan riesgo. Se acerca la gran fecha límite, y la presión continúa. Mientras nuevo Me gustan las características (siempre hay nuevas características), las cosas viejas tienen que permanecer. Incluso los errores antiguos a menudo deben agregarse al sistema reescrito ”.

Como con cualquier transformación, buscamos crear victorias rápidas y entregar valor incremental temprano antes de continuar iterando. Análisis inicial nos ayuda a identificar el trabajo más pequeño posible que será útil lograr un resultado comercial usando la nueva arquitectura.

Página 211

Blackboard Inc. es uno de los pioneros en proporcionar tecnología para instituciones educativas, con ingresos anuales de aproximadamente $ 650 millones en 2011. En ese momento, el equipo de desarrollo para su producto insignia de Learn, software empaquetado que se instaló y ejecutado en las instalaciones de sus clientes, vivía con el diario

consecuencias de una base de código J2EE heredada que se remonta a 1997. Como observa David Ashman, su arquitecto principal, “todavía tenemos fragmentos de código Perl aún incrustados en toda nuestra base de código ".

En 2010, Ashman se centró en la complejidad y el liderazgo creciente. tiempos asociados con el antiguo sistema, observando que "nuestra construcción, integración, y los procesos de prueba se hicieron cada vez más complejo y propenso a errores. Y cuanto más grande es el producto, más tiempo nuestros plazos de entrega y, lo peor, los resultados para nuestros clientes. A incluso recibir comentarios de nuestro proceso de integración requeriría veinticuatro a treinta y seis horas.

Figura 23: *repositorio de código de Blackboard Learn: antes de los Building Blocks (Fuente: “DOES14 - David Ashman - Blackboard Learn - Mantenga la cabeza adentro las nubes ", video de YouTube, 30:43, publicado por DevOps Enterprise Summit 2014, 28 de octubre de 2014 https://www.youtube.com/watch?v=SSmixnMps14)*

Cómo esto comenzó a afectar la productividad del desarrollador se hizo visible a Ashman en gráficos generados a partir de su repositorio de código fuente desde el año 2005.

Página 212

En la figura 24, el gráfico superior representa el número de líneas de código.

en el repositorio de código monolítico de Blackboard Learn; El fondo
El gráfico representa el número de confirmaciones de código. El problema que
Ashman se hizo evidente que el número de confirmaciones de código
comenzó a disminuir, mostrando objetivamente la creciente dificultad de
introducir cambios en el código, mientras que el número de líneas de código
continuó aumentando. Ashman señaló: "Para mí, decía que necesitábamos
hacer algo, de lo contrario los problemas seguirían empeorando,
sin fin a la vista ".

Como resultado, en 2012 Ashman se centró en implementar un código
proyecto de arquitectura que usó el patrón estrangulador. El equipo
logrado esto creando internamente lo que llamaron *Edificio*
Bloques , que permitieron a los desarrolladores trabajar en módulos separados que
fueron desacoplados de la base de código monolítico y accedieron
a través de API fijas. Esto les permitió trabajar con mucho más
autonomía, sin tener que comunicarse constantemente y
coordinar con otros equipos de desarrollo.

Figura 24: *repositorio de código de Blackboard Learn: después de Building Blocks (Fuente: "DOES14 - David Ashman - Blackboard Learn - Mantenga la cabeza adentro las nubes." Video de YouTube, 30:43, publicado por DevOps Enterprise Summit 2014, 28 de octubre de 2014 , https://www.youtube.com/watch? v = SSmixnMpsI4 .)*

Cuando los Building Blocks se pusieron a disposición de los desarrolladores, el tamaño
del repositorio de código fuente monolítico comenzó a disminuir (como
medido por el número de líneas de código). Ashman explicó que esto
fue porque los desarrolladores estaban moviendo su código al Edificio
Bloque de repositorio de código fuente de módulos. "De hecho", informó Ashman,
"Cada desarrollador dado una opción trabajaría en el Building Block

codebase, donde podrían trabajar con más autonomía y
libertad y seguridad ".

El gráfico anterior muestra la conexión entre la exponencial
crecimiento en el número de líneas de código y el crecimiento exponencial
del número de confirmaciones de código para el código Building Blocks
repositorios La nueva base de código de Building Blocks permitida
los desarrolladores serán más productivos e hicieron el trabajo más seguro
porque los errores resultaron en fallas locales pequeñas en lugar de fallas mayores
catástrofes que impactaron el sistema global.

Ashman concluyó: "Hacer que los desarrolladores trabajen en el Edificio
La arquitectura de bloques hizo mejoras impresionantes en el código
modularidad, lo que les permite trabajar con más independencia y
libertad. En combinación con las actualizaciones de nuestro proceso de compilación,
También obtuve una respuesta más rápida y mejor sobre su trabajo, lo que significaba mejor
calidad. "

CONCLUSIÓN

En gran medida, la arquitectura dentro de la cual operan nuestros servicios
dicta cómo probamos e implementamos nuestro código. Esto fue validado en Puppet
Informe de estado de DevOps 2015 de Labs , que muestra que la arquitectura es una de
Los principales predictores de la productividad de los ingenieros que trabajan en ella.
y de cómo se pueden hacer cambios de forma rápida y segura.

Porque a menudo estamos atrapados con arquitecturas que fueron optimizadas para un
diferente conjunto de objetivos organizacionales, o para una era pasada, debemos
ser capaz de migrar de manera segura de una arquitectura a otra. El caso
estudios presentados en este capítulo, así como el estudio de caso de Amazon
presentado anteriormente, describa técnicas como el patrón de estrangulador que
puede ayudarnos a migrar entre arquitecturas de forma incremental, lo que nos permite
adaptarse a las necesidades de la organización.

PARTE III CONCLUSIÓN

En los capítulos anteriores de la Parte III, hemos implementado el Arquitectura y prácticas técnicas que permiten el flujo rápido de trabajo. de Dev a Ops, para que el valor se pueda entregar de forma rápida y segura a clientes.

En la Parte IV: La segunda vía, *las prácticas técnicas de retroalimentación* , nosotros creará la arquitectura y los mecanismos para permitir el recíproco rápido flujo de retroalimentación de derecha a izquierda, para buscar y solucionar problemas más rápido, irradiar comentarios y garantizar mejores resultados de nuestro trabajo. Esto permite nuestra organización para aumentar aún más la velocidad a la que puede adaptarse.

† La arquitectura de eBay pasó por las siguientes fases: Perl y archivos (v1, 1995), C ++ y Oracle (v2, 1997), XSL y Java (v3, 2002), Java de pila completa (v4, 2007), microservicios Polyglot (2013+).

‡ El patrón de aplicación de estrangulador implica reemplazar gradualmente un sistema completo, generalmente un sistema heredado, con un completamente nuevo Por el contrario, *ramificar por abstracción* , un término acuñado por Paul Hammant, es una técnica en la que crear una capa de abstracción entre las áreas que estamos cambiando. Esto permite el diseño evolutivo de la aplicación. arquitectura al tiempo que permite a todos trabajar desde el tronco / maestro y practicar la integración continua.

Parte

Introducción

En la Parte III, describimos la arquitectura y la técnica.
prácticas necesarias para crear un flujo rápido desde el desarrollo
en operaciones. Ahora en la Parte IV, describimos cómo
implementar las prácticas técnicas de la Segunda Vía,
que son necesarios para crear rápido y continuo
retroalimentación de Operaciones a Desarrollo.

Al hacer esto, acortamos y amplificamos los circuitos de retroalimentación para
que podemos ver problemas a medida que ocurren e irradiar esto
información para todos en la cadena de valor. Esto permite
nosotros para encontrar y solucionar problemas rápidamente antes en el software
ciclo de vida de desarrollo, idealmente mucho antes de que causen un
Fallo catastrófico.

Además, crearemos un sistema de trabajo donde el conocimiento adquirido aguas abajo en Operaciones es integrado en el trabajo ascendente de Desarrollo y Gestión de producto. Esto nos permite crear rápidamente mejoras y aprendizajes, ya sea de problema de producción, un problema de implementación, indicadores tempranos de problemas o nuestros patrones de uso del cliente.

Además, crearemos un proceso que permita todos para obtener comentarios sobre su trabajo, hace

información visible para permitir el aprendizaje, y nos permite Probar rápidamente hipótesis de productos, ayudándonos a determinar si Las características que estamos construyendo nos están ayudando a lograr nuestro metas organizacionales.

También demostraremos cómo crear telemetría desde nuestros procesos de compilación, prueba e implementación, así como del usuario comportamiento, problemas de producción e interrupciones, problemas de auditoría, y violaciones de seguridad. Amplificando señales como parte de nuestro trabajo diario, hacemos posible ver y resolver problemas a medida que ocurren, y cultivamos sistemas seguros de trabajo que nos permite realizar cambios y ejecutar con confianza Experimentos de productos, sabiendo que podemos detectar rápidamente y Remediación de fallas. Haremos todo esto explorando el siguiendo:

- Crear telemetría para permitir ver y resolver problemas

Usando nuestra telemetría para anticipar mejor los problemas y
lograr objetivos

Integrar la investigación y los comentarios de los usuarios en el trabajo.
de equipos de producto

Habilitar la retroalimentación para que Dev y Ops puedan realizar de forma segura
implementaciones

Permitir comentarios para aumentar la calidad de nuestro trabajo.
a través de revisiones por pares y programación de pares

Los patrones en este capítulo ayudan a reforzar lo común
objetivos de gestión de productos, desarrollo, control de calidad,

Operaciones e Infosec, y aliéntelos a compartir
la responsabilidad de garantizar que los servicios funcionen sin problemas
en producción y colaborar en la mejora de la
sistema en su conjunto. Donde sea posible, queremos vincular causa
para efectuar Cuantos más supuestos podamos invalidar, más
más rápido podemos descubrir y solucionar problemas, pero también
mayor nuestra capacidad de aprender e innovar.

A lo largo de los siguientes capítulos, implementaremos
bucles de retroalimentación, permitiendo a todos trabajar juntos
hacia objetivos compartidos, para ver los problemas a medida que ocurren,
habilite la detección y recuperación rápidas y asegúrese de que
las características no solo funcionan según lo diseñado en la producción, sino que también

también lograr objetivos organizacionales y apoyo
Aprendizaje organizacional.

14 Crear telemetría para Habilitar ver y resolver Problemas

Un hecho de la vida en Operaciones es que las cosas van mal: pequeños cambios pueden resultar en muchos resultados inesperados, incluidos cortes y fallas globales que afectan a todos nuestros clientes.
Esta es la realidad de operar sistemas complejos; ninguna persona sola puede ver todo el sistema

y entender cómo encajan todas las piezas.

Cuando ocurren interrupciones en la producción y otros problemas en nuestro trabajo diario, a menudo no tenemos La información que necesitamos para resolver el problema. Por ejemplo, durante una interrupción, es posible que no poder determinar si el problema se debe a una falla en nuestra aplicación (por ejemplo, un defecto en el código), en nuestro entorno (por ejemplo, un problema de red, un problema de configuración del servidor), o algo completamente externo a nosotros (por ejemplo, un ataque masivo de denegación de servicio).

En Operaciones, podemos tratar este problema con la siguiente regla general: Cuando algo sale mal en la producción, solo reiniciamos el servidor. Si eso no funciona, reinicie el servidor al lado Si eso no funciona, reinicie todos los servidores. Si eso no funciona, culpa los desarrolladores, siempre están causando interrupciones.

En contraste, el estudio de Microsoft Operations Framework (MOF) en 2001 encontró que Las organizaciones con los niveles de servicio más altos reiniciaron sus servidores veinte veces menos con frecuencia que el promedio y tenía cinco veces menos "pantallas azules de la muerte". En otras palabras, descubrieron que las organizaciones con mejor desempeño eran mucho mejores para diagnosticar y arreglando incidentes de servicio, en lo que Kevin Behr, Gene Kim y George Spafford llamaron un "Cultura de causalidad" en *el Manual de Visible Ops* . Los de alto rendimiento utilizaron una disciplina enfoque para resolver problemas, utilizando la telemetría de producción para comprender posibles factores contribuyentes para enfocar su resolución de problemas, en oposición a los de bajo rendimiento que reiniciaría ciegamente los servidores.

Para permitir este comportamiento disciplinado de resolución de problemas, necesitamos diseñar nuestros sistemas de modo que continuamente crean *telemetría* , ampliamente definida como "una comunicación automatizada proceso por el cual las mediciones y otros datos se recopilan en puntos remotos y son posteriormente transmitido al equipo receptor para su monitoreo ". Nuestro objetivo es crear telemetría dentro de nuestras aplicaciones y entornos, tanto en nuestra producción y pre entornos de producción, así como en nuestra tubería de implementación.

Michael Rembetsy y Patrick McDonnell describieron cómo la supervisión de la producción era parte crítica de la transformación DevOps de Etsy que comenzó en 2009. Esto se debió a que estaban estandarizando y haciendo la transición de toda su pila de tecnología a la pila LAMP (Linux, Apache, MySQL y PHP), abandonando una miríada de tecnologías diferentes utilizado en la producción que eran cada vez más difíciles de soportar.

En la Conferencia de velocidad de 2012, McDonnell describió cuánto riesgo creó esto, "Nosotros estaban cambiando algunas de nuestras infraestructuras más críticas, que, idealmente, los clientes

Nunca te des cuenta. Sin embargo, definitivamente se darían cuenta si arruinamos algo. Nosotros necesitabamos más métricas para darnos la confianza de que en realidad no estábamos rompiendo cosas mientras estábamos haciendo estos grandes cambios, tanto para nuestros equipos de ingeniería como para los miembros del equipo en áreas técnicas, como el marketing ".

McDonnell explicó además: "Comenzamos a recopilar toda la información de nuestro servidor en una herramienta

llamado Ganglia, que muestra toda la información en Graphite, una herramienta de código abierto que invertido fuertemente en. Comenzamos a agregar métricas juntas, desde negocios métricas para implementaciones. Esto es cuando modificamos Graphite con lo que llamamos 'nuestro tecnología de línea vertical inigualable e inigualable 'que se superpone en cada gráfico métrico cuando ocurrieron los despliegues. Al hacer esto, podríamos ver más rápidamente cualquier involuntario despliegue de efectos secundarios. Incluso comenzamos a colocar pantallas de TV por toda la oficina para que todos podían ver cómo funcionaban nuestros servicios ".

Al permitir a los desarrolladores agregar telemetría a sus funciones como parte de su trabajo diario, ellos creó suficiente telemetría para ayudar a que las implementaciones sean seguras. Para 2011, Etsy estaba rastreando doscientos mil métricas de producción en cada capa de la pila de aplicaciones (por ejemplo, características de la aplicación, estado de la aplicación, base de datos, sistema operativo, almacenamiento, redes, seguridad, etc.) con las treinta métricas comerciales más importantes exhibidas de manera destacada en su "panel de despliegue". Para 2014, estaban rastreando más de ochocientos mil métricas, que muestran su implacable objetivo de instrumentar todo y facilitar ingenieros para hacerlo.

Como Ian Malpass, ingeniero de Etsy, bromeó: "Si la ingeniería en Etsy tiene una religión, es la Iglesia de los Gráficos. Si se mueve, lo rastreamos. A veces dibujaremos una gráfica de algo que todavía no se está moviendo, en caso de que decida huir ... Seguir todo es clave para moverse rápido, pero la única forma de hacerlo es hacer que el seguimiento sea fácil ... Permitimos ingenieros para rastrear lo que necesitan rastrear, en un abrir y cerrar de ojos, sin requerir tiempo chupando cambios de configuración o procesos complicados ".

Una de las conclusiones del *Informe del estado de DevOps de 2015* fue que los de alto rendimiento podrían resolver incidentes de producción 168 veces más rápido que sus pares, con la mediana alta artista intérprete o ejecutante que tiene un MTTR medido en minutos, mientras que el intérprete medio bajo tuvo un MTTR medido en días. Las dos principales prácticas técnicas que permitieron el MTTR rápido fueron las uso del control de versiones por parte de Operaciones y tener telemetría y monitoreo proactivo en el entorno de producción.

Figura 25: *Tiempo de resolución de incidentes para artistas de alto, medio y bajo rendimiento.*
(Fuente: Puppet Labs, Informe del estado de DevOps 2014 *).*

Como se creó en Etsy, nuestro objetivo en este capítulo es garantizar que siempre tengamos suficiente
telemetría para que podamos confirmar que nuestros servicios funcionan correctamente en producción.
Y cuando surjan problemas, haga posible determinar rápidamente qué va mal
y tomar decisiones informadas sobre la mejor manera de solucionarlo, idealmente mucho antes de que los clientes estén
impactado Además, la telemetría es lo que nos permite reunir nuestra mejor comprensión.
de la realidad y detectar cuando nuestra comprensión de la realidad es incorrecta.

CREA NUESTRA TELEMETRÍA CENTRALIZADA INFRAESTRUCTURA

La monitorización y el registro operativo no son nuevos: múltiples generaciones de
Los ingenieros de operaciones han utilizado y personalizado marcos de monitoreo (por ejemplo, HP
OpenView, IBM Tivoli y BMC Patrol / BladeLogic) para garantizar el buen estado de la producción
sistemas. Los datos generalmente se recopilaban a través de agentes que se ejecutaban en servidores o a través de agentes
menos monitoreo (por ejemplo, trampas SNMP o monitores basados en encuestas). A menudo había un gráfico
Interfaz de usuario (GUI) front-end, y los informes de back-end a menudo se aumentaron a través de herramientas
como Crystal Reports.

Del mismo modo, las prácticas de desarrollo de aplicaciones con registro efectivo y gestión de
la telemetría resultante no es nueva: existe una variedad de bibliotecas de registro maduras para casi todos
lenguajes de programación.

Sin embargo, durante décadas hemos terminado con silos de información, donde el desarrollo solo crea eventos de registro que son interesantes para los desarrolladores, y Operations solo monitorea si los ambientes están arriba o abajo. Como resultado, cuando ocurren eventos inoportunos, no se puede determinar por qué todo el sistema no está funcionando como se diseñó o qué El componente está fallando, lo que impide nuestra capacidad de devolver nuestro sistema a un estado de funcionamiento.

Para que podamos ver todos los problemas a medida que ocurren, debemos diseñar y desarrollar nuestro aplicaciones y entornos para que generen suficiente telemetría, lo que nos permite Comprender cómo se comporta nuestro sistema en su conjunto. Cuando todos los niveles de nuestra aplicación se apilan tenemos monitoreo y registro, habilitamos otras capacidades importantes, como gráficos y visualizar nuestras métricas, detección de anomalías, alertas proactivas y escalada, etc.

En *The Art of Monitoring* , James Turnbull describe una arquitectura de monitoreo moderna, que ha sido desarrollado y utilizado por ingenieros de operaciones en empresas de escala web (por ejemplo, Google, Amazon, Facebook). La arquitectura a menudo consistía en herramientas de código abierto, como Nagios y Zenoss, que fueron personalizados y desplegados a una escala que era difícil de cumplir con el software comercial con licencia en el momento. Esta arquitectura tiene el siguientes componentes:

Recolección de datos en la capa de lógica de negocios, aplicaciones y entornos: En cada una de estas capas, estamos creando telemetría en forma de eventos, registros y métrica. Los registros pueden almacenarse en archivos específicos de la aplicación en cada servidor (por ejemplo, /var/log/httpd-error.log), pero preferiblemente queremos que todos nuestros registros se envíen a un servicio común que permite una fácil centralización, rotación y eliminación. Esto es proporcionado por la mayoría sistemas operativos, como syslog para Linux, el registro de eventos para Windows, etc. Además, recopilamos métricas en todas las capas de la pila de aplicaciones para mejorar Comprender cómo se comporta nuestro sistema. A nivel del sistema operativo, podemos recopilar métricas como uso de CPU, memoria, disco o red a lo largo del tiempo utilizando herramientas como collectd, Ganglios, etc. Otras herramientas que recopilan información de rendimiento incluyen AppDynamics, Nueva reliquia y Pingdom.

Un enrutador de eventos responsable de almacenar nuestros eventos y métricas: esto la capacidad potencialmente permite la visualización, tendencias, alertas, detección de anomalías y etc. Al recopilar, almacenar y agregar toda nuestra telemetría, habilitamos mejor análisis posteriores y controles de salud. Aquí también es donde almacenamos configuraciones relacionadas con nuestros servicios (y sus aplicaciones y entornos compatibles) y es probable que donde hacemos alertas basadas en umbrales y controles de salud. †

Una vez que hemos centralizado nuestros registros, podemos transformarlos en métricas contándolos en el enrutador de eventos, por ejemplo, un evento de registro como "señal de salida child pid 14024 Falla de segmentación "se puede contar y resumir como una métrica de segfault única en nuestro Toda la infraestructura de producción.

Al transformar los registros en métricas, ahora podemos realizar operaciones estadísticas en ellos, como como usar la detección de anomalías para encontrar valores atípicos y variaciones incluso antes en el ciclo del problema.

Por ejemplo, podríamos configurar nuestras alertas para que nos notifiquen si pasamos de "diez segfaults por último semana "a" miles de segfaults en la última hora ", lo que nos lleva a investigar más a fondo.

Además de recopilar telemetría de nuestros servicios y entornos de producción, nosotros también debe recopilar telemetría de nuestra canalización de implementación cuando se producen eventos importantes,

como cuando nuestras pruebas automatizadas pasan o fallan y cuando realizamos implementaciones en cualquier ambiente. También debemos recopilar telemetría sobre cuánto tiempo nos lleva ejecutar nuestras compilaciones y pruebas. Al hacer esto, podemos detectar condiciones que podrían indicar problemas, como si la prueba de rendimiento o nuestra construcción dura el doble de lo normal, lo que nos permite encontrar y solucionar errores antes de que entren en producción.

Figura 26: *Marco de monitoreo (Fuente: Turnbull,* The Art of Monitoring *, edición Kindle, Cap. 2.)*

Además, debemos asegurarnos de que sea fácil ingresar y recuperar información de nuestro Infraestructura de telemetría. Preferiblemente, todo debe hacerse a través de API de autoservicio, en lugar de exigir a las personas que abran boletos y esperen para recibir informes.

Idealmente, crearemos telemetría que nos diga exactamente cuándo sucede algo de interés, como

así como dónde y cómo. Nuestra telemetría también debe ser adecuada para uso manual y automatizado.
análisis y debe poder analizarse sin tener la aplicación que produjo
Los troncos a mano. Como señaló Adrian Cockcroft, "El monitoreo es tan importante que nuestro
los sistemas de monitoreo deben estar más disponibles y escalables que los sistemas que están siendo
monitoreado ".

De aquí en adelante, el término *telemetría* se usará indistintamente con *métricas* , que
incluye todo el registro de eventos y las métricas creadas por nuestros servicios en todos los niveles de nuestra aplicación
apilar y generar a partir de todos nuestros entornos de producción y preproducción, así como
de nuestra tubería de implementación.

CREAR TELEMETRÍA DE REGISTRO DE APLICACIONES QUE AYUDA PRODUCCIÓN

Ahora que tenemos una infraestructura de telemetría centralizada, debemos asegurarnos de que
Las aplicaciones que creamos y operamos están creando suficiente telemetría. Hacemos esto al tener
Los ingenieros de Dev y Ops crean telemetría de producción como parte de su trabajo diario, ambos para
Servicios nuevos y existentes.

Scott Prugh, arquitecto jefe y vicepresidente de desarrollo de CSG, dijo: "Cada vez
La NASA lanza un cohete, tiene millones de sensores automáticos que informan el estado de cada
componente de este valioso activo. Y, sin embargo, a menudo no tenemos el mismo cuidado con el software.
- descubrimos que la creación de telemetría de aplicaciones e infraestructura es una de las más altas
devolvemos las inversiones que hemos realizado. En 2014, creamos más de mil millones de eventos de telemetría por
día, con más de cien mil ubicaciones de códigos instrumentadas ".

En las aplicaciones que creamos y operamos, cada característica debe ser instrumentada, si fuera
lo suficientemente importante como para que un ingeniero lo implemente, sin duda es lo suficientemente importante como para
generar suficiente telemetría de producción para que podamos confirmar que está funcionando como
diseñado y que los resultados deseados se están logrando.[1]

Cada miembro de nuestro flujo de valor utilizará la telemetría de varias maneras. Por ejemplo,
los desarrolladores pueden crear temporalmente más telemetría en su aplicación para diagnosticar mejor
problemas en su estación de trabajo, mientras que los ingenieros de Ops pueden usar la telemetría para diagnosticar un
problema de producción Además, Infosec y los auditores pueden revisar la telemetría para confirmar
la efectividad de un control requerido, y un gerente de producto puede usarlos para rastrear
resultados comerciales, uso de funciones o tasas de conversión.

Para admitir estos diversos modelos de uso, tenemos diferentes niveles de registro, algunos de los cuales
También puede activar alertas, como las siguientes:

> **Nivel de DEPURACIÓN: la** información en este nivel trata sobre todo lo que sucede en el
> programa, más utilizado durante la depuración. A menudo, los registros de depuración están deshabilitados en

producción pero temporalmente habilitado durante la resolución de problemas.

Nivel INFO: la información en este nivel consiste en acciones dirigidas por el usuario o por el sistema específico (por ejemplo, "inicio de transacción con tarjeta de crédito").

Nivel de ADVERTENCIA: la información en este nivel nos informa de las condiciones que podrían convertirse en un error (por ejemplo, una llamada a la base de datos que demora más de un tiempo predefinido). Estas probablemente iniciará una alerta y solución de problemas, mientras que otros mensajes de registro pueden ayudar Comprendemos mejor lo que condujo a esta condición.

Nivel de ERROR: la información en este nivel se centra en las condiciones de error (por ejemplo, llamada API fallas, condiciones de error interno).

Nivel FATAL: la información en este nivel nos dice cuándo debemos terminar (por ejemplo, un network daemon no puede enlazar un socket de red).

Elegir el nivel de registro correcto es importante. Dan North, un antiguo ThoughtWorks consultor que participó en varios proyectos en los que la entrega continua central

Página 226

los conceptos tomaron forma, observa: "Al decidir si un mensaje debe ser ERROR o AVISO, imagine que lo despiertan a las 4 a. M. El tóner de impresora bajo no es un ERROR ".

Para ayudar a garantizar que tengamos información relevante para las operaciones confiables y seguras de nuestro servicio, debemos asegurarnos de que todos los eventos de aplicaciones potencialmente importantes generen entradas de registro, incluidas las proporcionadas en esta lista reunidas por Anton A. Chuvakin, un vicepresidente de investigación en el grupo de gestión de riesgos y seguridad GTP de Gartner:

Decisiones de autenticación / autorización (incluido el cierre de sesión)

Sistema y acceso a datos

Cambios en el sistema y la aplicación (especialmente cambios privilegiados)

Cambios de datos, como agregar, editar o eliminar datos

Entrada no válida (posible inyección maliciosa, amenazas, etc.)

Recursos (RAM, disco, CPU, ancho de banda o cualquier otro recurso que tenga hard o soft límites)

Salud y disponibilidad.

Startups y paradas

Fallos y errores

Disyuntores de circuito

Retrasos

Copia de seguridad exitosa / fallida

Para facilitar la interpretación y el significado de todas estas entradas de registro, deberíamos (idealmente) crear categorías jerárquicas de registro, como para atributos no funcionales (por ejemplo, rendimiento, seguridad) y para los atributos relacionados con las características (por ejemplo, búsqueda, clasificación).

USE LA TELEMETRÍA PARA GUIAR LA RESOLUCIÓN DE PROBLEMAS

Como se describió al comienzo de este capítulo, los de alto rendimiento utilizan un enfoque disciplinado para resolver problemas Esto está en contraste con la práctica más común de usar el rumor y rumores, que pueden conducir a la lamentable métrica del *tiempo medio hasta que se declare inocente* : ¿Qué tan rápido podemos convencer a todos los demás de que no causamos la interrupción?

Cuando existe una cultura de culpa en torno a las interrupciones y los problemas, los grupos pueden evitar documentar cambios y mostrar telemetría donde todos puedan verlos para evitar ser culpado por interrupciones.

Otros resultados negativos debido a la falta de telemetría pública incluyen una política altamente cargada ambiente, la necesidad de desviar las acusaciones y, lo que es peor, la incapacidad de crear instituciones conocimiento sobre cómo ocurrieron los incidentes y los aprendizajes necesarios para prevenirlos los errores vuelven a suceder en el futuro.§

Página 227

Por el contrario, la telemetría nos permite utilizar el método científico para formular hipótesis. sobre qué está causando un problema en particular y qué se requiere para resolverlo. Ejemplos de Las preguntas que podemos responder durante la resolución del problema incluyen:

¿Qué evidencia tenemos de nuestro monitoreo de que realmente está ocurriendo un problema?

¿Cuáles son los eventos y cambios relevantes en nuestras aplicaciones y entornos que podría haber contribuido al problema?

¿Qué hipótesis podemos formular para confirmar el vínculo entre las causas propuestas? y efectos?

¿Cómo podemos probar cuáles de estas hipótesis son correctas y efectúan con éxito una solución?

El valor de la resolución de problemas basada en hechos radica no solo en un MTTR significativamente más rápido (y mejores resultados para el cliente), pero también en su refuerzo de la percepción de ganar / ganar relación entre Desarrollo y Operaciones.

Página 228

HABILITAR LA CREACIÓN DE MEDIDAS DE PRODUCCIÓN COMO PARTE DE TRABAJO DIARIO

Para que todos puedan encontrar y solucionar problemas en su trabajo diario, necesitamos
Permitir a todos crear métricas en su trabajo diario que se puedan crear, mostrar,
y analizado Para hacer esto, debemos crear la infraestructura y las bibliotecas necesarias para hacer
Es lo más fácil posible para cualquier persona en Desarrollo u Operaciones crear telemetría para cualquier
funcionalidad que construyen. En el ideal, debería ser tan fácil como escribir una línea de código en

crear una nueva métrica que se muestre en un tablero común donde todos en el valor
corriente puede verlo.

Esta fue la filosofía que guió el desarrollo de uno de los más utilizados.
bibliotecas de métricas, llamadas StatsD, que se creó y de código abierto en Etsy. Como John
Allspaw describió: “Diseñamos StatsD para evitar que cualquier desarrollador diga: 'También es
mucha molestia para instrumentar mi código. Ahora pueden hacerlo con una línea de código. Era
importante para nosotros que para un desarrollador, agregar telemetría de producción no se sintiera tan difícil como
haciendo un cambio de esquema de base de datos ".

StatsD puede generar temporizadores y contadores con una línea de código (en Ruby, Perl, Python, Java,
y otros idiomas) y a menudo se usa junto con Graphite o Grafana, que
renderiza eventos métricos en gráficos y tableros.

Figura 27: *Una línea de código para generar telemetría usando StatsD y Graphite en Etsy (Fuente: Ian Malpass, "Medir cualquier cosa, medir todo".*

La Figura 27 anterior muestra un ejemplo de cómo una sola línea de código crea un evento de inicio de sesión de usuario (en
en este caso, una línea de código PHP: "StatsD :: increment (" login.successes ")). La resultante
El gráfico muestra el número de inicios de sesión exitosos y fallidos por minuto, y superpuestos en
Los gráficos son líneas verticales que representan una implementación de producción.

Cuando generamos gráficos de nuestra telemetría, también los superpondremos cuando
se producen cambios en la producción, porque sabemos que la gran mayoría de la producción
los problemas son causados por cambios en la producción, que incluyen implementaciones de código. Esto es parte de
lo que nos permite tener una alta tasa de cambio, al tiempo que conservamos un sistema de trabajo seguro.

Las bibliotecas alternativas a StatsD que permiten a los desarrolladores generar telemetría de producción pueden
ser agregado y analizado fácilmente incluye JMX y métricas codahale. Otras herramientas que
crear métricas invaluables para la resolución de problemas incluyen New Relic, AppDynamics y

Dynatrace Se pueden usar herramientas como munin y collectd para crear una funcionalidad similar. ¶

Al generar telemetría de producción como parte de nuestro trabajo diario, creamos una mejora constante capacidad no solo de ver los problemas a medida que ocurren, sino también de diseñar nuestro trabajo para que Se pueden revelar problemas en el diseño y las operaciones, lo que permite un número creciente de métricas a rastrear, como vimos en el estudio de caso de Etsy.

CREAR ACCESO DE AUTOSERVICIO A LA TELEMETRÍA Y RADIADORES DE INFORMACIÓN

En los pasos anteriores, permitimos que Desarrollo y Operaciones crearan y mejoraran telemetría de producción como parte de su trabajo diario. En este paso, nuestro objetivo es irradiar esto información al resto de la organización, asegurando que cualquiera que quiera información sobre cualquiera de los servicios que estamos ejecutando puede obtenerlo sin necesidad de un sistema de producción acceso o cuentas privilegiadas, o tener que abrir un ticket y esperar días para que alguien configurar el gráfico para ellos.

Al hacer que la telemetría sea rápida, fácil de obtener y suficientemente centralizada, todos los que están en el valor Stream puede compartir una visión común de la realidad. Por lo general, esto significa que las métricas de producción se irradiará en páginas web generadas por un servidor centralizado, como Graphite o cualquiera de Las otras tecnologías descritas en la sección anterior.

Queremos que nuestra telemetría de producción sea altamente visible, lo que significa ponerla en el centro áreas donde trabajan el Desarrollo y las Operaciones, lo que permite a todos los interesados para ver cómo están funcionando nuestros servicios. Como mínimo, esto incluye a todos en nuestro valor flujo, como Desarrollo, Operaciones, Gestión de productos e Infosec.

Esto a menudo se conoce como un *radiador de información* , definido por Agile Alliance como "el término genérico para cualquiera de una serie de pantallas escritas a mano, dibujadas, impresas o electrónicas que un equipo coloca en una ubicación muy visible, de modo que todos los miembros del equipo, así como los transeúntes pueden ver la información más reciente de un vistazo: recuento de pruebas automatizadas, velocidad, informes de incidentes, estado de integración continua, etc. Esta idea se originó como parte de el sistema de producción de Toyota ".

Al colocar radiadores de información en lugares altamente visibles, promovemos la responsabilidad entre miembros del equipo, demostrando activamente los siguientes valores:

> El equipo no tiene nada que ocultar a sus visitantes (clientes, partes interesadas, etc.)
>
> El equipo no tiene nada que esconderse de sí mismo: reconoce y enfrenta los problemas.

Ahora que poseemos la infraestructura para crear e irradiar telemetría de producción a toda la organización, también podemos optar por transmitir esta información a nuestro interno clientes e incluso a nuestros clientes externos. Por ejemplo, podríamos hacer esto creando páginas de estado del servicio que se pueden ver públicamente para que los clientes puedan aprender cómo funcionan los servicios dependerá de lo que están realizando.

Aunque puede haber cierta resistencia para proporcionar esta cantidad de transparencia, Ernest Mueller describe el valor de hacerlo:

> Una de las primeras acciones que tomo al comenzar en una organización es usar información radiadores para comunicar problemas y detallar los cambios que estamos haciendo, esto suele ser extremadamente bien recibido por nuestras unidades de negocio, que a menudo se quedaron en la oscuridad antes. Y para los grupos de Desarrollo y Operaciones que deben trabajar juntos para entregar un servicio a los demás, necesitamos esa comunicación constante, información y comentarios.

Incluso podemos extender aún más esta transparencia, en lugar de tratar de mantener al cliente, Impactando problemas en secreto, podemos transmitir esta información a nuestros clientes externos. Esto demuestra que valoramos la transparencia, ayudando así a construir y ganar La confianza de los clientes.** Ver Apéndice 10 .

> Como se describe en la Parte III, LinkedIn se creó en 2003 para ayudar a los usuarios a conectarse "a su red para mejores oportunidades de trabajo ". Para noviembre de 2015, LinkedIn tenía más de 350 millones de miembros que generan decenas de miles de solicitudes por segundo, lo que resulta en millones de consultas por segundo en los sistemas de back-end de LinkedIn.
>
> Prachi Gupta, Director de Ingeniería en LinkedIn, escribió en 2011 sobre el importancia de la telemetría de producción: “En LinkedIn, enfatizamos asegurarnos de que el sitio está activo y nuestros miembros tienen acceso a la funcionalidad completa del sitio en todo momento. Cumplir este compromiso requiere que detectemos y respondamos a fallas y cuellos de botella a medida que comienzan a suceder. Es por eso que usamos estos gráficos de series de tiempo para monitoreo del sitio para detectar y reaccionar ante incidentes en minutos ... Este monitoreo La técnica ha demostrado ser una gran herramienta para los ingenieros. Nos permite movernos rápido y comprar es hora de detectar, clasificar y solucionar problemas ".
>
> Sin embargo, en 2010, a pesar de que había un volumen increíblemente grande de telemetría Al generarse, era extremadamente difícil para los ingenieros obtener acceso a los datos. solo analizarlo. Así comenzó el proyecto de pasante de verano de Eric Wong en LinkedIn, que se convirtió en la iniciativa de telemetría de producción que creó InGraphs.
>
> Wong escribió: "Para obtener algo tan simple como el uso de CPU de todos los hosts que ejecutan un servicio particular, necesitaría presentar un boleto y alguien gastaría 30 minutos preparándolo [un informe] ".
>
> En ese momento, LinkedIn estaba usando Zenoss para recopilar métricas, pero como explica Wong, "Obtener datos de Zenoss requería cavar a través de una interfaz web lenta, así que escribí algunos scripts de python para ayudar a agilizar el proceso. Mientras todavía había manual intervención en la configuración de la recopilación de indicadores, pude reducir el tiempo dedicado navegando por la interfaz de Zenoss ".
>
> En el transcurso del verano, continuó agregando funcionalidad a InGraphs para que los ingenieros podían ver exactamente lo que querían ver, agregando la capacidad de hacer cálculos en múltiples conjuntos de datos, ver tendencias semana a semana para comparar rendimiento histórico e incluso definir paneles personalizados para elegir exactamente qué las métricas se mostrarían en una sola página.

Al escribir sobre los resultados de agregar funcionalidad a InGraphs y el valor de esto
Gupta señala que "se destacó la eficacia de nuestro sistema de monitoreo
en un instante donde nuestra funcionalidad de monitoreo de InGraphs está vinculada a un correo web importante
el proveedor comenzó a bajar y se dio cuenta de que tenía un problema en
¡su sistema solo después de que nos comunicamos con ellos!

Lo que comenzó como un proyecto de pasantía de verano es ahora una de las partes más visibles.
de las operaciones de LinkedIn. InGraphs ha tenido tanto éxito que los gráficos en tiempo real
aparecen prominentemente en las oficinas de ingeniería de la compañía donde los visitantes no pueden fallar
para verlos.

ENCUENTRE Y LLENE CUALQUIER LABOR DE TELEMETRÍA

Ahora hemos creado la infraestructura necesaria para crear rápidamente telemetría de producción.
en toda nuestra pila de aplicaciones e irradiarla en toda nuestra organización.

En este paso, identificaremos cualquier brecha en nuestra telemetría que impida nuestra capacidad de rápidamente
detectar y resolver incidentes; esto es especialmente relevante si Dev y Ops actualmente tienen poco
(o no) telemetría. Utilizaremos estos datos más adelante para anticipar mejor los problemas, así como para
Permitir a todos recopilar la información que necesitan para tomar mejores decisiones para lograr
metas organizacionales.

Lograr esto requiere que creamos suficiente telemetría en todos los niveles de la pila de aplicaciones
para todos nuestros entornos, así como para las canalizaciones de implementación que los admiten. Nosotros
necesita métricas de los siguientes niveles:

Nivel empresarial: los ejemplos incluyen el número de transacciones de ventas, los ingresos por ventas
transacciones, registros de usuarios, tasa de abandono, resultados de pruebas A / B, etc.

Nivel de aplicación: los ejemplos incluyen tiempos de transacción, tiempos de respuesta del usuario,
fallas de aplicación, etc.

**Nivel de infraestructura (p. Ej., Base de datos, sistema operativo, redes,
almacenamiento): los** ejemplos incluyen el tráfico del servidor web, la carga de la CPU, el uso del disco, etc.

**Nivel de software del cliente (por ejemplo, JavaScript en el navegador del cliente, móvil
aplicación): los** ejemplos incluyen errores de aplicación y bloqueos, medidos por el usuario
tiempos de transacción, etc.

Nivel de canalización de implementación: los ejemplos incluyen el estado de la tubería de construcción (por ejemplo, rojo o
verde para nuestros diversos conjuntos de pruebas automatizadas), cambiar los plazos de implementación,
frecuencias de despliegue, promociones de entorno de prueba y estado del entorno.

Al tener cobertura de telemetría en todas estas áreas, podremos ver la salud de
todo en lo que se basa nuestro servicio, utilizando datos y hechos en lugar de rumores,
señalando, culpa, y así sucesivamente.

Además, habilitamos mejor la detección de eventos relevantes para la seguridad al monitorear cualquier
fallas en la aplicación y la infraestructura (por ejemplo, terminaciones anormales del programa, aplicación
errores y excepciones, y errores de servidor y almacenamiento). No solo esta telemetría mejora

informar a Desarrollo y Operaciones cuando nuestros servicios fallan, pero estos errores son
a menudo indican que una vulnerabilidad de seguridad está siendo explotada activamente.

Al detectar y corregir problemas antes, podemos solucionarlos mientras son pequeños y fáciles
para arreglar, con menos clientes afectados. Además, después de cada incidente de producción, nosotros
debería identificar cualquier telemetría faltante que podría haber permitido una detección más rápida y
recuperación; o, mejor aún, podemos identificar estas brechas durante el desarrollo de características en nuestros pares
proceso de revisión.

APLICACIÓN Y MEDIDAS EMPRESARIALES

A nivel de aplicación, nuestro objetivo es asegurarnos de que estamos generando telemetría no solo
en torno al estado de la aplicación (p. ej., uso de memoria, recuento de transacciones, etc.), pero también a
medir en qué medida estamos logrando nuestros objetivos organizacionales (por ejemplo, cantidad de nuevos
usuarios, eventos de inicio de sesión de usuario, duración de las sesiones de usuario, porcentaje de usuarios activos, con qué frecuencia
características están siendo utilizadas, y así sucesivamente).

Por ejemplo, si tenemos un servicio que respalda el comercio electrónico, queremos asegurarnos de que
tener telemetría en torno a todos los eventos del usuario que conducen a una transacción exitosa que
genera ingresos Entonces podemos instrumentar todas las acciones del usuario que se requieren para nuestro
resultados deseados del cliente.

Estas métricas variarán según los diferentes dominios y objetivos de la organización. por
por ejemplo, para sitios de comercio electrónico, es posible que deseemos maximizar el tiempo que pasamos en el sitio;
sin embargo, para los motores de búsqueda, es posible que deseemos reducir el tiempo que pasamos en el sitio, ya que
Las sesiones pueden indicar que los usuarios tienen dificultades para encontrar lo que buscan.

En general, las métricas comerciales serán parte de un *embudo de adquisición de clientes* , que es el
pasos teóricos que un cliente potencial tomará para realizar una compra. Por ejemplo, en un e-
sitio de comercio, los eventos de viaje medibles incluyen tiempo total en el sitio, enlace del producto
clics, adiciones al carrito de compras y pedidos completados.

Ed Blankenship, Gerente de Producto Senior para Microsoft Visual Studio Team Services,
describe: "A menudo, los equipos de características definirán sus objetivos en un embudo de adquisición, con el
Objetivo de su característica que se utiliza en el trabajo diario de cada cliente. A veces son
descrito informalmente como 'pateadores de neumáticos', 'usuarios activos', 'usuarios comprometidos' y 'profundamente comprometidos
usuarios ', con telemetría que respalda cada etapa ".

Nuestro objetivo es hacer que todas las métricas comerciales sean *accionables;* estas métricas principales deberían ayudar informar cómo cambiar nuestro producto y ser susceptible de experimentación y pruebas A / B.
Cuando las métricas no son accionables, es probable que sean métricas de vanidad que brindan poca utilidad información: queremos almacenarla, pero probablemente no mostrarla, y mucho menos alertar.

Idealmente, cualquiera que vea nuestros radiadores de información podrá entender el información que estamos mostrando en el contexto de los resultados organizacionales deseados, como objetivos en torno a los ingresos, el logro del usuario, las tasas de conversión, etc. Debemos definir y vincular cada métrica a una métrica de resultado comercial en las primeras etapas de definición de características y desarrollo y medir los resultados después de implementarlos en producción.
Además, hacer esto ayuda a los propietarios de productos a describir el contexto comercial de cada característica para todos en el flujo de valor.

Página 233

Figura 28: *Cantidad de entusiasmo del usuario por las nuevas funciones en las publicaciones del foro de usuarios después de las implementaciones (Fuente: Mike Brittain, "Tracking Every Release", CodeasCraft.com, 8 de diciembre de 2010, https://codeascraft.com/2010/12/08/track-every-release/.)*

Se puede crear un contexto comercial adicional al conocer y mostrar visualmente el tiempo períodos relevantes para la planificación y operaciones comerciales de alto nivel, tales como transacciones altas períodos asociados con temporadas pico de ventas en días festivos, cierre financiero de fin de trimestre períodos o auditorías de cumplimiento programadas. Esta información puede usarse como un recordatorio para evite programar cambios riesgosos cuando la disponibilidad sea crítica o evite ciertas actividades cuando Las auditorías están en progreso.

Al irradiar cómo los clientes interactúan con lo que construimos en el contexto de nuestros objetivos, nosotros permitir comentarios rápidos para presentar equipos para que puedan ver si las capacidades que tenemos En realidad, se está utilizando el edificio y hasta qué punto están logrando los objetivos comerciales. Como un

resultado, reforzamos las expectativas culturales que instrumentan y analizan al cliente
el uso también es parte de nuestro trabajo diario, por lo que entendemos mejor cómo nuestro trabajo contribuye a
nuestros objetivos organizacionales

MEDIDAS DE INFRAESTRUCTURA

Tal como lo hicimos para las métricas de aplicación, nuestro objetivo para la producción y la no producción
la infraestructura es para garantizar que estamos generando suficiente telemetría para que si surge un problema
ocurre en cualquier entorno, podemos determinar rápidamente si la infraestructura es un
causa contribuyente del problema. Además, debemos ser capaces de determinar exactamente qué
en la infraestructura está contribuyendo al problema (por ejemplo, base de datos, sistema operativo,
almacenamiento, redes, etc.).

Queremos hacer visible la mayor cantidad de telemetría de infraestructura posible, en todos los
interesados en tecnología, idealmente organizados por servicio o aplicación. En otras palabras, cuando
algo va mal con algo en nuestro entorno, necesitamos saber exactamente qué
Las aplicaciones y los servicios podrían verse o están siendo afectados. ††

En décadas pasadas, la creación de vínculos entre un servicio y la infraestructura de producción
dependía a menudo de un esfuerzo manual (como ITIL CMDB o la creación de la configuración
definiciones dentro de herramientas de alerta en herramientas como Nagios). Sin embargo, cada vez más estos enlaces
ahora están registrados automáticamente dentro de nuestros servicios, que luego son dinámicamente
descubierto y utilizado en producción a través de herramientas como Zookeeper, Etcd, Consul, etc.

Página 234

Estas herramientas permiten que los servicios se registren a sí mismos, almacenando información que otros servicios
necesita interactuar con él (por ejemplo, dirección IP, números de puerto, URI). Esto resuelve el manual
naturaleza de ITIL CMDB y es absolutamente necesario cuando los servicios están compuestos de
cientos (o miles o incluso millones) de nodos, cada uno con IP asignada dinámicamente
direcciones.‡‡

Independientemente de cuán simples o complejos sean nuestros servicios, graficar nuestras métricas comerciales
junto con nuestras métricas de aplicaciones e infraestructura nos permiten detectar cuándo van las cosas
incorrecto. Por ejemplo, podemos ver que los nuevos registros de clientes caen al 20% de las normas diarias,
e inmediatamente también veo que todas nuestras consultas a la base de datos están tardando cinco veces más
de lo normal, lo que nos permite enfocar nuestra resolución de problemas.

Además, las métricas empresariales crean un contexto para nuestras métricas de infraestructura, lo que permite
Desarrollo y operaciones para trabajar mejor juntos hacia objetivos comunes. Como Jody
Mulkey, CTO de Ticketmaster / LiveNation, observa: "En lugar de medir las operaciones
contra la cantidad de tiempo de inactividad, creo que es mucho mejor medir tanto Dev como Ops
frente a las consecuencias comerciales reales del tiempo de inactividad: cuántos ingresos deberíamos tener
logrado, pero no lo hizo ".§§

Tenga en cuenta que además de monitorear nuestros servicios de producción, también necesitamos telemetría para
esos servicios en nuestros entornos de preproducción (por ejemplo, desarrollo, prueba, puesta en escena, etc.).
Hacer esto nos permite encontrar y solucionar problemas antes de que entren en producción, como

detectar cuándo tenemos tiempos de inserción de base de datos cada vez mayores debido a un índice de tabla que falta.

REVESTIMIENTO DE OTRA INFORMACIÓN RELEVANTE EN NUESTROS MÉTRICOS

Incluso después de haber creado nuestra tubería de implementación que nos permite hacer pequeños y cambios frecuentes en la producción, los cambios siguen creando riesgos inherentemente. Efectos secundarios operacionales no son solo interrupciones, sino también interrupciones y desviaciones significativas del estándar operaciones

Para hacer visibles los cambios, hacemos visible el trabajo superponiendo toda la implementación de producción actividades en nuestros gráficos. Por ejemplo, para un servicio que maneja una gran cantidad de entrada transacciones, los cambios de producción pueden resultar en un *período de liquidación* significativo , donde el rendimiento se degrada sustancialmente a medida que fallan todas las búsquedas de caché.

Para comprender mejor y preservar la calidad del servicio, queremos entender qué tan rápido el rendimiento vuelve a la normalidad y, si es necesario, tome medidas para mejorar el rendimiento.

Del mismo modo, queremos superponer otras actividades operativas útiles, como cuando el servicio es en mantenimiento o copia de seguridad, en lugares donde es posible que deseemos mostrar o suprimir alertas.

CONCLUSIÓN

Las mejoras permitidas por la telemetría de producción de Etsy y LinkedIn nos muestran cómo Es importante ver los problemas a medida que ocurren, para que podamos buscar la causa y rápidamente remediar la situación Al tener todos los elementos de nuestro servicio emitiendo telemetría que pueden ser analizado, ya sea en nuestra aplicación, base de datos o en nuestro entorno, y haciendo que la telemetría está ampliamente disponible, podemos encontrar y solucionar problemas mucho antes de que causen algo catastrófico, idealmente mucho antes de que un cliente se dé cuenta de que algo

incorrecto. El resultado no es solo clientes más felices, sino que, al reducir la cantidad de lucha contra incendios y crisis cuando las cosas van mal, tenemos una vida más feliz y productiva lugar de trabajo con menos estrés y menores niveles de agotamiento.

† Las herramientas de ejemplo incluyen Sensu, Nagios, Zappix, LogsStash, Splunk, Sumo Logic, Datadog y Riemann.

‡ Existe una variedad de bibliotecas de registro de aplicaciones que facilitan a los desarrolladores la creación de telemetría útil, y debemos elegir una que nos permita enviar Todas nuestras aplicaciones se registran en la infraestructura de registro centralizada que creamos en la sección anterior. Los ejemplos populares incluyen rrd4j y log4j para Java, y log4r y ruby-cabin para Ruby.

§ En 2004, Gene Kim, Kevin Behr y George Spafford describieron esto como un síntoma de falta de una "cultura de causalidad", señalando que el alto rendimiento Las organizaciones reconocen que el 80% de todas las interrupciones son causadas por el cambio y el 80% del MTTR se gasta tratando de determinar qué cambió.

¶ Otro conjunto de herramientas para ayudar en el monitoreo, la agregación y la recopilación incluyen Splunk, Zabbix, Sumo Logic, DataDog, así como Nagios, Cacti, Sensu, RRDTool, Netflix Atlas, Riemann y otros. Los analistas a menudo llaman a esta amplia categoría de herramientas "monitores de rendimiento de aplicaciones".

** La creación de un panel simple debe ser parte de la creación de cualquier producto o servicio nuevo; las pruebas automatizadas deben confirmar que tanto el servicio como el panel están funcionando correctamente, ayudando tanto a nuestros clientes como a nuestra capacidad de implementar código de manera segura.

†† Exactamente como lo prescribiría una base de datos de administración de configuración de ITIL (CMDB).

‡‡ El cónsul puede ser de interés específico, ya que crea una capa de abstracción que permite fácilmente la asignación de servicios, el monitoreo, los bloqueos y la configuración de valores clave tiendas, así como la agrupación de host y la detección de fallas.

§§ Esto podría ser el costo del tiempo de inactividad de producción o los costos asociados con una función tardía. En términos de desarrollo de productos, la segunda métrica se conoce como costo de demora, y es clave para tomar decisiones efectivas de priorización.

15 Analizar Telemetría para Anticípese mejor

Problemas y lograr Metas

Como vimos en el capítulo anterior, necesitamos una producción suficiente.
telemetría en nuestras aplicaciones e infraestructura para ver y resolver problemas
a medida que ocurren. En este capítulo, crearemos herramientas que nos permitan descubrir
variaciones y señales de falla cada vez más débiles ocultas en nuestra producción
telemetría para que podamos evitar fallas catastróficas. Numerosas estadísticas
Se presentarán técnicas, junto con estudios de casos que demuestren su
utilizar.

Un gran ejemplo de análisis de telemetría para buscar y corregir de forma proactiva
Los problemas antes de que los clientes se vean afectados se pueden ver en Netflix, un sitio global
proveedor de streaming de películas y series de televisión. Netflix tuvo ingresos de
$ 6.2 mil millones de setenta y cinco millones de suscriptores en 2015. Uno de sus
El objetivo es proporcionar la mejor experiencia a quienes ven videos en línea
en todo el mundo, que requiere una entrega robusta, escalable y resistente
infraestructura. Roy Rapoport describe uno de los desafíos de administrar
el servicio de entrega de video basado en la nube de Netflix: “Dado un rebaño de ganado que
todos deberían verse y actuar de la misma manera, ¿qué ganado se ve diferente del resto?
O más concretamente, si tenemos un clúster de cómputo sin estado de mil nodos,
todos ejecutan el mismo software y están sujetos al mismo tráfico aproximado
cargar, nuestro desafío es encontrar cualquier nodo que no se parezca al resto del
nodos ".

Una de las técnicas estadísticas que el equipo utilizó en Netflix en 2012 fue
detección de valores atípicos, definida por Victoria J. Hodge y Jim Austin de la
Universidad de York como detección de "condiciones anormales de funcionamiento a partir de las cuales
puede producirse una degradación significativa del rendimiento, como un avión
defecto de rotación del motor o un problema de flujo en una tubería ".

Rapoport explica que Netflix "utilizó la detección de valores atípicos de una manera muy simple
manera, que era calcular primero cuál era la 'normalidad actual' en este momento,

población dada de nodos en un clúster de cómputo. Y luego identificamos qué nodos no se ajustaban a ese patrón y eliminaron esos nodos de producción."

Rapoport continúa: "Podemos marcar automáticamente los nodos que se comportan mal sin tener que definir realmente cuál es el comportamiento "adecuado" de ninguna manera. Y dado que estamos diseñados para ejecutarse de manera resistente en la nube, no le decimos cualquiera en Operaciones para hacer algo; en cambio, simplemente matamos a los enfermos o mal comportamiento del nodo de cómputo, y luego registrarlo o notificar a los ingenieros en cualquier forma que quieran ".

Al implementar el proceso de detección de valores atípicos del servidor, Rapoport declara, Netflix ha "reducido masivamente el esfuerzo de encontrar servidores enfermos, y, más importantemente, redujo masivamente el tiempo que requieren los estados Rapoport d para arreglar ellos, lo que resulta en una mejor calidad de servicio. El beneficio de usar estos técnicas para preservar la cordura de los empleados, el equilibrio trabajo / vida y el servicio la calidad no puede ser exagerada ". El trabajo realizado en Netflix destaca uno de manera muy específica podemos usar la telemetría para mitigar los problemas antes de que impactar a nuestro cliente.

A lo largo de este capítulo exploraremos muchas estadísticas y visualización técnicas (incluida la detección de valores atípicos) que podemos usar para analizar nuestra telemetría para anticipar mejor los problemas. Esto nos permite resolver problemas. más rápido, más barato y antes que nunca, antes de que nuestro cliente o cualquier persona en nuestra organización se ve afectada; Además, también crearemos más contexto de nuestros datos para ayudarnos a tomar mejores decisiones y lograr nuestro metas organizacionales.

UTILICE MEDIOS Y DESVIACIONES ESTÁNDAR PARA DETECTAR PROBLEMAS POTENCIALES

Una de las técnicas estadísticas más simples que podemos usar para analizar un la métrica de producción está calculando su *media* (o promedio) y *estándar desviaciones* . Al hacer esto, podemos crear un filtro que detecte cuando esto la métrica es significativamente diferente de su norma, e incluso configura nuestro alerta para que podamos tomar medidas correctivas (por ejemplo, notificar de guardia personal de producción a las 2 a.m. para investigar cuándo se realizan consultas a la base de datos significativamente más lento que el promedio).

Cuando los servicios de producción críticos tienen problemas, despertar a las personas a las 2 a.m. puede ser lo correcto. Sin embargo, cuando creamos alertas que no son procesables o son falsos positivos, hemos despertado innecesariamente personas en A mitad de la noche. Como John Vincent, uno de los primeros líderes en DevOps movimiento, observó: "La fatiga de alerta es el mayor problema que tenemos en este momento ... Necesitamos ser más inteligentes sobre nuestras alertas o todos iremos insano."

Creamos mejores alertas al aumentar la relación señal / ruido, centrándonos en las variaciones o valores atípicos que importan. Supongamos que estamos analizando el cantidad de intentos de inicio de sesión no autorizados por día. Nuestros datos recopilados tienen un Distribución gaussiana (es decir, distribución normal o curva de campana) que coincide la gráfica en la figura 29. La línea vertical en el medio de la curva de campana es la media, y las desviaciones estándar primera, segunda y tercera indicadas por las otras líneas verticales contienen 68%, 95% y 99.7% de los datos, respectivamente.

Figura 29: *Desviaciones estándar (σ) y media (μ) con distribución gaussiana (Fuente: entrada de Wikipedia "Distribución normal", https://en.wikipedia.org/wiki/Normal_distribution.)*

Un uso común de las desviaciones estándar es inspeccionar periódicamente los datos. establecer una métrica y alerta si ha variado significativamente de la media. por Por ejemplo, podemos establecer una alerta para cuando el número de inicios de sesión no autorizados los intentos por día son tres desviaciones estándar mayores que la media. Siempre que este conjunto de datos tenga distribución gaussiana, esperaríamos que solo el 0.3% de los puntos de datos dispararían la alerta.

Incluso este tipo simple de análisis estadístico es valioso, porque nadie tenía que definir un valor umbral estático, algo que no es factible si están rastreando miles o cientos de miles de métricas de producción.

Para el resto de este libro, utilizaremos los términos *telemetría, métrica,* y *conjuntos de datos* indistintamente, en otras palabras, una *métrica* (por ejemplo, "carga de página veces ") se asignará a un *conjunto de datos* (por ejemplo, 2 ms, 8 ms, 11 ms, etc.), el término utilizado por estadísticos para describir una matriz de puntos de datos donde cada columna representa una variable de la cual se realizan operaciones estadísticas.

INSTRUMENTO Y ALERTA SOBRE NO DESEADO RESULTADOS

Tom Limoncelli, coautor de *The Practice of Cloud System Administración: diseño y operación de grandes sistemas distribuidos* y un ex ingeniero de confiabilidad del sitio en Google, relata lo siguiente historia sobre monitoreo: “Cuando la gente me pide recomendaciones sobre qué para monitorear, bromeo diciendo que en un mundo ideal, eliminaríamos todas las alertas que Actualmente tenemos en nuestro sistema de monitoreo. Luego, después de cada usuario visible interrupción, nos preguntaríamos qué indicadores habrían predicho esa interrupción y luego agréguelos a nuestro sistema de monitoreo, alertando según sea necesario. Repetir. Ahora solo tenemos alertas que evitan interrupciones, en lugar de ser bombardeados por alertas después de que ya se produjo un corte de energía ".

En este paso, replicaremos los resultados de dicho ejercicio. Uno de los La forma más fácil de hacer esto es analizar nuestros incidentes más graves en los últimos pasado (por ejemplo, 30 días) y crear una lista de telemetría que podría haber habilitado detección y diagnóstico más temprano y rápido del problema, así como también más fácil y una confirmación más rápida de que se había implementado una solución efectiva.

Por ejemplo, si tuvimos un problema donde nuestro servidor web NGINX se detuvo En respuesta a las solicitudes, veríamos los principales indicadores que podrían nos advirtió anteriormente que estábamos empezando a desviarse del estándar operaciones, tales como:

Nivel de aplicación: aumento de los tiempos de carga de la página web, etc.
Nivel del sistema operativo: la memoria libre del servidor se está agotando, el espacio en disco se está agotando, etc.

Nivel de base de datos **:** los tiempos de transacción de la base de datos demoran más de normal, etc.

Nivel de red: número de servidores en funcionamiento detrás de la carga balanceo cayendo, etc.

Cada una de estas métricas es un precursor potencial de un incidente de producción. por cada uno, configuraríamos nuestros sistemas de alerta para notificarles cuando desviarse suficientemente de la media, de modo que podamos tomar medidas correctivas.

Al repetir este proceso en señales de falla cada vez más débiles, encontramos problemas cada vez más temprano en el ciclo de vida, lo que resulta en menos impacto en el cliente incidentes y casi accidentes. En otras palabras, estamos evitando problemas como además de permitir una detección y corrección más rápidas.

PROBLEMAS QUE SURGEN CUANDO NUESTRO LOS DATOS DE TELEMETRÍA TIENEN NO GAUSSIANOS DISTRIBUCIÓN

El uso de medias y desviaciones estándar para detectar la varianza puede ser extremadamente útil. Sin embargo, el uso de estas técnicas en muchos de los datos de telemetría los conjuntos que usamos en Operaciones no generarán los resultados deseados. Como el Dr. Toufic Boubez observa: “No solo recibiremos llamadas de atención a las 2 am, sino que también obtenerlos a las 2:37 am, 4:13 am, 5:17 am Esto sucede cuando el los datos subyacentes que estamos monitoreando no tienen un gaussiano distribución."

En otras palabras, cuando la distribución del conjunto de datos no tiene el Curva de campana gaussiana descrita anteriormente, las propiedades asociadas con

No se aplican las desviaciones estándar. Por ejemplo, considere el escenario en del cual estamos monitoreando la cantidad de descargas de archivos por minuto desde nuestra página web. Queremos detectar períodos cuando tenemos inusualmente altos cantidad de descargas, como cuando nuestra velocidad de descarga es mayor que tres desviaciones estándar de nuestro promedio, para que podamos proactivamente Añadir más capacidad.

La Figura 30 muestra nuestra cantidad de descargas simultáneas por minuto durante tiempo, con una barra superpuesta en la parte superior. Cuando la barra es negra, el número de descargas dentro de un período determinado (a veces llamado una "ventana deslizante") es al menos tres desviaciones estándar del promedio. De lo contrario, es gris.

Figura 30: *Descargas por minuto: alerta excesiva cuando se utiliza "3 estándar desviación "regla*
(Fuente: Dr. Toufic Boubez, "Matemáticas simples para la detección de anomalías").

El problema obvio que muestra el gráfico es que estamos alertando casi todo el tiempo. Esto es porque en casi cualquier período de tiempo dado, tenemos instancias cuando el recuento de descargas excede nuestras tres desviaciones estándar límite.

Para confirmar esto, cuando creamos un histograma (ver figura 31) que muestra el
frecuencia de descargas por minuto, podemos ver que no tiene el
forma de curva de campana clásica y simétrica. En cambio, es obvio que el
la distribución está sesgada hacia el extremo inferior, lo que muestra que la mayoría de
el tiempo tenemos muy pocas descargas por minuto pero esa descarga
los recuentos aumentan con frecuencia tres desviaciones estándar más altas.

Figura 31: *Descargas por minuto: histograma de datos que muestran datos no gaussianos distribución*
(Fuente: Dr. Toufic Boubez, "Matemáticas simples para la detección de anomalías").

Muchos conjuntos de datos de producción son de distribución no gaussiana. Dr. Nicole Forsgren explica: “En operaciones, muchos de nuestros conjuntos de datos tienen lo que nosotros llamada distribución 'chi cuadrado'. El uso de desviaciones estándar para estos datos no solo da como resultado alertas excesivas o insuficientes, pero también da como resultado un sinsentido resultados ". Ella continúa: "Cuando calculas el número de descargas que son tres desviaciones estándar por debajo de la media, terminas con un número negativo, que obviamente no tiene sentido ".

Las alertas excesivas provocan que los ingenieros de operaciones se despierten en el medio de la noche por períodos prolongados de tiempo, incluso cuando hay pocos acciones que pueden tomar apropiadamente. El problema asociado con Sub-alerta es igual de importante. Por ejemplo, supongamos que somos monitorear el número de transacciones completadas y las completadas el recuento de transacciones se reduce en un 50% en la mitad del día debido a un software falla de componentes. Si esto todavía está dentro de tres desviaciones estándar de la es decir, no se generará ninguna alerta, lo que significa que nuestros clientes descubrir el problema antes de que lo hagamos, momento en el cual el problema puede ser Mucho más difícil de resolver.

Afortunadamente, existen técnicas que podemos utilizar para detectar anomalías incluso conjuntos de datos no gaussianos, que se describen a continuación.

Otra herramienta desarrollada en Netflix para aumentar la calidad del servicio, Scryer, aborda algunas de las deficiencias de Amazon Auto Scaling (AAS), que aumenta y disminuye dinámicamente el cálculo de AWS

Página 244

El servidor cuenta según los datos de carga de trabajo. Scryer funciona prediciendo Las demandas de los clientes se basarán en patrones de uso históricos. y proporciona la capacidad necesaria.

Scryer abordó tres problemas con AAS. El primero fue tratar con picos rápidos en la demanda. Porque los tiempos de inicio de la instancia de AWS puede ser de diez a cuarenta y cinco minutos, la capacidad de cálculo adicional fue a menudo se entregan demasiado tarde para lidiar con los picos de demanda. El segundo El problema era que después de las interrupciones, la rápida disminución del cliente

la demanda llevó a AAS a eliminar demasiada capacidad de cómputo para manejar futura demanda entrante. El tercer problema fue que AAS no

factor en los patrones de tráfico de uso conocidos al programar el cálculo

capacidad.

Figura 32: *demanda de visualización de clientes de Netflix durante cinco días (Fuente: Daniel Jacobson, Danny Yuan y Neeraj Joshi, "Scryer: el auto predictivo de Netflix Motor de escala ",* The Netflix Tech Blog *, 5 de noviembre de 2013, http://techblog.netflix.com/2013/11/scryer-netflixs-predictive-auto-scaling.html .)*

Netflix aprovechó el hecho de que sus consumidores vieron

los patrones fueron sorprendentemente consistentes y predecibles, a pesar de que no

teniendo distribuciones gaussianas. A continuación se muestra un cuadro que refleja al cliente

solicitudes por segundo a lo largo de la semana laboral, mostrando regular

y patrones constantes de visualización de clientes de lunes a viernes.

Scryer utiliza una combinación de detecciones atípicas para arrojar espurias

puntos de datos y luego utiliza técnicas como Fast Fourier

Transformación (FFT) y regresión lineal para suavizar los datos mientras

preservando picos de tráfico legítimos que se repiten en sus datos. El resultado

es que Netflix puede pronosticar la demanda de tráfico con una precisión sorprendente.

Figura 33: *Netflix Scryer pronostica el tráfico de clientes y el AWS resultante calendario de recursos informáticos (Fuente: Jacobson, Yuan, Joshi, "Scryer: Motor predictivo de escalado automático de Netflix ".)*

Solo meses después de usar Scryer por primera vez en producción, Netflix mejoró significativamente su experiencia de visualización del cliente, mejoró disponibilidad del servicio y costos reducidos de Amazon EC2.

USO DE TÉCNICAS DE DETECCIÓN DE ANOMALÍA

Cuando nuestros datos no tienen distribución gaussiana, todavía podemos encontrar notables variaciones utilizando una variedad de métodos. Estas técnicas son ampliamente categorizado como *detección de anomalías, a* menudo definido como "la búsqueda de artículos o eventos que no se ajustan a un patrón esperado ". Algunos de Estas capacidades se pueden encontrar dentro de nuestras herramientas de monitoreo, mientras que otras puede requerir ayuda de personas con habilidades estadísticas.

Tarun Reddy, vicepresidente de desarrollo y operaciones de Rally Software, aboga activamente por esta colaboración activa entre Operaciones y estadísticas, observando: "Para mejorar la calidad del servicio, ponemos todo nuestro métricas de producción en Tableau, un paquete de software de análisis estadístico. Incluso tenemos un ingeniero de operaciones capacitado en estadística que escribe código R (otro paquete estadístico): este ingeniero tiene su propia cartera de pedidos, llena con solicitudes de otros equipos dentro de la empresa que desean encontrar varianza antes, antes de que cause una varianza aún mayor que podría afectar a nuestros clientes ".

Una de las técnicas estadísticas que podemos usar se llama *suavizado* , que es especialmente adecuado si nuestros datos son una serie temporal, lo que significa que cada punto de datos tiene una marca de tiempo (por ejemplo, eventos de descarga, eventos de transacciones completadas, etc.). El suavizado a menudo implica el uso de promedios móviles (o promedios móviles), que transforman nuestros datos promediando cada punto con todos los demás datos dentro de nuestra ventana corredera. Esto tiene el efecto de suavizar los cortos fluctuaciones a largo plazo y resaltando tendencias o ciclos a largo plazo. †

Un ejemplo de este efecto de suavizado se muestra en la figura 34. El negro
la línea representa los datos sin procesar, mientras que la línea azul indica los treinta días
promedio móvil (es decir, el promedio de los últimos treinta días).‡

Figura 34: *Precio de las acciones de Autodesk y filtro de media móvil de treinta días (Fuente: Jacobson, Yuan, Joshi, "Scryer: Netflix's Predictive Auto Scaling Motor.")*

Existen técnicas de filtrado más exóticas, como las transformaciones rápidas de Fourier,
que ha sido ampliamente utilizado en el procesamiento de imágenes, y el Kolmogorov-
Prueba de Smirnov (que se encuentra en Graphite y Grafana), que a menudo se usa para encontrar
similitudes o diferencias en los datos métricos periódicos / estacionales.

Podemos esperar que un gran porcentaje de telemetría con respecto a los datos del usuario
tendrá similitudes periódicas / estacionales: tráfico web, transacciones minoristas,
ver películas, y muchos otros comportamientos de los usuarios tienen muy regular y
Sorprendentemente predecibles patrones diarios, semanales y anuales. Esto nos permite
para poder detectar situaciones que varían de las normas históricas, como
cuando nuestra tasa de transacción de pedidos en un martes por la tarde cae al 50% de
Nuestras normas semanales.

Debido a la utilidad de estas técnicas en el pronóstico, podemos ser
capaz de encontrar personas en los departamentos de Marketing o Business Intelligence
con el conocimiento y las habilidades necesarias para analizar estos datos. Podemos querer
para buscar a estas personas y explorar trabajando juntos para identificar
problemas y uso de detección mejorada de anomalías y predicción de incidentes para
resuelvelos. §

> En Monitorama en 2014, el Dr. Toufic Boubez describió el poder de
> utilizando técnicas de detección de anomalías, destacando específicamente el

efectividad de la prueba de Komogorov-Smirnov, una técnica que es
a menudo se usa en estadísticas para determinar si dos conjuntos de datos difieren
significativamente y se encuentra en la popular herramienta Graphite y Grafana.
El propósito de presentar este estudio de caso aquí no es como un tutorial,
pero para demostrar cómo se puede usar una clase de técnicas estadísticas
en nuestro trabajo, así como también cómo se está utilizando en nuestras organizaciones
en aplicaciones completamente diferentes.

La Figura 35 muestra el número de transacciones por minuto en un e-
sitio de comercio. Tenga en cuenta la periodicidad semanal del gráfico, con
Volumen de transacciones cayendo los fines de semana. Por inspección visual,
podemos ver que algo peculiar parece suceder en el cuarto
semana en que el volumen normal de transacciones no vuelve a la normalidad
niveles el lunes. Esto sugiere un evento que deberíamos investigar.

Figura 35: *Volumen de transacción: sub-alerta usando "3 desviación estándar"*
regla
(Fuente: Dr. Toufic Boubez, "Matemáticas simples para la detección de anomalías").

Usar la regla de tres desviaciones estándar solo nos alertaría dos veces,
falta la caída crítica del lunes en el volumen de transacciones. Idealmente,
también nos gustaría recibir alertas de que los datos se han derivado de nuestro
patrón esperado para el lunes.

"Incluso decir 'Kolmogorov-Smirnov' es una excelente manera de impresionar
todos ", bromea el Dr. Boubez. "Pero lo que los ingenieros de Ops deberían decir
estadísticos es que este tipo de técnicas *no paramétricas* son
ideal para datos de operaciones, ya que no hace suposiciones sobre

normalidad o cualquier otra distribución de probabilidad, que es crucial para nosotros
para entender lo que está sucediendo en nuestros sistemas muy complejos. Estas
Las técnicas comparan dos distribuciones de probabilidad, lo que nos permite
comparar datos periódicos o estacionales, lo que nos ayuda a encontrar variaciones en
datos que varían de un día a otro o de una semana a otra ".

La Figura 36, en la página siguiente, muestra el mismo conjunto de datos con
se aplicó el filtro KS, con la tercera área destacando lo anómalo
Lunes donde el volumen de transacciones no volvió a los niveles normales.
Esto nos habría alertado de un problema en nuestro sistema que podría
han sido prácticamente imposibles de detectar mediante inspección visual o
utilizando desviaciones estándar. En este escenario, esta detección temprana podría
evitar un evento impactante para el cliente, así como también permitirnos
lograr nuestros objetivos organizacionales.

Figura 36: *Volumen de transacción: uso de la prueba de Kolmogorov-Smirnov para alertar sobre*

(Fuente: Dr. Toufic Boubez, "Matemáticas simples para la detección de anomalías").

CONCLUSIÓN

En este capítulo, exploramos varias técnicas estadísticas diferentes que puede usarse para analizar nuestra telemetría de producción para que podamos encontrar y solucionar problemas antes que nunca, a menudo cuando todavía son pequeños y mucho antes Causan resultados catastróficos. Esto nos permite encontrarnos cada vez más débiles. la falla indica que podemos actuar, creando un sistema cada vez más seguro de trabajo, así como aumentar nuestra capacidad para lograr nuestros objetivos.

Página 249

Se presentaron estudios de casos específicos, incluido cómo Netflix usó estos técnicas para eliminar proactivamente servidores de cómputo de producción y autoescala su infraestructura de cómputo. También discutimos cómo usar un media móvil y el filtro Kolmogorov-Smirnov, que pueden ser encontrado en las herramientas gráficas de telemetría populares.

En el próximo capítulo, describiremos cómo integrar la producción. telemetría en el trabajo diario de Desarrollo para hacer implementaciones más seguras y mejoran el sistema en su conjunto.

† El suavizado y otras técnicas estadísticas también se utilizan para manipular archivos gráficos y de audio. Por ejemplo, imagen suavizado (o desenfoque) ya que cada píxel se reemplaza por el promedio de todos sus vecinos.

‡ Otros ejemplos de filtros de suavizado incluyen promedios móviles ponderados o suavizado exponencial (que linealmente o ponderar exponencialmente los puntos de datos más recientes sobre los puntos de datos anteriores, respectivamente), y así sucesivamente.

§ Las herramientas que podemos usar para resolver este tipo de problemas incluyen Microsoft Excel (que sigue siendo una de las formas más fáciles y rápidas) para manipular datos con fines únicos), así como paquetes estadísticos como SPSS, SAS y el proyecto de código abierto R, ahora uno de los paquetes estadísticos más utilizados. Se han creado muchas otras herramientas, incluidas varias que Etsy tiene de código abierto, como Oculus, que encuentra gráficos con formas similares que pueden indicar correlación; Owewelyly, que rastrea volúmenes y frecuencias de alerta; y Skyline, que intenta identificar comportamientos anómalos en el sistema y la aplicación. gráficos

16 Habilitar comentarios para Desarrollo y Las operaciones pueden con seguridad Implementar código

En 2006, Nick Galbreath fue vicepresidente de ingeniería de Right Media, responsable de ambos los departamentos de Desarrollo y Operaciones para una plataforma de publicidad en línea que exhibió y sirvió más de diez mil millones de impresiones diarias.

Galbreath describió el panorama competitivo en el que operaban:

> En nuestro negocio, los niveles de inventario de anuncios eran extremadamente dinámicos, por lo que necesitábamos responder a las condiciones del mercado en minutos. Esto significaba que Desarrollo tenía que ser capaz de hacer cambios de código rápidamente y ponerlos en producción tan pronto como sea posible posible, de lo contrario perderíamos ante competidores más rápidos. Encontramos que tener un grupo separado para las pruebas, e incluso la implementación, fue simplemente demasiado lento. Tuvimos que

> integre todas estas funciones en un solo grupo, con responsabilidades y objetivos compartidos. Lo creas o no, nuestro mayor desafío fue lograr que los desarrolladores superaran su miedo de desplegar su propio código!

Aquí hay una ironía interesante: Dev a menudo se queja de que Ops tiene miedo de implementar código Pero en este caso, cuando se le da el poder de implementar su propio código, los desarrolladores tuvieron el mismo miedo de realizar implementaciones de código.

El miedo a implementar código que fue compartido por Dev y Ops en Right Media no es raro. Sin embargo, Galbreath observó que proporcionar más rápido y más frecuente retroalimentación a los ingenieros que realizan implementaciones (ya sea Dev u Ops), así como reduciendo el tamaño del lote de su trabajo, creó seguridad y luego confianza.

Después de observar que muchos equipos pasan por esta transformación, Galbreath describe su progresión de la siguiente manera:

> Comenzamos sin que nadie en Dev u Ops esté dispuesto a presionar el botón "desplegar código" que hemos creado que automatiza todo el proceso de implementación del código, debido a la miedo paralizante de ser la primera persona en traer potencialmente toda la producción sistemas caídos. Eventualmente, cuando alguien es lo suficientemente valiente como voluntario para impulsar código en producción, inevitablemente, debido a suposiciones o producción incorrectas sutilezas que no se apreciaron completamente, el primer despliegue de producción no funciona sin problemas, y debido a que no tenemos suficiente telemetría de producción, solo encontramos sobre los problemas cuando los clientes nos lo dicen.

Para solucionar el problema, nuestro equipo corrige urgentemente el código y lo empuja a producción, pero esta vez con más telemetría de producción agregada a nuestras aplicaciones y entorno.

Page 252

De esta manera, podemos confirmar que nuestro servicio restaurado correctamente, y estaremos capaz de detectar este tipo de problema antes de que un cliente nos lo diga la próxima vez.

Más tarde, más desarrolladores comienzan a introducir su propio código en producción. Y porque estamos trabajando en un sistema complejo, probablemente todavía romperemos algo en la producción, pero esta vez podremos ver rápidamente qué funcionalidad se rompió y decidir rápidamente ya sea para retroceder o avanzar, resolviendo el problema. Esta es una gran victoria para el todo el equipo y todos celebran, ahora estamos en racha.

Sin embargo, el equipo quiere mejorar los resultados de sus implementaciones, por lo que los desarrolladores obtenga de manera proactiva más revisiones de sus cambios de código (descritas en el capítulo 18), y todos se ayudan mutuamente a escribir mejores pruebas automatizadas para que podamos encontrar errores antes despliegue. Y porque ahora todos saben que cuanto menor es nuestra producción cambios, cuantos menos problemas tengamos, los desarrolladores comenzarán a verificar cada vez más pequeños incrementos de código más frecuentemente en la tubería de implementación, asegurando que su

El cambio funciona con éxito en la producción antes de pasar al siguiente cambio.

Ahora estamos implementando código con más frecuencia que nunca, y la estabilidad del servicio es mejor que nunca Hemos vuelto a descubrir que el secreto para un flujo suave y continuo es haciendo cambios pequeños y frecuentes que cualquiera puede inspeccionar y comprender fácilmente.

Galbreath observa que la progresión anterior beneficia a todos, incluso Desarrollo, Operaciones e Infosec. "Como la persona que también es responsable de seguridad, es tranquilizador saber que podemos implementar arreglos en la producción rápidamente, porque los cambios entran en producción durante todo el día. Además siempre me sorprende lo interesado que está cada ingeniero en la seguridad cuando encuentras problemas en su código de los que son responsables y que pueden solucionar rápidamente sí mismos."

La historia de Right Media muestra que no es suficiente simplemente automatizar la implementación proceso: también debemos integrar el monitoreo de la telemetría de producción en nuestro trabajo de despliegue, así como establecer las normas culturales de que todos son igualmente responsable de la salud de todo el flujo de valor.

En este capítulo, creamos los mecanismos de retroalimentación que nos permiten mejorar la estado del flujo de valor en cada etapa del ciclo de vida del servicio, desde el diseño del producto a través del desarrollo y despliegue y en operación y eventualmente retiro.
Al hacer esto, nos aseguramos de que nuestros servicios estén "listos para la producción", incluso lo antes posible etapas del proyecto, así como la integración de los aprendizajes de cada lanzamiento y problema de producción en nuestro trabajo futuro, lo que resulta en una mayor seguridad y productividad para todo el mundo.

UTILICE LA TELEMETRÍA PARA HACER MÁS SEGURAS LAS IMPLEMENTACIONES

En este paso, nos aseguramos de que estamos monitoreando activamente nuestra telemetría de producción cuando cualquiera realiza un despliegue de producción, como se ilustra en la historia de Right Media.
Esto permite que quien esté haciendo el despliegue, ya sea Dev u Ops, determine rápidamente si las características funcionan como se diseñó después de que se ejecuta la nueva versión

producción. Después de todo, nunca deberíamos considerar nuestra implementación o producción de código El cambio debe hacerse hasta que esté funcionando según lo diseñado en el entorno de producción.

Hacemos esto al monitorear activamente las métricas asociadas con nuestra función durante nuestro implementación para asegurarnos de que no hemos roto nuestro servicio inadvertidamente o, lo que es peor, que rompió otro servicio. Si nuestro cambio rompe o perjudica alguna funcionalidad, rápidamente trabajar para restablecer el servicio, trayendo a quien sea necesario para diagnosticar y arreglar el problema.†

Como se describe en la Parte III, nuestro objetivo es detectar errores en nuestra canalización de implementación antes entran en producción. Sin embargo, todavía habrá errores que no detectamos, y nosotros confíe en la telemetría de producción para restaurar rápidamente el servicio. Podemos elegir apagar características rotas con alternancia de características (que a menudo es la opción más fácil y menos riesgosa) ya que no implica implementaciones para la producción), ni *arregla* (es decir, crea código cambios para corregir el defecto, que luego se introducen en producción a través de canalización de implementación) o *retroceder* (por ejemplo, volver a la versión anterior mediante la función alterna o al sacar servidores rotos de la rotación usando el azul-verde o patrones de liberación canaria, etc.)

Si bien avanzar a menudo puede ser peligroso, puede ser extremadamente seguro cuando tenemos Pruebas automatizadas y procesos de implementación rápidos, y suficiente telemetría que permite nosotros para confirmar rápidamente si todo funciona correctamente en producción.

La Figura 37 muestra una implementación del cambio de código PHP en Etsy que generó un pico en Advertencias de tiempo de ejecución de PHP: en este caso, el desarrollador notó rápidamente el problema en minutos y generó una solución y la implementó en producción, resolviendo el problema en menos de diez minutos

Debido a que las implementaciones de producción son una de las principales causas de problemas de producción, cada El evento de implementación y cambio se superpone en nuestros gráficos métricos para garantizar que todos en la cadena de valor conocen la actividad relevante, lo que permite una mejor comunicación y coordinación, así como detección y recuperación más rápidas.

Figura 37: La *implementación en Etsy.com provoca advertencias en tiempo de ejecución de PHP y se corrige rápidamente (Fuente: Mike Brittain, "Seguimiento de cada lanzamiento")*

DEV COMPARTE DERECHOS DE ROTACIÓN DE PÁGINAS CON OPS

Incluso cuando nuestros despliegues y lanzamientos de producción se realizan sin problemas, en cualquier complejo servicio todavía tendremos problemas inesperados, como incidentes y cortes de energía que suceder en momentos inoportunos (todas las noches a las 2 am). Si no se fija, esto puede causar problemas recurrentes y sufrimiento para los ingenieros de Ops en el futuro, especialmente cuando estos los problemas no se hacen visibles para los ingenieros responsables de crear el problema.

Incluso si el problema resulta en un defecto asignado al equipo de funciones, puede ser priorizado por debajo de la entrega de nuevas funciones. El problema puede seguir recurriendo por semanas, meses o incluso años, causando caos e interrupción continua en las operaciones. Este es un ejemplo de cómo los centros de trabajo aguas arriba pueden optimizar localmente por sí mismos pero en realidad degrada el rendimiento de todo el flujo de valor.

Para evitar que esto suceda, haremos que todos en el flujo de valor compartan responsabilidades posteriores de manejar incidentes operacionales. Podemos hacer esto por poner a los desarrolladores, gerentes de desarrollo y arquitectos en la rotación de buscapersonas, tal como Pedro Canahuati, Director de Ingeniería de Producción de Facebook, lo hizo en 2009. Esto asegura que todos en el flujo de valor reciban comentarios viscerales sobre cualquier flujo ascendente decisiones arquitectónicas y de codificación que toman.

Al hacer esto, Operations no lucha, aislado y solo con código relacionado problemas de producción; en cambio, todos están ayudando a encontrar el equilibrio adecuado entre arreglar defectos de producción y desarrollo de nuevas funciones, independientemente de dónde residamos El flujo de valor. Como Patrick Lightbody, vicepresidente sénior de gestión de productos en New Relic, observado en 2011: “Descubrimos que cuando despertamos a los desarrolladores a las 2 am, los defectos eran arreglado más rápido que nunca ".

Un efecto secundario de esta práctica es que ayuda a la gestión del desarrollo a ver que los objetivos comerciales no se logran simplemente porque las características se han marcado como "hecho". En cambio, la función solo se realiza cuando funciona según lo diseñado en producción, sin causar escaladas excesivas o trabajo no planificado para Desarrollo o Operaciones.*

Esta práctica es igualmente aplicable para equipos orientados al mercado, responsables de ambos desarrollando la función y ejecutándola en producción y orientada funcionalmente equipos Como observó Arup Chakrabarti, Gerente de Ingeniería de Operaciones en PagerDuty durante una presentación de 2014, "Cada vez es menos común que las empresas tengan equipos de guardia dedicados; en cambio, todos los que tocan el código de producción y Se espera que los entornos sean accesibles en caso de inactividad ".

Independientemente de cómo hayamos organizado nuestros equipos, los principios subyacentes siguen siendo los mismo: cuando los desarrolladores reciben comentarios sobre cómo funcionan sus aplicaciones en la producción, lo que incluye arreglarlo cuando se rompe, se acercan al cliente, esto crea un buy-in del que se benefician todos en la cadena de valor.

HAGA QUE LOS DESARROLLADORES SIGAN EL TRABAJO ABAJO

Una de las técnicas más poderosas en interacción y diseño de experiencia de usuario (UX) es investigación contextual. Esto es cuando el equipo del producto observa a un cliente usar el aplicación en su entorno natural, a menudo trabajando en su escritorio. Haciendo tan a menudo descubre formas sorprendentes en las que los clientes luchan con la aplicación, como requerir decenas de clics para realizar tareas simples en su trabajo diario, cortar y pegar texto desde múltiples pantallas, o escribiendo notas en papel. Todos estos son ejemplos de Comportamientos compensatorios y soluciones para problemas de usabilidad.

La reacción más común para los desarrolladores después de participar en una observación del cliente. es consternación, a menudo afirmando "cuán horrible fue ver las muchas formas en que hemos estado infligiendo dolor en nuestros clientes ". Estas observaciones de los clientes casi siempre resultan en aprendizaje significativo y un ferviente deseo de mejorar la situación para el cliente.

Nuestro objetivo es utilizar esta misma técnica para observar cómo nuestro trabajo afecta nuestro trabajo interno. clientes. Los desarrolladores deben seguir su trabajo río abajo, para que puedan ver cómo los centros de trabajo posteriores deben interactuar con su producto para que funcione producción. §

Los desarrolladores quieren seguir su trabajo río abajo, al ver las dificultades del cliente de primera mano, toman decisiones mejores y más informadas en su trabajo diario.

Al hacer esto, creamos comentarios sobre los aspectos no funcionales de nuestro código, todos los elementos que no están relacionados con la función orientada al cliente e identifican formas en que puede mejorar la capacidad de implementación, la capacidad de administración, la operabilidad, etc.

La observación de UX a menudo tiene un poderoso impacto en los observadores. Al describir su primer observación del cliente, Gene Kim, el fundador y CTO de Tripwire durante trece años

y coautor de este libro, dijo:

> Uno de los peores momentos de mi carrera profesional fue en 2006 cuando pasé un
> toda la mañana viendo a uno de nuestros clientes usar nuestro producto. Lo estaba mirando
> realizar una operación que esperábamos que los clientes hicieran semanalmente y, para nuestro extremo
> horror, descubrimos que requería sesenta y tres clics. Esta persona mantuvo
> disculpándose, diciendo cosas como: "Lo siento, probablemente hay una mejor manera de hacer esto".
>
> Desafortunadamente, no había una mejor manera de hacer esa operación. Otro cliente
> describió cómo la configuración inicial del producto tomó 1.300 pasos. De repente, entendí por qué
> el trabajo de administrar nuestro producto siempre fue asignado al ingeniero más nuevo en el
> equipo: nadie quería el trabajo de ejecutar nuestro producto. Esa fue una de las razones por las que
> ayudó a crear la práctica de UX en mi empresa, para ayudar a compensar el dolor que estábamos
> infligiendo a nuestros clientes.

La observación UX permite la creación de calidad en la fuente y da como resultado una mayor
empatía por los miembros del equipo en la cadena de valor. Idealmente, la observación UX nos ayuda
a medida que creamos requisitos codificados no funcionales para agregar a nuestra cartera compartida de
trabajo, eventualmente permitiéndonos integrarlos de manera proactiva en cada servicio que construimos,
lo cual es una parte importante de la creación de una cultura de trabajo de DevOps. ¶

HAY DESARROLLADORES INICIALMENTE AUTOMÁTICOS SERVICIO DE PRODUCCION

Incluso cuando los desarrolladores escriben y ejecutan su código en producción
entornos en su trabajo diario, las operaciones aún pueden ser desastrosas
lanzamientos de producción porque es la primera vez que realmente vemos cómo se comporta nuestro código
durante un lanzamiento y bajo verdaderas condiciones de producción. Este resultado ocurre porque
Los aprendizajes operativos a menudo ocurren demasiado tarde en el ciclo de vida del software.

Si no se aborda, el resultado es a menudo un software de producción que es difícil de operar. Como
un ingeniero anónimo de Operaciones dijo una vez: "En nuestro grupo, la mayoría de los administradores de sistemas
duró solo seis meses. Las cosas siempre estaban rompiendo en producción, las horas eran
loco, y las implementaciones de aplicaciones fueron dolorosas más allá de lo imaginable: la peor parte fue
emparejar los clústeres de servidores de aplicaciones, lo que nos llevaría seis horas. Durante cada
momento, todos sentimos que los desarrolladores nos odiaban personalmente ".

Esto puede ser el resultado de no tener suficientes ingenieros de operaciones para soportar todo el producto
equipos y los servicios que ya tenemos en producción, lo que puede suceder en ambos
equipos funcionales y orientados al mercado.

Una posible contramedida es hacer lo que hace Google, que es tener Desarrollo
los grupos autogestionan sus servicios en producción antes de ser elegibles para un
Grupo Ops centralizado para gestionar. Al hacer que los desarrolladores sean responsables de la implementación
y apoyo a la producción, es mucho más probable que tengamos una transición suave a
Operaciones** **

Para evitar la posibilidad de que los servicios problemáticos y autogestionados entren en producción
y creando riesgo organizacional, podemos definir requisitos de lanzamiento que deben cumplirse
para que los servicios interactúen con clientes reales y estén expuestos a una producción real

Página 257

tráfico. Además, para ayudar a los equipos de productos, los ingenieros de Ops deben actuar como
consultores para ayudarlos a preparar sus servicios para la producción.

Al crear una guía de lanzamiento, ayudamos a garantizar que cada equipo de producto se beneficie de
Experiencia acumulativa y colectiva de toda la organización, especialmente Operaciones.
La orientación y los requisitos de lanzamiento probablemente incluirán lo siguiente:

Recuento de defectos y gravedad: ¿Realmente funciona la aplicación como se diseñó?

Tipo / frecuencia de alertas de buscapersonas: ¿La aplicación genera un mensaje no compatible?
cantidad de alertas en producción?

Cobertura de monitoreo: ¿Es la cobertura de monitoreo suficiente para restaurar el servicio?
¿cuando las cosas van mal?

Arquitectura del sistema: ¿el servicio está lo suficientemente acoplado como para soportar una tasa alta?
de cambios y despliegues en producción?

Proceso de implementación: ¿existe un método predecible, determinista y suficiente?
proceso automatizado para implementar código en producción?

Higiene de producción: ¿Hay evidencia de suficientes buenos hábitos de producción que
permitiría que el soporte de producción sea administrado por otra persona?

Superficialmente, estos requisitos pueden parecer similares a la producción tradicional.
listas de verificación que hemos usado en el pasado. Sin embargo, las diferencias clave son que requerimos
Monitoreo efectivo para estar en su lugar, implementaciones para ser confiable y determinista, y
Una arquitectura que admite implementaciones rápidas y frecuentes.

Si se encuentran deficiencias durante la revisión, el ingeniero de operaciones asignado debe ayudar
el equipo de funciones resuelve los problemas o incluso ayuda a rediseñar el servicio si es necesario, por lo que
que se puede implementar y administrar fácilmente en producción.

En este momento, también podemos querer saber si este servicio está sujeto a alguna regulación

objetivos de cumplimiento o si es probable que sea en el futuro:

¿El servicio genera una cantidad significativa de ingresos? (Por ejemplo, si es más del 5% de los ingresos totales de una corporación pública de los Estados Unidos, es un "significativo cuenta "y dentro del alcance para el cumplimiento de la Sección 404 de la Ley Sarbanes-Oxley de 2002 [SOX].)

¿El servicio tiene un alto tráfico de usuarios o tiene altos costos de interrupción / deterioro? (es decir, ¿Los problemas operativos corren el riesgo de crear disponibilidad o riesgo de reputación?)

¿El servicio almacena la información del titular de la tarjeta de pago, como la tarjeta de crédito? números o información de identificación personal, como números de Seguro Social o registros de atención al paciente? ¿Hay otros problemas de seguridad que podrían crear regulaciones, obligación contractual, privacidad o riesgo de reputación?

¿El servicio tiene algún otro requisito de cumplimiento regulatorio o contractual? asociado con él, como las regulaciones de exportación de EE. UU., PCI-DSS, HIPAA, etc.

Esta información ayuda a garantizar que gestionemos eficazmente no solo los riesgos técnicos asociado con este servicio, pero también cualquier riesgo potencial de seguridad y cumplimiento. También Proporciona información esencial en el diseño del entorno de control de producción.

Figura 38: *El "Servicio de devolución" en Google (Fuente: "SRE @ Google: Miles de DevOps Since 2004 ", video de YouTube, 45:57, publicado por USENIX, 12 de enero de 2012, https://www.youtube.com/watch?v=iIuTnhdTzK0.)*

Al integrar los requisitos de operabilidad en las primeras etapas del desarrollo

procesar y hacer que el Desarrollo inicialmente autogestione sus propias aplicaciones y servicios, el proceso de transición de nuevos servicios a producción se vuelve más fluido, cada vez más fácil y más predecible para completar. Sin embargo, para servicios que ya están en producción, necesitamos un mecanismo diferente para garantizar que las operaciones nunca se atasquen con un servicio insostenible en producción. Esto es especialmente relevante para Organizaciones de operaciones orientadas funcionalmente.

En este paso, podemos crear un *mecanismo de devolución de servicio,* en otras palabras, cuando un el servicio de producción se vuelve lo suficientemente frágil, las operaciones tienen la capacidad de regresar Responsabilidad de soporte de producción de vuelta al Desarrollo.

Cuando un servicio vuelve a un estado administrado por el desarrollador, la función de Operaciones cambia desde soporte de producción hasta consulta, ayudando al equipo a hacer el servicio Listo para la producción.

Este mecanismo sirve como nuestra válvula de escape de presión, asegurando que nunca Operaciones en una situación en la que están atrapados en la gestión de un servicio frágil mientras una cantidad cada vez mayor de deuda técnica los entierra y amplifica un problema local en un problema global. Este mecanismo también ayuda a garantizar que las operaciones tengan suficiente capacidad para trabajar en trabajos de mejora y proyectos preventivos.

La devolución sigue siendo una práctica de larga data en Google y es quizás una de las Las mejores demostraciones del respeto mutuo entre los ingenieros de Dev y Ops. Haciendo

Page 259

esto, el desarrollo es capaz de generar rápidamente nuevos servicios, con ingenieros de Ops uniéndose el equipo cuando los servicios se vuelven estratégicamente importantes para la empresa y, en raras ocasiones casos, devolviéndolos cuando se vuelven demasiado problemáticos para manejar en producción. [††] El siguiente estudio de caso de Ingeniería de confiabilidad del sitio en Google describe cómo los procesos de Revisión de preparación de traspaso y Revisión de preparación de lanzamiento evolucionado, y los beneficios que resultaron.

> Uno de los muchos hechos sorprendentes sobre Google es que tienen una funcionalidad orientación para sus ingenieros de operaciones, a los que se hace referencia como "confiabilidad del sitio Ingenieros ”(SRE), un término acuñado por Ben Treynor Sloss en 2004.[‡‡] Ese año Treynor Sloss comenzó con un personal de siete SRE que creció a más de 1,200 SRE para 2014. Como dijo Treynor Sloss, "si Google falla alguna vez, es mi culpa". Treynor Sloss se ha resistido a crear una definición de oración única de lo que son los SRE, pero él una vez describió a los SRE como "lo que sucede cuando un ingeniero de software tiene la tarea de lo que solía llamarse operaciones ".
>
> Cada SRE informa a la organización de Treynor Sloss para ayudar a garantizar la coherencia de

calidad de personal y contratación, y están integrados en equipos de productos en todo Google (que también proporcionan su financiación). Sin embargo, los SRE todavía son tan escasos que se asignan solo a los equipos de productos que tienen la mayor importancia para el empresa o aquellos que deben cumplir con los requisitos reglamentarios. Además, esos servicios deben tener una baja carga operativa. Productos que no cumplen con el Los criterios necesarios permanecen en un estado administrado por el desarrollador.

Incluso cuando los nuevos productos se vuelven lo suficientemente importantes para que la compañía garantice Al estar asignado un SRE, los desarrolladores aún deben haber autogestionado su servicio en producción durante al menos seis meses antes de que sea elegible para tener un SRE asignado al equipo.

Para ayudar a garantizar que estos equipos de productos autogestionados aún puedan beneficiarse de la experiencia colectiva de la organización SRE, Google creó dos conjuntos de seguridad comprueba dos etapas críticas de lanzamiento de nuevos servicios llamados *Lanzamiento Revisión de preparación* y *Revisión de preparación* para la *transferencia* (LRR y HRR, respectivamente).

El LRR debe realizarse y firmarse antes de que se inicie cualquier nuevo servicio de Google puesto a disposición del público para los clientes y recibe tráfico de producción en vivo, mientras la HRR se realiza cuando el servicio se pasa a un estado administrado por Ops, generalmente meses después de la LRR. Las listas de verificación LRR y HRR son similares, pero el HRR es mucho más estricto y tiene estándares de aceptación más altos, mientras que el LRR es autoinformado por los equipos de producto.

Cualquier equipo de producto que pase por un LRR o HRR tiene un SRE asignado a ellos para ayudarlos a comprender los requisitos y ayudarlos a cumplirlos requisitos Las listas de verificación de lanzamiento de LRR y HRR han evolucionado con el tiempo, por lo que cada equipo puede beneficiarse de las experiencias colectivas de todos los lanzamientos anteriores,

ya sea exitoso o no exitoso. Tom Limoncelli señaló durante su Presentación de “SRE @ Google: Miles de DevOps desde 2004” en 2012, “Cada Cuando hacemos un lanzamiento, aprendemos algo. Siempre habrá algunas personas que tienen menos experiencia que otros haciendo lanzamientos y lanzamientos. El LRR y Las listas de verificación de HRR son una forma de crear esa memoria organizacional ”.

Requerir que los equipos de productos autogestionen sus propios servicios en las fuerzas de producción. Desarrollo para caminar en la piel de Ops, pero guiado por LRR y HRR, lo que no solo hace que la transición del servicio sea más fácil y más predecible, sino que también Ayuda a crear empatía entre los centros de trabajo aguas arriba y aguas abajo.

Figura 39: *"Revisión de preparación para el lanzamiento y revisión de preparación para transferencias" en Google (Fuente: "SRE @ Google: Miles de DevOps desde 2004", video de YouTube, 45:57, publicado por USENIX, 12 de enero de 2012, https://www.youtube.com/watch?v=iIuTnhdTzK0.)*

Limoncelli señaló: "En el mejor de los casos, los equipos de productos han estado utilizando el LRR lista de verificación como guía, trabajando para cumplirla en paralelo con el desarrollo de su servicio y llegar a SRE para obtener ayuda cuando la necesiten ".

Además, Limoncelli observó: "Los equipos que tienen la HRR más rápida la aprobación de producción son las que trabajaron con SRE más temprano, desde el principio etapas de diseño hasta el lanzamiento. Y lo mejor es que siempre es fácil obtener un SRE para ser voluntario para ayudar con su proyecto. Cada SRE ve valor en dar consejos proyectar equipos temprano, y es probable que sea voluntario unas pocas horas o días para hacer solo ese."

La práctica de SRE para ayudar a los equipos de productos a tiempo es una norma cultural importante eso se refuerza continuamente en Google. Limoncelli explicó: "Producto de ayuda equipos es una inversión a largo plazo que dará sus frutos muchos meses después cuando llega el momento de lanzar. Es una forma de 'buena ciudadanía' y 'servicio comunitario' que se valora, se considera rutinariamente al evaluar ingenieros para SRE promociones ".

CONCLUSIÓN

En este capítulo, discutimos los mecanismos de retroalimentación que nos permiten mejorar nuestro servicio en cada etapa de nuestro trabajo diario, ya sea implementando cambios en producción, arreglando el código cuando las cosas van mal y los ingenieros son localizados, teniendo

los desarrolladores siguen su trabajo en sentido descendente, creando requisitos no funcionales que ayudar a los equipos de desarrollo a escribir más código listo para producción, o incluso entregar los servicios problemáticos vuelven a ser autogestionados por Desarrollo.

Al crear estos bucles de retroalimentación, hacemos que las implementaciones de producción sean más seguras, aumentamos el Disponibilidad de producción del código creado por Desarrollo, y ayuda a crear un mejor funcionamiento relación entre Desarrollo y Operaciones reforzando objetivos compartidos, responsabilidades y empatía.

En el próximo capítulo, exploramos cómo la telemetría puede permitir la hipótesis desarrollo y pruebas A / B para realizar experimentos que nos ayudan a lograr nuestro objetivos organizacionales y ganar en el mercado.

† Al hacer esto, junto con la arquitectura requerida, "optimizamos para MTTR, en lugar de MTBF", una máxima popular de DevOps para describir nuestro deseo de optimice para recuperarse de fallas rápidamente, en lugar de intentar prevenir fallas.

‡ ITIL define la garantía como cuando un servicio puede ejecutarse en producción de manera confiable sin intervención por un período de tiempo predefinido (por ejemplo, dos semanas). Esta idealmente, la definición de garantía debería integrarse en nuestra definición colectiva de "hecho".

§ Al seguir el trabajo aguas abajo, podemos descubrir formas de ayudar a mejorar el flujo, como la automatización de pasos complejos y manuales (por ejemplo, la aplicación de emparejamiento clústeres de servidores que requieren seis horas para completarse con éxito); realizar el empaquetado del código una vez en lugar de crearlo varias veces en diferentes etapas del despliegue de control de calidad y producción; trabajando con probadores para automatizar conjuntos de pruebas manuales, eliminando así un cuello de botella común para más despliegue frecuente; y crear documentación más útil en lugar de que alguien descifre las notas de la aplicación de desarrollador para compilar paquetes instaladores

¶ Más recientemente, Jeff Sussna intentó codificar aún más cómo lograr mejor los objetivos de UX en lo que él llama "conversaciones digitales", que están destinadas a Ayudar a las organizaciones a entender el viaje del cliente como un sistema complejo, ampliando el contexto de calidad. Los conceptos clave incluyen el diseño para servicio, no software; minimizando la latencia y maximizando la fuerza de la retroalimentación; diseñando para el fracaso y operando para aprender; utilizando operaciones como entrada al diseño; y buscando empatía.

** Aumentamos aún más la probabilidad de que se solucionen los problemas de producción al garantizar que los equipos de desarrollo permanezcan intactos y no se disuelvan después de que se complete el proyecto.

†† En las organizaciones con financiación basada en proyectos, puede que no haya desarrolladores a quienes devolver el servicio, ya que el equipo ya se ha disuelto o Es posible que no tenga el presupuesto o el tiempo para asumir la responsabilidad del servicio. Las posibles contramedidas incluyen realizar un bombardeo de mejora para mejorar el servicio, financiación temporal o esfuerzos de mejora de personal, o retiro del servicio.

‡‡ En este libro, utilizamos el término "ingeniero de operaciones", pero los dos términos, "ingeniero de operaciones" e "ingeniero de confiabilidad del sitio", son intercambiables.

17 Hipótesis integrada Desarrollo impulsado y pruebas A / B en nuestro Trabajo diario

Con demasiada frecuencia en proyectos de software, los desarrolladores trabajan en características durante meses o años, que abarcan varias versiones, sin confirmar si el negocio deseado los resultados se están cumpliendo, como si un particular característica es lograr los resultados deseados o incluso ser utilizado en absoluto.

Peor aún, incluso cuando descubrimos que una característica dada no es logrando los resultados deseados, haciendo correcciones a la la función puede ser superada por otras nuevas funciones, asegurando que la característica de bajo rendimiento nunca lograr su objetivo comercial previsto. En general, Jez Humilde observa, "la forma más ineficiente de probar un modelo de negocio o idea de producto es construir el completo producto para ver si la demanda prevista realmente existe ".

Antes de crear una función, debemos preguntar rigurosamente nosotros mismos, "¿Deberíamos construirlo y por qué?" Deberíamos luego realiza los experimentos más baratos y rápidos posible validar a través de la investigación del usuario si el

característica prevista realmente logrará el deseado
resultados. Podemos usar técnicas como la hipótesis
desarrollo impulsado, embudos de adquisición de clientes y
Pruebas A / B, conceptos que exploramos a lo largo de este
capítulo. Intuit, Inc. proporciona un ejemplo dramático de cómo
las organizaciones usan estas técnicas para crear productos
que los clientes adoran, para promover el aprendizaje organizacional,
y para ganar en el mercado.

Intuit se enfoca en crear negocios y finanzas
Soluciones de gestión para simplificar la vida de los pequeños.
empresas, consumidores y profesionales de la contabilidad. En
2012, tuvieron $ 4.5 mil millones en ingresos y 8,500
empleados, con productos emblemáticos que incluyen
QuickBooks, TurboTax, Mint y, hasta hace poco,
Acelerar. †

Scott Cook, el fundador de Intuit, ha defendido durante mucho tiempo
construyendo una cultura de innovación, alentando a los equipos a
adoptar un enfoque experimental para el desarrollo de productos
y exhortando a los líderes a apoyarlos. Como él dijo,
"En lugar de centrarse en el voto del jefe ... el énfasis es
en hacer que la gente real se comporte de verdad
experimentos y basar tus decisiones en eso ". Esto es
El epítome de un enfoque científico del producto
desarrollo.

Cook explicó que lo que se necesita es "un sistema donde
cada empleado puede hacer rápido, a alta velocidad

experimentos ... Dan Maurer dirige a nuestro consumidor
división ... [que] ejecuta el sitio web TurboTax. Cuando el
asumimos el control, hicimos unos siete experimentos al año ".

Página 265

Él continuó: "Al instalar una innovación desenfrenada
cultura [en 2010], ahora hacen 165 experimentos en el
tres meses de la temporada de impuestos [EE. UU.]. Resultado del negocio?
[La] tasa de conversión del sitio web ha aumentado un 50 por ciento ...
A la gente [los miembros del equipo] les encanta, porque ahora su
las ideas pueden llegar al mercado ".

Aparte del efecto en la tasa de conversión del sitio web, uno
Uno de los elementos más sorprendentes de esta historia es que
TurboTax realizó experimentos de producción durante
sus temporadas pico de tráfico. Durante décadas, especialmente en
venta minorista, el riesgo de interrupciones que afectan los ingresos durante
la temporada de vacaciones era tan alta que a menudo poníamos
en su lugar un cambio de congelación de mediados de octubre a mediados de
Enero.

Sin embargo, al realizar implementaciones y lanzamientos de software
rápido y seguro, el equipo de TurboTax se convirtió en usuario en línea
experimentación y cualquier cambio de producción requerido a
actividad de bajo riesgo que podría realizarse durante el
mayor tráfico y períodos de generación de ingresos.

Esto destaca la noción de que el período cuando
la experimentación tiene el valor más alto es durante el pico
temporadas de tráfico. ¿Había esperado el equipo TurboTax hasta

16 de abril, el día después de la fecha límite de presentación de impuestos de EE. UU., A implementar estos cambios, la empresa podría haber perdido muchos de sus posibles clientes, e incluso algunos de sus clientes existentes, a la competencia.

Cuanto más rápido podamos experimentar, iterar e integrar Comentarios sobre nuestro producto o servicio, cuanto más rápido podamos aprende y supera la competencia. Y cómo

rápidamente podemos integrar nuestros comentarios depende de nuestra capacidad de desplegar y lanzar software.

El ejemplo de Intuit muestra que el equipo Intuit TurboTax fue capaz de hacer que esta situación funcione para ellos y ganó en El mercado como resultado.

BREVE HISTORIA DE LAS PRUEBAS A / B

Como destaca la historia de Intuit TurboTax, un extremadamente potente técnica de investigación de usuarios está definiendo el embudo de adquisición de clientes y realización de pruebas A / B. Las técnicas de prueba A / B fueron pioneras en *directo marketing de respuesta* , que es uno de los dos principales categorías de estrategias de marketing. El otro se llama *marketing masivo* o *marketing de marca* y a menudo se basa en colocando tantas impresiones de anuncios delante de las personas como posible influir en las decisiones de compra.

En eras anteriores, antes del correo electrónico y las redes sociales, directo

el marketing de respuesta significaba enviar miles de
postales o volantes por correo postal, y preguntando a los prospectos
aceptar una oferta llamando a un número de teléfono,
devolver una postal o hacer un pedido.

En estas campañas, se realizaron experimentos para
determinar qué oferta tuvo las tasas de conversión más altas.
Experimentaron modificando y adaptando el
oferta, reformulación de la oferta, variación de la redacción
estilos, diseño y tipografía, empaques, etc., para
determinar cuál fue el más efectivo para generar el

acción deseada (por ejemplo, llamar a un número de teléfono, ordenar un
producto).

Cada experimento a menudo requería hacer otro diseño
e imprimir, enviar miles de ofertas, y
esperando semanas para que vuelvan las respuestas. Cada
el experimento generalmente cuesta decenas de miles de dólares por
prueba y requiere semanas o meses para completar.
Sin embargo, a pesar del gasto, las pruebas iterativas son fáciles
valió la pena si aumentó significativamente las tasas de conversión (por ejemplo,
El porcentaje de encuestados que solicitan un producto
del 3% al 12%).

Casos bien documentados de pruebas A / B incluyen campaña
recaudación de fondos, marketing en Internet y Lean Startup
metodología. Curiosamente, también ha sido utilizado por el

Gobierno británico para determinar qué letras eran más
eficaz en la recaudación de ingresos por impuestos vencidos de
ciudadanos delincuentes. ‡

INTEGRANDO PRUEBAS A / B EN NUESTRA PRUEBA DE CARACTERÍSTICAS

La técnica A / B más utilizada en UX moderna
la práctica implica un sitio web donde los visitantes son aleatorios
seleccionado para mostrar una de las dos versiones de una página, ya sea
un control (la "A") o un tratamiento (la "B"). Residencia en
análisis estadístico del comportamiento posterior de estos
dos cohortes de usuarios, demostramos si hay una
diferencia significativa en los resultados de los dos,
establecer un vínculo *causal* entre el tratamiento (p. ej., un

cambio en una característica, elemento de diseño, color de fondo)
y el resultado (por ejemplo, tasa de conversión, orden promedio
Talla).

Por ejemplo, podemos realizar un experimento para ver
ya sea modificando el texto o el color en un botón "comprar"
aumenta los ingresos o si ralentiza la respuesta
tiempo de un sitio web (al introducir un retraso artificial como
tratamiento) reduce los ingresos. Este tipo de pruebas A / B
nos permite establecer un valor en dólares en rendimiento
mejoras

A veces, las pruebas A / B también se conocen como en línea
experimentos controlados y pruebas divididas. También es posible
para ejecutar experimentos con más de una variable. Esta
nos permite ver cómo interactúan las variables, una técnica
conocido como prueba multivariante.

Los resultados de las pruebas A / B son a menudo sorprendentes. Ronny
Kohavi, ingeniero distinguido y gerente general de
el grupo de Análisis y Experimentación en Microsoft,
observó que después de "evaluar bien diseñado y
experimentos ejecutados que fueron diseñados para mejorar un
métrica clave, solo alrededor de un tercio tuvieron éxito en
¡Mejorando la métrica clave! ” En otras palabras, dos tercios de
las características tienen un impacto insignificante o realmente hacen
cosas peores Kohavi continúa señalando que todos estos
originalmente se pensaba que las características eran razonables, buenas
ideas, elevando aún más la necesidad de pruebas de usuario sobre
intuición y opiniones de expertos.

Las implicaciones de los datos de Kohavi son asombrosas. Si nosotros
no están realizando investigaciones de usuarios, lo más probable es que dos-

tercios de las características que estamos construyendo ofrecen cero o
valor *negativo* para nuestra organización, incluso mientras hacen
nuestra base de código cada vez más compleja, aumentando así nuestra
costos de mantenimiento en el tiempo y hacer nuestro software
Más difícil de cambiar. Además, el esfuerzo por construir
Estas características a menudo se realizan a expensas de la entrega

características que *aportarían* valor (es decir, costo de oportunidad). Jez Humble bromeó: "Llevado al extremo, el organización y clientes habrían estado mejor dando vacaciones a todo el equipo, en lugar de construir uno de estas características que no agregan valor ".

Nuestra contramedida es integrar las pruebas A / B en el forma en que diseñamos, implementamos, probamos e implementamos nuestras funciones. Realizar investigaciones y experimentos significativos para el usuario asegura que nuestros esfuerzos ayuden a alcanzar a nuestros clientes y objetivos organizacionales, y nos ayudan a ganar en el mercado.

INTEGRAR PRUEBAS A / B EN NUESTRO LANZAMIENTO

Las pruebas A / B rápidas e iterativas son posibles al ser capaz de hacer implementaciones de producción de manera rápida y fácil en demanda, uso de alternancia de funciones y entrega potencial múltiples versiones de nuestro código simultáneamente para el cliente segmentos Hacer esto requiere una producción útil telemetría en todos los niveles de la pila de aplicaciones.

Al conectarnos a nuestras funciones alterna, podemos controlar qué porcentaje de usuarios ve la versión de tratamiento de un experimentar. Por ejemplo, podemos tener la mitad de nuestro

los clientes sean nuestro grupo de tratamiento y la mitad obtenga

se muestra lo siguiente: "Enlace de elementos similares en no disponible artículos en el carrito ". Como parte de nuestro experimento, comparamos El comportamiento del grupo de control (sin oferta) contra el grupo de tratamiento (oferta hecha), tal vez midiendo cantidad de compras realizadas en esa sesión.

Etsy de código abierto su marco de experimentación Feature API (anteriormente conocido como Etsy A / B API), que no solo admite pruebas A / B sino también en línea rampas, lo que permite la exposición de estrangulamiento a los experimentos. Otros productos de prueba A / B incluyen Optimizely, Google Análisis, etc.

En una entrevista de 2014 con Kendrick Wang de Apptimize, Lacy Rhoades en Etsy describió su viaje:
"La experimentación en Etsy proviene de un deseo de hacer decisiones informadas, y garantizar que cuando lancemos características para nuestros millones de miembros, funcionan. También a menudo, teníamos características que llevaban mucho tiempo y tenían que mantenerse sin ninguna prueba de su éxito o cualquier popularidad entre los usuarios. Las pruebas A / B nos permiten ... decir un vale la pena trabajar en la función tan pronto como esté en marcha ".

INTEGRANDO PRUEBAS A / B EN NUESTRA CARACTERÍSTICA

Una vez que tengamos la infraestructura para admitir la función A / B lanzamiento y prueba, debemos asegurarnos de que los propietarios del producto pensar en cada característica como una hipótesis y usar nuestro lanzamientos de producción como experimentos con usuarios reales para

probar o refutar esa hipótesis. Construyendo los experimentos deben diseñarse en el contexto de la embudo de adquisición general del cliente. Barry O'Reilly, co-autor de *Lean Enterprise: cómo alto rendimiento Las organizaciones innovan a escala* , describen cómo podemos enmarcar hipótesis en el desarrollo de características en el siguiente forma:

> **Creemos** que aumentar el tamaño de las imágenes del hotel en la página de reserva
>
> **Resultará** en una mejor participación del cliente y conversión
>
> **Tendremos confianza para proceder cuando** vemos un aumento del 5% en los clientes que revisan el hotel imágenes que luego proceden a reservar en cuarenta y ocho horas

Adoptar un enfoque experimental para el producto. el desarrollo nos exige no solo desglosar el trabajo en unidades pequeñas (historias o requisitos), pero también validar si cada unidad de trabajo está entregando el resultados esperados. Si no es así, modificamos nuestro camino mapa de trabajo con caminos alternativos que realmente lograr esos resultados.

> Cuanto más rápido podamos iterar e integrar la retroalimentación en el producto o servicio que estamos entregando a

clientes, cuanto más rápido podamos aprender y más grande
El impacto que podemos crear. Cuan dramáticamente
los resultados pueden verse afectados por tiempos de ciclo más rápidos
fue evidente en Yahoo! Respuestas a medida que avanzaban
un lanzamiento cada seis semanas a lanzamientos múltiples
cada semana.

En 2009, Jim Stoneham fue Gerente General de
el Yahoo! Grupo de comunidades que incluía Flickr
y respuestas. Anteriormente, había sido principalmente
responsable de Yahoo! Respuestas, compitiendo contra
otras compañías de preguntas y respuestas como Quora, Aardvark,
y Stack Exchange.

En ese momento, las respuestas tenían aproximadamente 140
millones de visitantes mensuales, con más de veinte millones
usuarios activos que responden preguntas en más de veinte
idiomas diferentes. Sin embargo, el crecimiento de usuarios y
los ingresos se habían aplanado y la participación del usuario
los puntajes estaban disminuyendo.

Stoneham observa que "Yahoo Answers fue y
sigue siendo uno de los juegos sociales más grandes en
La Internet; decenas de millones de personas están activamente
tratando de 'subir de nivel' proporcionando respuestas de calidad a
preguntas más rápido que el próximo miembro de la
comunidad. Hubo muchas oportunidades para
ajustar la mecánica del juego, los bucles virales y otros

interacciones comunitarias. Cuando tratas con
estos comportamientos humanos, tienes que ser capaz de hacer
iteraciones rápidas y pruebas para ver qué clics con
personas."

Página 273

Él continúa: "Estos [experimentos] son las cosas
que Twitter, Facebook y Zynga hicieron tan bien.
Esas organizaciones estaban haciendo experimentos en
al menos dos veces por semana, incluso estaban revisando
los cambios que hicieron antes de sus despliegues,
para asegurarse de que todavía estaban en camino. Así que aquí estoy,
ejecutar [el] sitio de preguntas y respuestas más grande del mercado,
queriendo hacer pruebas rápidas de características iterativas, pero nosotros
no se puede liberar más rápido que una vez cada 4 semanas.
En contraste, las otras personas en el mercado tenían un
bucle de retroalimentación 10 veces más rápido que nosotros ".

Stoneham observó que tanto como producto
propietarios y desarrolladores hablan de ser métricas
impulsado, si los experimentos no se realizan con frecuencia
(diaria o semanalmente), el enfoque del trabajo diario es simplemente
en la función en la que están trabajando, en lugar de
resultados del cliente.

Como el Yahoo! El equipo de respuestas pudo pasar a
implementaciones semanales, y luego múltiples
implementaciones por semana, su capacidad de experimentar
con nuevas características aumentaron dramáticamente. Su

asombrosos logros y aprendizajes sobre el
próximos doce meses de experimentación incluidos
aumento de visitas mensuales del 72%, aumento de usuarios
compromiso de tres veces, y el equipo se duplicó
sus ingresos Para continuar su éxito, el equipo
centrado en optimizar las siguientes métricas principales:

Tiempo para la primera respuesta: qué tan rápido fue una respuesta
publicado en una pregunta del usuario?

Página 274

Tiempo para responder mejor: ¿Qué tan rápido hizo el usuario?
¿La comunidad otorga una mejor respuesta?

Votos a favor por respuesta: ¿Cuántas veces fue un
respuesta votada por la comunidad de usuarios?

Respuestas / semana / persona: cuántas respuestas fueron
usuarios creando?

Segunda tasa de búsqueda: con qué frecuencia los visitantes tenían que
buscar de nuevo para obtener una respuesta? (Más bajo es mejor.)

Stoneham concluyó: "Este fue exactamente el
aprendiendo que necesitábamos ganar en el mercado
—Y cambió más que nuestra velocidad característica.
Nos transformamos de un equipo de empleados a un
equipo de propietarios. Cuando te mueves a esa velocidad,
y están mirando los números y los resultados

diariamente, su nivel de inversión cambia radicalmente ".

CONCLUSIÓN

El éxito requiere que no solo implementemos y lancemos
software rápidamente, pero también para superar nuestro
competencia. Técnicas como la hipótesis
desarrollo, definición y medición del cliente
El embudo de adquisición y las pruebas A / B nos permiten realizar
Experimentos de usuario de forma segura y sencilla, lo que nos permite
Dé rienda suelta a la creatividad y la innovación, y cree
Aprendizaje organizacional. Y, mientras triunfar es
importante, el aprendizaje organizacional que proviene de

la experimentación también les da a los empleados la propiedad de
objetivos comerciales y satisfacción del cliente. En el
próximo capítulo, examinamos y creamos revisión y
procesos de coordinación como una forma de aumentar la calidad de
Nuestro trabajo actual.

† En 2016, Intuit vendió el negocio Quicken a la firma de capital privado HIG Capital.

‡ Hay muchas otras formas de realizar investigaciones de usuarios antes de embarcarse en el desarrollo. Entre
Los métodos más económicos incluyen realizar encuestas, crear prototipos (ya sea simulacros
ups utilizando herramientas como Balsamiq o versiones interactivas escritas en código), y realizando
pruebas de usabilidad. Alberto Savoia, Director de Ingeniería de Google, acuñó el término.
pretotipado para la práctica de usar prototipos para validar si estamos construyendo lo correcto
cosa. La investigación del usuario es tan económica y fácil en relación con el esfuerzo y el costo de construir un
característica inútil en el código que, en casi todos los casos, no deberíamos priorizar una característica sin
alguna forma de validación

18 Crear revisión y Procesos de Coordinación para aumentar la calidad de nuestro Trabajo actual

En los capítulos anteriores, creamos la telemetría necesaria para ver y resolver problemas en producción y en todas las etapas de nuestra tubería de implementación, y creamos circuitos de retroalimentación rápidos

de los clientes para ayudar a mejorar el aprendizaje organizacional, aprendizaje que fomenta
propiedad y responsabilidad por la satisfacción del cliente y el rendimiento de las funciones, que
nos ayuda a tener éxito

Nuestro objetivo en este capítulo es permitir que Desarrollo y Operaciones reduzcan el riesgo de
cambios de producción antes de que se realicen. Tradicionalmente, cuando revisamos los cambios para
implementación, tendemos a depender en gran medida de las revisiones, inspecciones y aprobaciones justo antes de
despliegue. Con frecuencia, esas aprobaciones son otorgadas por equipos externos que a menudo están demasiado lejos
eliminado del trabajo para tomar decisiones informadas sobre si un cambio es arriesgado o no,
y el tiempo requerido para obtener todas las aprobaciones necesarias también alarga nuestro liderazgo de cambio
veces.

El proceso de revisión por pares en GitHub es un ejemplo sorprendente de cómo puede aumentar la inspección
calidad, hacer que las implementaciones sean seguras e integrarse en el flujo del trabajo diario de todos.
Fueron pioneros en el proceso llamado *solicitud de extracción* , una de las formas más populares de pares
revisar que abarcan Dev y Ops.

Scott Chacon, CIO y cofundador de GitHub, escribió en su sitio web que las solicitudes de extracción son
El mecanismo que permite a los ingenieros contarles a otros sobre los cambios que han llevado a un
repositorio en GitHub. Una vez que se envía una solicitud de extracción, las partes interesadas pueden revisar el conjunto de
cambios, discutir posibles modificaciones e incluso impulsar las confirmaciones de seguimiento si es necesario.
Los ingenieros que envían una solicitud de extracción a menudo solicitarán un "+1", "+2", etc., dependiendo
sobre cuántas revisiones necesitan, o ingenieros de "@mention" que les gustaría recibir críticas
desde.

En GitHub, las solicitudes de extracción también son el mecanismo utilizado para implementar código en producción
a través de un conjunto colectivo de prácticas que llaman "GitHub Flow", así es como solicitan los ingenieros
revisiones de código, recopilar e integrar comentarios, y anunciar que el código se implementará en
producción (es decir, rama "maestra").

Figura 40: *Comentarios y sugerencias sobre una solicitud de extracción de GitHub (Fuente: Scott Chacon, "GitHub Flow", ScottChacon.com, 31 de agosto de 2011, http://scottchacon.com/2011/08/31/github-flow.html.)*

GitHub Flow se compone de cinco pasos:

1. Para trabajar en algo nuevo, el ingeniero crea una rama descriptiva a partir de master (por ejemplo, "new-oauth2-scopes").
2. El ingeniero se compromete a esa rama localmente, empujando regularmente su trabajo a la misma rama nombrada en el servidor.
3. Cuando necesitan retroalimentación o ayuda, o cuando piensan que la sucursal está lista para fusionarse, abren una solicitud de extracción.
4. Cuando obtienen sus revisiones deseadas y obtienen las aprobaciones necesarias de la función, el El ingeniero puede fusionarlo en maestro.
5. Una vez que los cambios de código se fusionan y se envían al maestro, el ingeniero los implementa en producción

Estas prácticas, que integran la revisión y la coordinación en el trabajo diario, han permitido GitHub ofrece funciones de forma rápida y confiable al mercado con alta calidad y seguridad. Por ejemplo, en 2012 realizaron una asombrosa cantidad de 12,602 implementaciones. En particular, en 23 de agosto, después de una cumbre de toda la compañía donde se intercambiaron ideas interesantes y discutió, la compañía tuvo su día de despliegue más ocupado del año, con 563 construcciones y 175 implementaciones exitosas en producción, todo hecho posible a través de la solicitud de extracción proceso.

A lo largo de este capítulo integraremos prácticas, como las utilizadas en GitHub, para cambiar nuestra dependencia de inspecciones y aprobaciones periódicas, y pasar a pares integrados revisión realizada continuamente como parte de nuestro trabajo diario. Nuestro objetivo es asegurar que Development, Operations e Infosec colaboran continuamente para que cambiemos hacer a nuestros sistemas funcionará de manera confiable, segura, segura y según lo diseñado.

LOS PELIGROS DE LOS PROCESOS DE APROBACIÓN DEL CAMBIO

La falla de Knight Capital es uno de los errores de implementación de software más destacados en memoria reciente Un error de implementación de quince minutos resultó en una pérdida comercial de $ 440 millones, durante el cual los equipos de ingeniería no pudieron desactivar los servicios de producción. los las pérdidas financieras pusieron en peligro las operaciones de la empresa y obligaron a la empresa a ser vendida el fin de semana para que puedan continuar operando sin poner en peligro todo el financiero sistema.

John Allspaw observó que cuando ocurren incidentes de alto perfil, como Knight Capital accidente de despliegue, generalmente hay dos narrativas *contrafácticas de* por qué el accidente ocurrió.†

La primera narrativa es que el accidente se debió a una falla en el control de cambios, que parece válido porque podemos imaginar una situación en la que podrían haber mejores prácticas de control de cambios detectó el riesgo antes y evitó que el cambio entrara en producción. Y si nosotros no pudimos evitarlo, podríamos haber tomado medidas para permitir una detección y recuperación más rápidas.

La segunda narrativa es que el accidente se debió a una falla en la prueba. Esto también parece válido, con mejores prácticas de prueba podríamos haber identificado el riesgo antes y cancelarlo implementación, o al menos podríamos haber tomado medidas para permitir una detección y recuperación más rápidas.

La sorprendente realidad es que en entornos con poca confianza, comando y control culturas, los resultados de este tipo de control de cambios y pruebas de contramedidas a menudo dar como resultado una mayor probabilidad de que los problemas vuelvan a ocurrir, potencialmente incluso con peores resultados

Gene Kim (coautor de este libro) describe su comprensión de que el cambio y las pruebas los controles pueden tener el efecto contrario al que se pretende como "uno de los más momentos importantes de mi carrera profesional. Este momento 'aha' fue el resultado de un conversación en 2013 con John Allspaw y Jez Humble sobre Knight Capital accidente, haciéndome cuestionar algunas de mis creencias centrales que he formado en los últimos diez años, especialmente haber sido entrenado como auditor ".

Él continúa: “Por muy molesto que fuera, también fue un momento muy formativo para mí. No solo me convencieron de que estaban en lo correcto, probamos estas creencias en el *Estado de 2014 del Informe DevOps* , que condujo a algunos hallazgos sorprendentes que refuerzan ese edificio Es probable que las culturas de alta confianza sean el mayor desafío de gestión de esta década ".

PELIGROS POTENCIALES DE "CONTROLAR DEMASIADO CAMBIOS "

Los controles de cambio tradicionales pueden conducir a resultados no deseados, como contribuir a tiempos de entrega y reducción de la fuerza y la inmediatez de los comentarios de la implementación proceso. Para entender cómo sucede esto, examinemos los controles que ponemos a menudo en su lugar cuando ocurren fallas en el control de cambios:

- Agregar más preguntas que deben responderse al formulario de solicitud de cambio

Requerir más autorizaciones, como un nivel más de aprobación de gestión (por ejemplo, en lugar de simplemente la aprobación del vicepresidente de operaciones, ahora requerimos que el CIO también aprobar) o más partes interesadas (por ejemplo, ingeniería de redes, juntas de revisión de arquitectura, etc.)

Requerir más tiempo de espera para las aprobaciones de cambio para que las solicitudes de cambio puedan ser adecuadamente evaluado

Estos controles a menudo agregan más fricción al proceso de implementación al multiplicar cantidad de pasos y aprobaciones, y el aumento de los tamaños de lote y los plazos de implementación, que sabemos reduce la probabilidad de resultados de producción exitosos tanto para Dev como para Ops Estos controles también reducen la rapidez con la que recibimos comentarios de nuestro trabajo.

Una de las creencias centrales del sistema de producción de Toyota es que "las personas más cercanas a un problema normalmente saben más al respecto ". Esto se vuelve más pronunciado a medida que el trabajo es realizado y el sistema en el que se produce el trabajo se vuelve más complejo y dinámico, como es típico en las secuencias de valor de DevOps. En estos casos, crear pasos de aprobación de personas que están ubicados cada vez más lejos del trabajo, en realidad pueden reducir la probabilidad de éxito. Como se ha demostrado una y otra vez, cuanto mayor sea la distancia entre la persona haciendo el trabajo (es decir, el implementador del cambio) y la persona que decide hacer el trabajo (es decir, el autorizador de cambios), peor es el resultado.

En el *Informe sobre el estado de DevOps 2014 de* Puppet Labs , uno de los hallazgos clave fue que Las organizaciones ejecutoras confiaron más en la revisión por pares y menos en la aprobación externa de cambios La Figura 41 muestra que mientras más organizaciones confían en las aprobaciones de cambios, peor su rendimiento de TI en términos de estabilidad (tiempo medio para restaurar el servicio y cambiar tasa de falla) y rendimiento (tiempos de implementación, frecuencia de implementación).

En muchas organizaciones, las juntas asesoras de cambio desempeñan un papel importante en la coordinación y gobernar el proceso de entrega, pero su trabajo no debe ser evaluar manualmente cada cambio, ni ITIL exige tal práctica.

Para entender por qué este es el caso, considere la situación de estar en un aviso de cambio junta, revisando un cambio complejo compuesto por cientos de miles de líneas de código cambios, y creado por cientos de ingenieros.

En un extremo, no podemos predecir de manera confiable si un cambio será exitoso ya sea por leer una descripción de cien palabras del cambio o simplemente validar que una lista de verificación se ha completado. En el otro extremo, escrutando dolorosamente miles de líneas de código. Es poco probable que los cambios revelen nuevas ideas. Parte de esto es la naturaleza de hacer cambios dentro de un sistema complejo. Incluso los ingenieros que trabajan dentro de la base de código como parte de su trabajo diario a menudo se sorprende con los efectos secundarios de lo que deberían ser cambios de bajo riesgo.

Figura 41: Las *organizaciones que dependen de la revisión por pares superan a las que tienen aprobaciones de cambio*
(Fuente: Puppet Labs, DevOps Survey Of Practice 2014 *)*

Por todas estas razones, necesitamos crear prácticas de control efectivas que más de cerca
se asemejan a la revisión por pares, lo que reduce nuestra dependencia de organismos externos para autorizar nuestros cambios.
También necesitamos coordinar y programar cambios de manera efectiva. Exploramos ambos en
Las siguientes dos secciones.

HABILITAR COORDINACIÓN Y PROGRAMACIÓN DE CAMBIOS

Cada vez que tenemos múltiples grupos trabajando en sistemas que comparten dependencias, nuestro
Es probable que sea necesario coordinar los cambios para garantizar que no interfieran con cada uno
otro (p. ej., cálculo de referencias, procesamiento por lotes y secuenciación de los cambios). En general, cuanto más
acoplada libremente a nuestra arquitectura, menos necesitamos comunicarnos y coordinarnos con
otros equipos componentes: cuando la arquitectura está realmente orientada al servicio, los equipos pueden hacer
cambios con un alto grado de autonomía, donde es poco probable que los cambios locales creen global
interrupciones

Sin embargo, incluso en una arquitectura débilmente acoplada, cuando muchos equipos están haciendo cientos de
implementaciones independientes por día, puede existir el riesgo de que los cambios interfieran con cada

otros (p. ej., pruebas simultáneas A / B). Para mitigar estos riesgos, podemos usar salas de chat para anunciar cambios y buscar de manera proactiva colisiones que puedan existir.

Para organizaciones más complejas y organizaciones con una relación más estrecha arquitecturas, es posible que necesitemos programar deliberadamente nuestros cambios, donde los representantes de los equipos se reúnen, no para autorizar cambios, sino para programar y secuenciar sus cambios para minimizar accidentes.

Sin embargo, ciertas áreas, como los cambios de infraestructura global (p. Ej., Conmutador de red central cambios) siempre tendrán un mayor riesgo asociado con ellos. Estos cambios siempre requieren contramedidas técnicas, como redundancia, conmutación por error, pruebas exhaustivas, y (idealmente) simulación.

Página 282

HABILITAR REVISIÓN DE CAMBIOS

En lugar de requerir la aprobación de un organismo externo antes del despliegue, podemos requerir ingenieros para obtener revisiones por pares de sus cambios. En desarrollo, esta práctica ha sido llamado *revisión de código* , pero es igualmente aplicable a cualquier cambio que realicemos en nuestras aplicaciones o entornos, incluidos servidores, redes y bases de datos. [1] El objetivo es encontrar errores haciendo que otros ingenieros cercanos al trabajo analicen nuestros cambios. Esta revisión mejora la calidad de nuestros cambios, que también crea los beneficios de la capacitación cruzada, el aprendizaje entre pares, y mejora de habilidades.

Un lugar lógico para requerir revisiones es antes de enviar el código al tronco en el control de origen, donde los cambios podrían tener un impacto global o en todo el equipo. Como mínimo, compañero los ingenieros deben revisar nuestro cambio, pero para áreas de mayor riesgo, como cambios en la base de datos o componentes críticos para el negocio con poca cobertura de prueba automatizada, es posible que necesitemos más revisión de un experto en la materia (por ejemplo, ingeniero de seguridad de la información, base de datos ingeniero) o varias revisiones (por ejemplo, "+2" en lugar de simplemente "+1").

El principio de los tamaños de lotes pequeños también se aplica a las revisiones de código. Cuanto mayor sea el tamaño de la cambio que necesita ser revisado, cuanto más se tarde en comprender y cuanto mayor sea el carga para el ingeniero de revisión. Como observó Randy Shoup: "Hay una no lineal relación entre el tamaño del cambio y el riesgo potencial de integrar ese cambio —Cuando pasa de un cambio de código de diez líneas a un código de cien líneas, el riesgo de algo que sale mal es más de diez veces mayor, y así sucesivamente ". Por eso es tan esencial para que los desarrolladores trabajen en pasos pequeños e incrementales en lugar de hacerlo de larga duración ramas características.

Además, nuestra capacidad de criticar significativamente los cambios en el código disminuye a medida que el cambio el tamaño sube Como Giray Özil tuiteó: "Pídale a un programador que revise diez líneas de código, él Encuentra diez problemas. Pídele que haga quinientas líneas, y él dirá que se ve bien.

Las pautas para las revisiones de código incluyen:

Todos deben tener a alguien que revise sus cambios (por ejemplo, el código, el entorno, etc.) antes de comprometerse con el tronco.

Todos deben monitorear el flujo de confirmación de sus compañeros de equipo para que Los conflictos potenciales pueden ser identificados y revisados.

Defina qué cambios califican como de alto riesgo y pueden requerir la revisión de un designado experto en la materia (p. ej., cambios en la base de datos, módulos sensibles a la seguridad como autenticación, etc.).§

Si alguien envía un cambio que es demasiado grande para razonarlo fácilmente, en otras palabras, no puede entender su impacto después de leerlo un par de veces, o necesita pedirle aclaraciones al remitente; debe dividirse en múltiples, más pequeños cambios que se pueden entender de un vistazo.

Para asegurarnos de que no solo seamos revisiones de sellos de goma, también podemos inspeccionar estadísticas de revisión de código para determinar el número de cambios propuestos aprobados versus no aprobado, y tal vez muestrear e inspeccionar revisiones de códigos específicos.

Page 283

Las revisiones de código vienen en varias formas:

Programación en pareja: los programadores trabajan en parejas (versección a continuación)

"Sobre el hombro": un desarrollador mira por encima del hombro del autor como este último camina a través del código.

Transmisión de correo electrónico: un sistema de administración de código fuente envía el código por correo electrónico a los revisores automáticamente después de que se registre el código.

Revisión de código asistida por herramientas: los autores y revisores utilizan herramientas especializadas diseñadas para revisión del código de pares (por ejemplo, Gerrit, solicitudes de extracción de GitHub, etc.) o instalaciones proporcionadas por el repositorios de código fuente (por ejemplo, GitHub, Mercurial, Subversion, así como otras plataformas como Gerrit, Atlassian Stash y Atlassian Crucible).

El escrutinio minucioso de los cambios en muchas formas es efectivo para localizar errores previamente pasado por alto. Las revisiones de código pueden facilitar el aumento de los compromisos de código y la producción implementaciones y admite implementación basada en troncales y entrega continua a escala, a medida que veremos en el siguiente caso de estudio.

Google es un excelente ejemplo de una empresa que emplea empleados basados en troncales desarrollo y entrega continua a escala. Como se señaló anteriormente en este libro, Eran

Messeri describió que en 2013 los procesos en Google permitieron más de trece
miles de desarrolladores trabajarán desde el tronco en un solo árbol de código fuente, realizando
más de 5,500 confirmaciones de código por semana, lo que resulta en cientos de implementaciones de producción
por semana. En 2010, se registraron más de 20 cambios en el tronco cada minuto,
dando como resultado que el 50% de la base de código se cambie cada mes.

Esto requiere una considerable disciplina por parte de los miembros del equipo de Google y obligatorio
revisiones de código, que cubren las siguientes áreas:

Legibilidad de código para idiomas (aplica la guía de estilo)

Asignaciones de propiedad de subárboles de código para mantener la coherencia y la corrección.

Transparencia de código y contribuciones de código entre equipos

La Figura 42 muestra cómo los tiempos de revisión de código se ven afectados por el tamaño del cambio. Sobre el
el eje x es el tamaño del cambio, y en el eje y es el tiempo de entrega requerido para el código
proceso de revisión. En general, cuanto mayor sea el cambio presentado para las revisiones de código, el
más tiempo de espera requerido para obtener las aprobaciones necesarias. Y los datos apuntan en
la esquina superior izquierda representa los cambios más complejos y potencialmente riesgosos que
requirió más deliberación y discusión.

Figura 42: *Tamaño del cambio frente al tiempo de entrega de las revisiones en Google (Fuente: Ashish Kumar, "Desarrollo a la velocidad y escala de Google", presentación en QCon, San Francisco, CA, 2010, https://qconsf.com/sf2010/dl/qcon-sanfran-2010 / slides / AshishKumar_DevelopingProductsattheSpeedandScaleofGoogle.pdf .)*

Mientras trabajaba como director de ingeniería de Google, Randy Shoup comenzó un proyecto personal para resolver un problema técnico que enfrentaba la organización. Él dijo: "Trabajé en ese proyecto durante semanas y finalmente pude preguntar un tema experto en la materia para revisar mi código. Eran casi tres mil líneas de código, que Le tomó al revisor días de trabajo. Él me dijo: 'Por favor no me hagas eso de nuevo.' Estaba agradecido de que este ingeniero se tomó el tiempo para hacer eso. Eso fue también cuando yo aprendí a hacer que las revisiones de código sean parte de mi trabajo diario ".

PELIGROS POTENCIALES DE HACER MÁS PRUEBAS MANUALES Y CAMBIE CONGELAS

Ahora que hemos creado revisiones por pares que reducen nuestro riesgo, acortan los plazos de entrega asociados con procesos de aprobación de cambios y permitir la entrega continua a escala, como vimos en En el estudio de caso de Google, examinemos los efectos de cómo las pruebas de contramedidas pueden a veces es contraproducente. Cuando ocurren fallas en las pruebas, nuestra reacción típica es hacer más pruebas. Sin embargo, si simplemente estamos realizando más pruebas al final del proyecto, podemos empeorar nuestros resultados.

Esto es especialmente cierto si estamos haciendo pruebas manuales, porque las pruebas manuales son naturalmente más lento y más tedioso que las pruebas automatizadas y realizar "pruebas adicionales" a menudo tiene la consecuencia de tomar mucho más tiempo para probar, lo que significa que estamos desplegando con menos frecuencia, lo que aumenta el tamaño de nuestro lote de implementación. Y sabemos por ambas teorías y practique que cuando aumentamos el tamaño de nuestro lote de implementación, nuestras tasas de éxito de cambio se van hacia abajo y nuestro incidente cuenta y el MTTR sube: lo contrario del resultado que queremos.

En lugar de realizar pruebas en grandes lotes de cambios programados alrededor cambiar los períodos de congelación, queremos integrar completamente las pruebas de nuestro trabajo diario como parte de la flujo continuo y continuo hacia la producción, y aumenta nuestra frecuencia de implementación. Por Al hacer esto, creamos calidad, lo que nos permite probar, implementar y lanzar productos cada vez más pequeños. Tamaños de lote.

HABILITE LA PROGRAMACIÓN DE PARES PARA MEJORAR TODOS NUESTROS CAMBIOS

La programación por pares es cuando dos ingenieros trabajan juntos en la misma estación de trabajo, un método popularizado por Extreme Programming and Agile a principios de la década de 2000. Como con el código revisiones, esta práctica comenzó en Desarrollo pero es igualmente aplicable al trabajo que cualquier El ingeniero lo hace en nuestro flujo de valor. En este libro, usaremos el término *emparejamiento* y *emparejamiento programación de manera* intercambiable, para indicar que la práctica no es solo para desarrolladores.

En un patrón común de emparejamiento, un ingeniero desempeña el papel del *conductor* , la persona que

en realidad escribe el código, mientras que el otro ingeniero actúa como *navegador, observador* o *puntero* , la persona que revisa el trabajo a medida que se realiza. Mientras revisa, el el observador también puede considerar la dirección estratégica del trabajo, proponiendo ideas para mejoras y posibles problemas futuros para abordar. Esto libera al conductor para enfocar todo su o su atención en los aspectos tácticos de completar la tarea, utilizando al observador como seguridad red y guia. Cuando los dos tienen especialidades diferentes, las habilidades se transfieren como un efecto secundario automático, ya sea a través de capacitación ad-hoc o compartiendo técnicas y soluciones alternativas.

Otro patrón de programación de pares refuerza el desarrollo basado en pruebas (TDD) al tener un ingeniero escribe la prueba automatizada y el otro ingeniero implementa el código. Jeff Atwood, uno de los fundadores de Stack Exchange, escribió: "No puedo evitar preguntarme si la pareja la programación no es más que una revisión del código de los esteroides ... La ventaja del par la programación es su inmediatez apasionante: es imposible ignorar al revisor cuando él o ella está sentada a tu lado ".

Continuó: “La mayoría de las personas optarán pasivamente [por revisar el código] si se les da la opción. Con la programación de pares, eso no es posible. Cada mitad del par *tiene* que entender el código, en ese mismo momento, como se está escribiendo. El emparejamiento puede ser invasivo, pero también puede forzar un nivel de comunicación que de otro modo nunca alcanzarías ".

La Dra. Laurie Williams realizó un estudio en 2001 que mostró que "los programadores emparejados son 15% más lento que dos programadores individuales independientes, mientras que el código 'sin errores' aumentó del 70% al 85%. Dado que las pruebas y la depuración suelen ser muchas veces más costosas que programación inicial, este es un resultado impresionante. Los pares generalmente consideran más diseño alternativas que los programadores que trabajan solos y llegan a una forma más simple y más fácil de mantener diseños; también detectan defectos de diseño temprano ". La Dra. Williams también informó que el 96% de ella Los encuestados declararon que disfrutaban más de su trabajo cuando programaban en parejas que cuando programaron solos.¶

La programación en pareja tiene el beneficio adicional de difundir el conocimiento a lo largo del organización y aumento del flujo de información dentro del equipo. Tener más experiencia los ingenieros revisan mientras que los códigos de ingenieros menos experimentados también son una forma efectiva de enseñar y ser enseñado

Elisabeth Hendrickson, vicepresidenta de ingeniería de Pivotal Software, Inc. y autora de *¡Explora! Reduzca el riesgo y aumente la confianza con las pruebas exploratorias* , tiene

hablado ampliamente acerca de hacer que cada equipo sea responsable de su propia calidad, como en oposición a hacer responsables departamentos separados. Ella argumenta que no hacerlo solo aumenta la calidad, pero aumenta significativamente el flujo de trabajo hacia la producción.

En su presentación de la Cumbre de DevOps Enterprise 2015, describió cómo en 2011, hubo dos métodos aceptados de revisión de código en Pivotal: programación de pares (que se aseguró de que cada línea de código fuera inspeccionada por dos personas) o una revisión de código proceso que fue administrado por Gerrit (que aseguró que cada confirmación de código tuviera dos las personas designadas "+1" el cambio antes de que se permitiera en el tronco).

El problema que observó Hendrickson con el proceso de revisión del código de Gerrit fue que A menudo, los desarrolladores tardarían una semana completa en recibir las revisiones requeridas. Peor aún, los desarrolladores calificados estaban experimentando el "frustrante y aplastante alma". experiencia de no poder obtener cambios simples en la base de código, porque nosotros inadvertidamente creó cuellos de botella intolerables ".

Hendrickson lamentó que "las únicas personas que tenían la capacidad de '+1' los cambios eran ingenieros superiores, que tenían muchas otras responsabilidades y que a menudo no les importaba mucho sobre las soluciones en las que estaban trabajando los desarrolladores junior o sus productividad. Creó una situación terrible, mientras esperabas tus cambios para Al ser revisados, otros desarrolladores estaban revisando sus cambios. Entonces por una semana, tú tendría que fusionar todos sus cambios de código en su computadora portátil, volver a ejecutar todas las pruebas para asegúrese de que todo funcionó, y (a veces) tendría que volver a enviar su cambios para revisión de nuevo! "

Para solucionar el problema y eliminar todos estos retrasos, terminaron desmantelando el proceso completo de revisión de código de Gerrit, en lugar de eso requiere programación de pares para implementar El código cambia al sistema. Al hacer esto, redujeron la cantidad de tiempo requerido para obtener el código revisado de semanas a horas.

Hendrickson se da cuenta rápidamente de que las revisiones de código funcionan bien en muchas organizaciones, pero requiere una cultura que valore la revisión del código tanto como valora escribir el código en El primer lugar. Cuando esa cultura aún no está en su lugar, la programación de pares puede servir como un valiosa práctica interina.

EVALUACIÓN DE LA EFICACIA DE LOS PROCESOS DE SOLICITUD DE TIRO

Debido a que el proceso de revisión por pares es una parte importante de nuestro entorno de control, necesitamos para poder determinar si funciona de manera efectiva o no. Un método es mirar interrupciones de producción y examinar el proceso de revisión por pares para cualquier cambio relevante.

Otro método proviene de Ryan Tomayko, CIO y cofundador de GitHub y uno de los inventores del proceso de solicitud de extracción. Cuando se le pide que describa la diferencia entre un mal solicitud de extracción y una buena solicitud de extracción, dijo que tiene poco que ver con el resultado de la producción. En cambio, una solicitud de extracción incorrecta es aquella que no tiene suficiente contexto para el lector, ya que tiene poca o ninguna documentación de lo que se pretende hacer con el cambio. Por ejemplo, una solicitud de extracción que simplemente tiene el siguiente texto: "Arreglando el problema # 3616 y # 3841". ** **

Esa fue una solicitud de extracción interna real de GitHub, que Tomayko criticó: "Esto fue probablemente escrito por un nuevo ingeniero aquí. En primer lugar, no hubo ingenieros específicos específicamente @ mencionado: como mínimo, el ingeniero debería haber mencionado a su mentor o un experto en la materia en el área que están modificando para asegurarse de que alguien

Revisiones apropiadas de su cambio. Peor aún, no hay ninguna explicación de cuáles son los cambios.
en realidad lo son, por qué es importante, o exponer cualquiera de los pensamientos del implementador ".

Por otro lado, cuando se le pide que describa una gran solicitud de extracción que indica una efectiva
proceso de revisión, Tomayko enumeró rápidamente los elementos esenciales: debe haber suficiente
detalle sobre por qué se está realizando el cambio, cómo se realizó el cambio, así como cualquier identificación
riesgos y contramedidas resultantes.

Tomayko también busca una buena discusión sobre el cambio, habilitado por todo el contexto que el
solicitud de extracción proporcionada: a menudo, habrá riesgos adicionales señalados, ideas sobre mejores
formas de implementar el cambio deseado, ideas sobre cómo mitigar mejor el riesgo, y así
adelante. Y si sucede algo malo o inesperado durante el despliegue, se agrega al
solicitud de extracción, con un enlace al problema correspondiente. Toda discusión ocurre sin colocar
culpa; en cambio, hay una conversación sincera sobre cómo evitar que el problema
recurrente en el futuro.

Como ejemplo, Tomayko produjo otra solicitud de extracción interna de GitHub para una base de datos
migración. Tenía muchas páginas, con largas discusiones sobre los riesgos potenciales,
que lleva a la siguiente declaración del autor de la solicitud de extracción: “Estoy presionando esto ahora.
Las compilaciones ahora fallan para la rama debido a una columna que falta en los servidores de CI. (Enlace
a Post-Mortem: corte de MySQL) "

El remitente del cambio se disculpó por la interrupción, describiendo qué condiciones y
supuestos erróneos llevaron al accidente, así como una lista de contramedidas propuestas para
evitar que vuelva a suceder. Esto fue seguido por páginas y páginas de discusión.
Leyendo la solicitud de extracción, Tomayko sonrió, "Ahora *que* es una gran solicitud de extracción".

Como se describió anteriormente, podemos evaluar la efectividad de nuestro proceso de revisión por pares
muestreo y examen de solicitudes de extracción, ya sea de toda la población de solicitudes de extracción o
aquellos que son relevantes para incidentes de producción.

CORTE SIN MIEDO PROCESOS BUREAUCRÁTICOS

Hasta ahora, hemos discutido la revisión por pares y los procesos de programación de pares que nos permiten
aumentar la calidad de nuestro trabajo sin depender de aprobaciones externas para cambios.
Sin embargo, muchas compañías todavía tienen procesos de aprobación de larga data que requieren
meses para navegar. Estos procesos de aprobación pueden aumentar significativamente los plazos de entrega, no
solo evitando que entreguemos valor rápidamente a los clientes, pero potencialmente aumentando
El riesgo para nuestros objetivos organizacionales. Cuando esto sucede, debemos rediseñar nuestro
procesos para que podamos alcanzar nuestros objetivos de manera más rápida y segura.

Como observó Adrian Cockcroft: “Una gran medida para publicar ampliamente es cuántas reuniones y
los tickets de trabajo son obligatorios para realizar un lanzamiento; el objetivo es reducir implacablemente
esfuerzo requerido para que los ingenieros realicen el trabajo y lo entreguen al cliente ".

Del mismo modo, el Dr. Tapabrata Pal, miembro técnico de Capital One, describió un programa en
Capital One llamado Got Goo ?, que involucra a un equipo dedicado que elimina obstáculos:
incluidas herramientas, procesos y aprobaciones que impiden la finalización del trabajo. Jason Cox,
Director sénior de ingeniería de sistemas en Disney, descrito en su presentación en el

DevOps Enterprise Summit en 2015 un programa llamado Join The Rebellion que tenía como objetivo eliminar el trabajo y los obstáculos del trabajo diario.

En Target en 2012, una combinación del Proceso de adopción de Enterprise Technology y Lead La Junta de Revisión de Arquitectura (proceso TEAP-LARB) resultó en una aprobación larga y complicada veces para cualquiera que intente incorporar nuevas tecnologías. El formulario TEAP necesitaba ser llenado cualquier persona que quiera proponer nuevas tecnologías para ser adoptadas, como una nueva base de datos o tecnologías de monitoreo. Estas propuestas fueron evaluadas y las consideradas apropiado se incluyeron en la agenda mensual de la reunión LARB.

Heather Mickman y Ross Clanton, Director de Desarrollo y Director de Operaciones en Target, Inc., respectivamente, estaban ayudando a liderar el movimiento DevOps en Target. Durante su iniciativa DevOps, Mickman había identificado una tecnología necesaria para permitir una iniciativa de las líneas de negocio (en este caso, Tomcat y Cassandra). La decisión del LARB fue que Operaciones no podía soportarlo en ese momento. Sin embargo, porque Mickman Estaba tan convencida de que esta tecnología era esencial, que propuso que su Desarrollo El equipo será responsable del servicio de soporte, así como de la integración, disponibilidad y seguridad, en lugar de depender del equipo de operaciones.

“A medida que avanzábamos por el proceso, quería entender mejor por qué el TEAP-LARB el proceso tardó mucho en completarse, y utilicé la técnica de 'los cinco por qué' ... finalmente llevó a la pregunta de por qué TEAP-LARB existía en primer lugar. Lo sorprendente Lo que pasaba era que nadie lo sabía, aparte de una vaga noción de que necesitábamos algún tipo de proceso de gobernanza. Muchos sabían que había habido algún tipo de desastre que nunca podría sucedió de nuevo hace años, pero tampoco nadie podía recordar exactamente qué fue ese desastre " Mickman observó.

Mickman concluyó que este proceso no era necesario para su grupo si eran responsable de las responsabilidades operativas de la tecnología que estaba introduciendo. Ella y agregó: "Dejo que todos sepan que las tecnologías futuras que admitiríamos no tiene que pasar por el proceso TEAP-LARB, tampoco ".

El resultado fue que Cassandra se introdujo con éxito dentro de Target y eventualmente ampliamente adoptado. Además, el proceso TEAP-LARB fue finalmente desmantelado. Fuera de agradecimiento, su equipo otorgó a Mickman el Lifetime Achievement Award por eliminar barreras para realizar el trabajo tecnológico dentro de Target.

CONCLUSIÓN

En este capítulo, discutimos cómo integrar las prácticas en nuestro trabajo diario que aumentan la calidad de nuestros cambios y reducir el riesgo de malos resultados de implementación, reduciendo nuestro dependencia de los procesos de aprobación. Los estudios de caso de GitHub y Target muestran que estos las prácticas no solo mejoran nuestros resultados, sino que también reducen significativamente los plazos de entrega y

Aumentar la productividad del desarrollador. Hacer este tipo de trabajo requiere una cultura de alta confianza.

Considere una historia que John Allspaw contó sobre un ingeniero junior recién contratado: el ingeniero preguntó si estaba bien implementar un pequeño cambio de HTML, y Allspaw respondió: "No ¿Lo sabes? Luego preguntó: "¿Hiciste que alguien revisara tu cambio? Sabes quien la mejor persona para preguntar es por cambios de este tipo? Hiciste todo lo que absolutamente

Page 289

¿podría asegurarse de que este cambio opera en la producción como fue diseñado? Si lo hiciste, entonces no me preguntes, ¡solo haz el cambio!

Al responder de esta manera, Allspaw le recordó al ingeniero que ella era la única responsabilidad por la calidad de su cambio: si hacía todo lo que sentía, podía darse confianza en que el cambio funcionaría, entonces no necesitaba pedirle aprobación a nadie, ella debería hacer el cambio.

Crear las condiciones que permitan a los implementadores de cambios ser dueños de la calidad de sus Los cambios son una parte esencial de la cultura generativa de alta confianza que nos esforzamos por construir. Además, estas condiciones nos permiten crear un sistema de trabajo cada vez más seguro, donde todos nos estamos ayudando mutuamente a alcanzar nuestras metas, abarcando los límites necesarios para ir allí.

PARTE IV CONCLUSIÓN

La Parte IV nos ha demostrado que al implementar bucles de retroalimentación podemos permitir que todos trabajen juntos hacia objetivos compartidos, ver los problemas a medida que ocurren y, con detección rápida y recuperación, asegúrese de que las funciones no solo funcionen según lo diseñado en la producción, sino que también logren objetivos organizacionales y aprendizaje organizacional. También hemos examinado cómo habilitar objetivos compartidos que abarcan Dev y Ops para que puedan mejorar la salud de todo el valor corriente.

Ahora estamos listos para ingresar a la Parte V: La Tercera Vía, *Las Prácticas Técnicas de Aprendizaje* , entonces podemos crear oportunidades de aprendizaje que sucedan antes y cada vez más rápido y a bajo precio, y para que podamos desatar una cultura de innovación y experimentación que permite a todos hacer un trabajo significativo que ayuda a nuestra organización a tener éxito.

† *El pensamiento contrafáctico* es un término utilizado en psicología que implica la tendencia humana a crear posibles alternativas a los eventos de la vida que ya ocurrió. En ingeniería de confiabilidad, a menudo involucra narrativas del "sistema como se imagina" en oposición al "sistema en la realidad".

‡ En este libro, los términos *revisión de código* y *revisión de cambio* se usarán indistintamente.

§ Por cierto, es probable que la junta asesora de cambios ya haya creado una lista de áreas de código y entornos de alto riesgo.

¶ Algunas organizaciones pueden requerir programación de emparejamiento, mientras que en otras, los ingenieros encuentran a alguien para emparejar el programa cuando trabajan en las áreas donde desean más escrutinio (como antes de registrarse) o para tareas desafiantes. Otra práctica común es establecer *horas de emparejamiento* para un subconjunto de la jornada laboral, quizás cuatro horas desde media mañana hasta media tarde.

** Gene Kim agradece a Shawn Davenport, James Fryman, Will Farr y Ryan Tomayko en GitHub por discutir las diferencias entre lo bueno y lo malo Solicitudes de extracción.

Parte

Introducción

En la Parte III, La primera vía: *las prácticas técnicas de Flow* , discutimos la implementación de las prácticas requeridas para crear un flujo rápido en nuestro flujo de valor. En la Parte IV, el Segunda forma: *las prácticas técnicas de retroalimentación* , nuestro El objetivo era crear la mayor cantidad de comentarios posible, desde

muchas áreas en nuestro sistema como sea posible, antes, más rápido,
Y más barato.

En la Parte V, La tercera vía: *las prácticas técnicas de Aprendiendo* , presentamos las prácticas que crean Oportunidades de aprendizaje, tan rápido, con frecuencia, a bajo precio y lo antes posible. Esto incluye crear aprendizajes de accidentes y fallas, que son inevitable cuando trabajamos dentro de sistemas complejos, también como organizar y diseñar nuestros sistemas de trabajo para que estamos constantemente experimentando y aprendiendo, haciendo continuamente nuestros sistemas más seguros. Los resultados incluir una mayor resiliencia y un colectivo cada vez mayor conocimiento de cómo funciona realmente nuestro sistema, para que podamos somos más capaces de lograr nuestros objetivos.

En los siguientes capítulos, institucionalizaremos rituales. que aumentan la seguridad, la mejora continua y

aprender haciendo lo siguiente:

Establecer una cultura justa para hacer posible la seguridad.

Inyectar fallas de producción para crear resiliencia

Convierta los descubrimientos locales en mejoras globales

Reserve tiempo para crear mejoras organizacionales
Y aprendiendo

También crearemos mecanismos para que cualquier nuevo
los aprendizajes generados en un área de la organización pueden
ser utilizado rápidamente en toda la organización, girando
mejoras locales en avances mundiales. En esto
así, no solo aprendemos más rápido que nuestra competencia,
ayudándonos a ganar en el mercado, pero también a crear un lugar más seguro,
cultura laboral más resistente que la gente está emocionada de ser
una parte de y eso les ayuda a alcanzar su máximo
potencial.

Cuando trabajamos dentro de un sistema complejo, es imposible para nosotros predecir todos los resultados por las acciones que tomamos. Esto contribuye a situaciones inesperadas y a veces catastróficas. accidentes, incluso cuando utilizamos herramientas de precaución estáticas, como listas de verificación y runbooks, que codifican nuestra comprensión actual del sistema.

Para permitirnos trabajar de manera segura dentro de sistemas complejos, nuestras organizaciones deben convertirse siempre mejor en autodiagnóstico y superación personal y debe ser hábil para detectar problemas, resolviéndolos y multiplicando los efectos haciendo que las soluciones estén disponibles en todo el organización. Esto crea un sistema dinámico de aprendizaje que nos permite comprender nuestro errores y traducir esa comprensión en acciones que eviten que esos errores recurrente en el futuro.

El resultado es lo que el Dr. Steven Spear describe como organizaciones resistentes, que son "hábiles en detectar problemas, resolverlos y multiplicar el efecto haciendo las soluciones disponible en toda la organización ". Estas organizaciones pueden curarse a sí mismas. "Para tal Una organización que responde a las crisis no es un trabajo idiosincrásico. Es algo que se hace todo el tiempo. Esta capacidad de respuesta es su fuente de fiabilidad ".

Un ejemplo sorprendente de la increíble capacidad de recuperación que puede resultar de estos principios y las prácticas se vieron el 21 de abril de 2011, cuando toda la disponibilidad de Amazon AWS US-EAST la zona se redujo, eliminando prácticamente a todos sus clientes que dependían de ella, incluidos Reddit y Quora.† Sin embargo, Netflix fue una sorprendente excepción, aparentemente no afectada por esta interrupción masiva de AWS.

Después del evento, se especuló considerablemente sobre cómo Netflix mantuvo su Servicios en ejecución. Una teoría popular fue que Netflix era uno de los mayores clientes de Amazon Web Services, recibió un tratamiento especial que les permitió mantener corriendo. Sin embargo, una publicación de blog de *Netflix Engineering* explicó que era su arquitectura Las decisiones de diseño en 2009 permitieron su excepcional resistencia.

En 2008, el servicio de entrega de video en línea de Netflix se ejecutó en una aplicación J2EE monolítica alojado en uno de sus centros de datos. Sin embargo, a partir de 2009, comenzaron a rediseñar este sistema es lo que ellos llamaron *nube nativa:* fue diseñado para ejecutarse completamente en el Nube pública de Amazon y ser lo suficientemente resistente como para sobrevivir a fallas significativas.

Uno de sus objetivos de diseño específicos era garantizar que los servicios de Netflix siguieran funcionando, incluso si toda una zona de disponibilidad de AWS cayó, como sucedió con US-EAST. Para hacer esto requirieron que su sistema estuviera débilmente acoplado, con cada componente con agresivo tiempos de espera para garantizar que los componentes defectuosos no derriben todo el sistema. cada característica y componente fue diseñado para degradarse con gracia. Por ejemplo, durante oleadas de tráfico que crearon picos de uso de CPU, en lugar de mostrar una lista de películas

personalizados para el usuario, mostrarían contenido estático, como el almacenamiento en caché o no personalizado

resultados, que requirieron menos cómputo.

Además, la publicación del blog explica que, además de implementar estas arquitecturas
patrones, también construyeron y habían estado ejecutando un servicio sorprendente y audaz llamado
Chaos Monkey , que simulaba fallas de AWS matando constantemente y al azar
servidores de producción. Lo hicieron porque querían que todos los "equipos de ingeniería estuvieran acostumbrados a un
nivel constante de falla en la nube "para que los servicios puedan" recuperarse automáticamente sin
cualquier intervención manual ".

En otras palabras, el equipo de Netflix dirigió Chaos Monkey para asegurarse de que tenían
lograron sus objetivos de resiliencia operacional, inyectando constantemente fallas en sus
Producción y entornos de producción.

Como era de esperar, cuando corrieron Chaos Monkey por primera vez en sus entornos de producción,
los servicios fallaron en formas que nunca podrían haber predicho o imaginado, al encontrar constantemente
y solucionando estos problemas durante las horas normales de trabajo, los ingenieros de Netflix rápidamente y
creó iterativamente un servicio más resistente, mientras que simultáneamente creaba una organización
aprendizajes (¡durante las horas normales de trabajo!) que les permitieron evolucionar mucho sus sistemas
más allá de su competencia.

Chaos Monkey es solo un ejemplo de cómo el aprendizaje puede integrarse en el trabajo diario. los
La historia también muestra cómo las organizaciones de aprendizaje piensan acerca de los fracasos, accidentes y errores.
como una oportunidad para aprender y no como algo para ser castigado. Este capítulo explora
cómo crear un sistema de aprendizaje y cómo establecer una *cultura justa* , así como cómo
ensayar rutinariamente y crear deliberadamente fallas para acelerar el aprendizaje.

ESTABLECER UNA CULTURA DE APRENDIZAJE JUSTA

Uno de los requisitos previos para una cultura de aprendizaje es que cuando ocurren accidentes (que
indudablemente lo hará), la respuesta a esos accidentes se considera "justa". Dr. Sidney Dekker, quien
ayudó a codificar algunos de los elementos clave de la cultura de seguridad y acuñó el término *cultura justa* ,
escribe: “Cuando las respuestas a incidentes y accidentes se consideran injustas, esto puede impedir la seguridad
investigaciones, promoviendo el miedo en lugar de la atención plena en las personas que son críticas para la seguridad
trabajar, hacer que las organizaciones sean más burocráticas en lugar de más cuidadosas y cultivar
secreto profesional, evasión y autoprotección ".

Esta noción de castigo está presente, sutil o prominentemente, en la forma en que muchos
Los gerentes han operado durante el siglo pasado. El pensamiento va, para lograr el
objetivos de la organización, los líderes deben ordenar, controlar, establecer procedimientos para
eliminar errores y hacer cumplir los procedimientos.

El Dr. Dekker llama a esta noción de eliminar el error al eliminar a las personas que causaron el
errores de la *teoría de* la *mala manzana* . Afirma que esto no es válido, porque "el error humano no es
nuestra causa de problemas; en cambio, el error humano es una consecuencia del diseño de las herramientas que
les dimos."

Si los accidentes no son causados por "manzanas podridas", sino que se deben a un diseño inevitable
problemas en el sistema complejo que creamos, entonces en lugar de "nombrar, culpar y
avergonzar a la persona que causó el fracaso, nuestro objetivo siempre debe ser maximizar

oportunidades para el aprendizaje organizacional, reforzando continuamente que valoramos las acciones que exponer y compartir más ampliamente los problemas en nuestro trabajo diario. Esto es lo que nos permite mejorar la calidad y la seguridad del sistema en el que operamos y reforzar el relaciones entre todos los que operan dentro de ese sistema.

Al convertir la información en conocimiento y construir los resultados del aprendizaje en nuestro sistemas, comenzamos a lograr los objetivos de una cultura justa, equilibrando las necesidades de seguridad y responsabilidad. Como John Allspaw, CTO de Etsy, afirma: "Nuestro objetivo en Etsy es ver los errores, errores, resbalones, fallas, etc. con una perspectiva de aprendizaje ".

Cuando los ingenieros cometen errores y se sienten seguros al dar detalles al respecto, no solo son dispuestos a rendir cuentas, pero también están entusiasmados en ayudar al resto del Empresa evitar el mismo error en el futuro. Esto es lo que crea el aprendizaje organizacional. Por otro lado, si castigamos a ese ingeniero, todos están desincentivados para proporcionar detalles necesarios para comprender el mecanismo, la patología y el funcionamiento de la falla, lo que garantiza que la falla volverá a ocurrir.

Dos prácticas efectivas que ayudan a crear una cultura justa basada en el aprendizaje son irreprensibles mortems y la introducción controlada de fallas en la producción para crear oportunidades practicar para los problemas inevitables que surgen dentro de sistemas complejos. Primero miraremos en autopsias sin culpa y siga con una exploración de por qué el fracaso puede ser un buen cosa.

PROGRAMA REUNIONES SIN CORTOMETRAJE SIN MORTEM DESPUÉS DE ACCIDENTES OCURREN

Para ayudar a permitir una cultura justa, cuando ocurren accidentes e incidentes significativos (por ejemplo, fallidos implementación, problema de producción que afectó a los clientes), debemos llevar a cabo un proceso *sin culpa después de la muerte* después de que el incidente se haya resuelto.[1] Post-mortems sin culpa, un término acuñado por John Allspaw, ayúdenos a examinar "errores de una manera que se centre en la situación aspectos del mecanismo de una falla y el proceso de toma de decisiones de individuos cercanos al fracaso ".

Para hacer esto, programamos la autopsia lo antes posible después de que ocurra el accidente y antes de que los recuerdos y los vínculos entre causa y efecto se desvanezcan o cambien las circunstancias. (De Por supuesto, esperamos hasta que el problema se haya resuelto para no distraer a las personas. que todavía están trabajando activamente en el tema).

En la reunión post mortem sin culpa, haremos lo siguiente:

- Construya una línea de tiempo y recopile detalles desde múltiples perspectivas sobre fallas, asegurando no castigamos a las personas por cometer errores
- Empoderar a todos los ingenieros para mejorar la seguridad permitiéndoles dar cuentas detalladas de sus contribuciones a los fracasos
- Permitir y alentar a las personas que cometen errores a ser los expertos que educan al

resto de la organización sobre cómo no hacerlos en el futuro

Acepte que siempre hay un espacio discrecional donde los humanos pueden decidir tomar acción o no, y que el juicio de esas decisiones reside en retrospectiva

Proponer contramedidas para evitar que ocurra un accidente similar en el futuro y asegurarse de que estas contramedidas se registren con una fecha objetivo y un propietario para seguimiento

Para que podamos obtener esta comprensión, las siguientes partes interesadas deben estar presentes en la reunión:

Las personas involucradas en las decisiones que pueden haber contribuido al problema.

Las personas que identificaron el problema.

Las personas que respondieron al problema.

Las personas que diagnosticaron el problema.

Las personas afectadas por el problema.

Y cualquier otra persona que esté interesada en asistir a la reunión.

Nuestra primera tarea en la reunión post-mortem sin culpa es registrar nuestra mejor comprensión de la línea de tiempo de los eventos relevantes a medida que ocurrieron. Esto incluye todas las acciones que tomamos y qué tiempo (idealmente respaldado por registros de chat, como IRC o Slack), qué efectos observamos (idealmente en forma de métricas específicas de nuestra telemetría de producción, en lugar de meras narrativas subjetivas), todos los caminos de investigación que seguimos y qué resoluciones fueron considerados.

Para permitir estos resultados, debemos ser rigurosos al registrar detalles y reforzar un cultura que la información se puede compartir sin temor a castigo o retribución. Porque de esto, especialmente para nuestras primeras autopsias, puede ser útil tener la reunión dirigida por un facilitador capacitado que no estuvo involucrado en el accidente.

Durante la reunión y la resolución posterior, debemos rechazar explícitamente frases "would have" o "could have", ya que son declaraciones *contrafácticas* que resultan de nuestra tendencia humana a crear posibles alternativas a eventos que ya han ocurrió.

Declaraciones contrafactuales, como "Podría haber ..." o "Si hubiera sabido sobre eso, debería tener ... ", enmarca el problema en términos del *sistema como se imagina en* lugar de en términos de *sistema que realmente existe* , que es el contexto al que debemos restringirnos. Ver

Apéndice 8.

Uno de los resultados potencialmente sorprendentes de estas reuniones es que la gente suele culpar
ellos mismos por cosas fuera de su control o cuestionan sus propias habilidades. Ian Malpass,
un ingeniero de Etsy observa: "En ese momento cuando hacemos algo que causa la
todo el sitio para bajar, tenemos esta sensación de 'agua helada en la columna vertebral', y probablemente la primera
Pensamos en nuestra cabeza: "Soy un asco y no tengo idea de lo que estoy haciendo". Necesitamos parar
nosotros mismos de hacer eso, ya que es la ruta hacia la locura, la desesperación y los sentimientos de ser un
impostor, que es algo que no podemos dejar que les pase a los buenos ingenieros. El mejor
La pregunta en la que centrarse es: "¿Por qué tenía sentido para mí cuando tomé esa acción?"

En la reunión, debemos reservar suficiente tiempo para hacer una lluvia de ideas y decidir qué
contramedidas para implementar. Una vez que se han identificado las contramedidas, deben
se le dará prioridad y se le dará un propietario y un cronograma para la implementación. Haciendo esto más allá
demuestra que valoramos la mejora de nuestro trabajo diario más que el trabajo diario en sí.

Dan Milstein, uno de los principales ingenieros de Hubspot, escribe que comienza sin culpa alguna.
reuniones post-mortem "al decir: 'Estamos tratando de prepararnos para un futuro donde estamos
estúpidos como somos hoy ". En otras palabras, no es aceptable tener una contramedida para
simplemente "ser más cuidadoso" o "ser menos estúpido"; en cambio, debemos diseñar
contramedidas para evitar que estos errores vuelvan a ocurrir.

Ejemplos de tales contramedidas incluyen nuevas pruebas automatizadas para detectar peligros
condiciones en nuestra tubería de implementación, agregando más telemetría de producción, identificando
categorías de cambios que requieren una revisión por pares adicional y la realización de ensayos de este
categoría de falla como parte de los ejercicios programados regularmente para el Día del Juego.

PUBLIQUE NUESTROS POST-MORTEMS TAN AMPLIO COMO ES POSIBLE

Después de llevar a cabo una reunión post mortem sin culpa, debemos anunciar ampliamente
disponibilidad de las notas de la reunión y cualquier artefacto asociado (por ejemplo, líneas de tiempo, registros de chat de IRC,
comunicaciones externas). Esta información debería (idealmente) colocarse en un lugar centralizado
ubicación donde toda nuestra organización puede acceder a ella y aprender del incidente.
La realización de autopsias es tan importante que incluso podemos prohibir los incidentes de producción.
desde que se cerró hasta que se haya completado la reunión post mortem.

Hacer esto nos ayuda a traducir los aprendizajes locales y las mejoras en aprendizajes globales y
mejoras Randy Shoup, ex director de ingeniería de Google App Engine,
describe cómo la documentación de las reuniones post mortem puede tener un gran valor para
otros en la organización, “Como puedes imaginar en Google, todo se puede buscar. Todos
los documentos post mortem se encuentran en lugares donde otros Googlers pueden verlos. Y confía en mi
cuando cualquier grupo tiene un incidente que suena similar a algo que sucedió antes,
estos documentos post mortem se encuentran entre los primeros documentos que se leen y estudian ". §

Publicar ampliamente autopsias y alentar a otros en la organización a leerlas aumenta el aprendizaje organizacional, y también se está volviendo cada vez más común para empresas de servicios en línea para publicar autopsias para interrupciones que afecten a los clientes. Esta a menudo aumenta significativamente la transparencia que tenemos con nuestros internos y externos clientes, lo que a su vez aumenta su confianza en nosotros.

Este deseo de llevar a cabo tantas reuniones post mortem sin culpa como sea necesario en Etsy llevó a algunos problemas: en el transcurso de cuatro años, Etsy acumuló una gran cantidad de notas de la reunión de mortem en páginas wiki, que se hicieron cada vez más difíciles de buscar, guardar, y colaborar desde.

Para ayudar con este problema, desarrollaron una herramienta llamada Morgue para registrar fácilmente aspectos de cada accidente, como el incidente MTTR y la gravedad, abordar mejor las zonas horarias (que se volvió relevante a medida que más empleados de Etsy trabajaban de forma remota) e incluyen otros datos, como texto enriquecido en formato Markdown, imágenes incrustadas, etiquetas e historial.

Morgue fue diseñado para facilitar que el equipo registre:

- Si el problema se debió a un incidente programado o no programado
- El dueño de la autopsia
- Registros de chat de IRC relevantes (especialmente importante para los problemas de las 3 a. M. tomar no puede suceder)
- Boletos relevantes de JIRA para acciones correctivas y sus fechas de vencimiento (información particularmente importante para la gestión)
- Enlaces a publicaciones en foros de clientes (donde los clientes se quejan de problemas)

Después de desarrollar y usar Morgue, el número de autopsias registradas en Etsy aumentó significativamente en comparación con cuando usaban páginas wiki, especialmente para P2, P3 y Incidentes P4 (es decir, problemas de menor gravedad). Este resultado reforzó la hipótesis de que si facilitaron la documentación de las autopsias a través de herramientas como Morgue, más personas registraría y detallaría los resultados de sus reuniones post mortem, permitiendo más Aprendizaje organizacional.

Dra. Amy C. Edmondson, profesora de liderazgo y gestión de Novartis en Harvard Business School y coautora de *Building the Future: Big Teaming for Audacious Innovación,* escribe:

> Nuevamente, el remedio, que no necesariamente implica mucho tiempo y gastos, es reducir el estigma del fracaso Eli Lilly ha hecho esto desde principios de la década de 1990 al mantener 'partidos fallidos' para honrar experimentos científicos inteligentes y de alta calidad que no logran lograr los resultados deseados. Las fiestas no cuestan mucho, y la redistribución valiosa

recursos, particularmente científicos, para nuevos proyectos más temprano que tarde pueden ahorrar cientos de miles de dólares, sin mencionar los posibles nuevos descubrimientos.

DISMINUYA LAS TOLERANCIAS DE INCIDENTES PARA ENCONTRAR NUNCA SEÑALES DE FALLA DE DEBIL

Inevitablemente, a medida que las organizaciones aprenden a ver y resolver problemas de manera eficiente, necesitan disminuir el umbral de lo que constituye un problema para seguir aprendiendo. Para hacer esto, buscamos amplificar las señales de falla débiles. Por ejemplo, como se describe en el capítulo 4, cuando Alcoa fue capaz de reducir la tasa de accidentes laborales para que ya no fueran lugar común, Paul O'Neill, CEO de Alcoa, comenzó a ser notificado de accidentes cercanos a accidentes en Además de los accidentes laborales reales.

El Dr. Spear resume los logros de O'Neill en Alcoa cuando escribe: "Aunque comenzó centrándose en los problemas relacionados con la seguridad en el lugar de trabajo, pronto descubrió que la seguridad los problemas reflejan la ignorancia del proceso y que esta ignorancia también se manifestaría en otros problemas como la calidad, la puntualidad y el rendimiento frente a la chatarra ".

Cuando trabajamos dentro de sistemas complejos, esta necesidad de amplificar señales de falla débiles es crítica para evitar fallas catastróficas. La forma en que la NASA manejó las señales de falla durante el espacio La era del transbordador sirve como un ejemplo ilustrativo: en 2003, dieciséis días en el espacio de *Columbia* misión del transbordador, explotó cuando volvió a entrar en la atmósfera de la tierra. Ahora sabemos que un Una pieza de espuma aislante se había desprendido del tanque de combustible externo durante el despegue.

Sin embargo, antes de *la* reentrada *de Columbia* , un puñado de ingenieros de nivel medio de la NASA tenían informó este incidente, pero sus voces no habían sido escuchadas. Observaron la huelga de espuma en monitores de video durante una sesión de revisión posterior al lanzamiento e inmediatamente notificó a la NASA gerentes, pero les dijeron que el problema de la espuma no era nada nuevo. Desalojo de espuma tenía lanzaderas dañadas en lanzamientos anteriores, pero nunca habían resultado en un accidente. Era consideró un problema de mantenimiento y no actuó hasta que fue demasiado tarde.

Michael Roberto, Richard MJ Bohmer y Amy C. Edmondson escribieron en un artículo de 2006 para *Harvard Business Review* cómo la cultura de la NASA contribuyó a este problema. Ellos describe cómo las organizaciones se estructuran típicamente en uno de dos modelos: un *estándar modelo* , donde la rutina y los sistemas gobiernan todo, incluido el estricto cumplimiento de plazos y presupuestos, o un *modelo experimental* , donde todos los días cada ejercicio y cada Se evalúa y debate una nueva información en una cultura que se asemeja a una investigación y laboratorio de diseño (I + D).

Observan: "Las empresas se meten en problemas cuando aplican la mentalidad incorrecta a un organización [que dicta cómo responden a *amenazas ambiguas* o, en la terminología de este libro, *señales de fallas débiles*]…. En la década de 1970, la NASA había creado una cultura de rigidez estandarización, promoviendo al Congreso el transbordador espacial como un producto barato y reutilizable

astronave." La NASA favoreció el estricto cumplimiento del proceso en lugar de un modelo experimental donde cada pieza de información necesitaba ser evaluada ya que ocurría sin sesgos. los La ausencia de aprendizaje y experimentación continua tuvo consecuencias nefastas. Los autores concluyen que lo que importa es la cultura y la mentalidad, no solo "tener cuidado", como escriba, "la vigilancia por sí sola no evitará amenazas ambiguas [señales de falla débiles] convirtiéndose en fracasos costosos (ya veces trágicos) ".

Nuestro trabajo en el flujo de valor de la tecnología, como los viajes espaciales, debe abordarse como un esfuerzo fundamentalmente experimental y gestionado de esa manera. Todo el trabajo que hacemos es un hipótesis potencialmente importante y una fuente de datos, en lugar de una aplicación de rutina y validación de prácticas pasadas. En lugar de tratar el trabajo tecnológico como un todo estandarizado, donde nos esforzamos por el cumplimiento del proceso, debemos buscar continuamente encontrar señales de falla cada vez más débiles para que podamos entender y administrar mejor el sistema que operar en.

REDEFINE EL FALLO Y ALIENTE EL RIESGO CALCULADO TOMANDO

Los líderes de una organización, ya sea de manera deliberada o inadvertida, refuerzan el cultura organizacional y valores a través de sus acciones. Auditoría, contabilidad y ética. Los expertos han observado durante mucho tiempo que el "tono en la parte superior" predice la probabilidad de fraude y Otras prácticas poco éticas. Para reforzar nuestra cultura de aprendizaje y la asunción de riesgos calculados, necesitamos líderes que refuercen continuamente que todos deberían sentirse cómodos con y responsable de salir a la superficie y aprender de los fracasos.

Sobre las fallas, Roy Rapoport de Netflix observa: "Qué es el *Informe del Estado de DevOps 2014* me demostró que las organizaciones DevOps de alto rendimiento fallarán y cometerán errores más a menudo. Esto no solo está bien, ¡es lo que las organizaciones necesitan! Incluso puedes verlo en el

Página 302

datos: si los de alto rendimiento se desempeñan treinta veces más frecuentemente pero con solo la mitad del cambiar la tasa de fallas, obviamente están teniendo más fallas ".

Él continúa: "Estaba hablando con un compañero de trabajo sobre una interrupción masiva que acabamos de tener en Netflix —Fue, francamente, un error tonto. De hecho, fue causado por un ingeniero que tenía derribado Netflix dos veces en los últimos dieciocho meses. Pero, por supuesto, esta es una persona que tendríamos Nunca dispares. En los mismos dieciocho meses, este ingeniero cambió el estado de nuestras operaciones. y automatización hacia adelante no por millas sino por años luz. Ese trabajo nos ha permitido hacer implementaciones de forma segura a diario, y ha realizado personalmente un gran número de despliegues de producción ".

Concluye: "DevOps debe permitir este tipo de innovación y los riesgos resultantes de las personas

cometiendo errores. Sí, tendrás más fallas en la producción. Pero eso es algo bueno, y no debe ser castigado ".

FALLAS EN LA PRODUCCIÓN DE INYECTOS PARA HABILITAR LA RESILIENCIA Y APRENDIENDO

Como vimos en la introducción del capítulo, inyectando fallas en el entorno de producción (como Chaos Monkey) es una forma de aumentar nuestra capacidad de recuperación. En esta sección, nosotros describir los procesos involucrados en ensayar e inyectar fallas en nuestro sistema para confirmar que hemos diseñado y diseñado nuestros sistemas correctamente, de modo que las fallas suceder de manera específica y controlada. Hacemos esto regularmente (o incluso continuamente) realizando pruebas para asegurarnos de que nuestros sistemas fallan con gracia.

Como Michael Nygard, autor de *Release It! Diseñar e implementar software listo para producción,* comenta: "Como construir zonas de deformación en los automóviles para absorber los impactos y mantener a los pasajeros seguro, puede decidir qué características del sistema son indispensables y generar fallos modos que mantienen las grietas alejadas de esas características. Si no diseña sus modos de falla, entonces obtendrás lo impredecible (y generalmente peligroso) que le suceda a surgir."

La resistencia requiere que primero definamos nuestros modos de falla y luego realicemos pruebas para asegúrese de que estos modos de falla funcionen según lo diseñado. Una forma de hacerlo es inyectando fallas en nuestro entorno de producción y ensayar fallas a gran escala por lo que estamos seguro de que podemos recuperarnos de los accidentes cuando ocurren, idealmente sin afectar nuestros clientes.

La historia de 2012 sobre Netflix y la interrupción de Amazon AWS-EAST presentada en el La introducción es solo un ejemplo. Un ejemplo aún más interesante de resiliencia en Netflix fue durante el "Gran reinicio de Amazon de 2014", cuando casi el 10% de toda la Amazonía La flota de servidores EC2 tuvo que reiniciarse para aplicar un parche de seguridad Xen de emergencia. Como Christos Kalantzis de Netflix Cloud Database Engineering recordó: “Cuando recibimos las noticias sobre el reinicios de emergencia EC2, nuestras mandíbulas cayeron. Cuando obtuvimos la lista de cuántos Cassandra los nodos se verían afectados, me sentí enfermo ". Pero Kalantzis continúa:" Entonces recordé todos los Ejercicios de Chaos Monkey por los que hemos pasado. Mi reacción fue: "¡Adelante!"

Una vez más, los resultados fueron sorprendentes. De los más de 2,700 nodos Cassandra utilizados en producción, 218 se reiniciaron y veintidós no se reiniciaron con éxito. Como Kalantzis y

Bruce Wong de Netflix Chaos Engineering escribió: "Netflix experimentó 0 tiempo de inactividad que fin de semana. Repetir y regular el ejercicio del fracaso, incluso en la persistencia [base de datos] capa, debe ser parte de la planificación de resiliencia de cada empresa. Si no fuera por Cassandra participación en Chaos Monkey, esta historia habría terminado de manera muy diferente ".

Aún más sorprendente, no solo nadie en Netflix estaba trabajando incidentes activos debido a fallas Cassandra nodos, nadie estaba en la oficina, estaban en Hollywood en una fiesta celebrando un hito de adquisición. Este es otro ejemplo que demuestra que centrarse proactivamente en la resiliencia a menudo significa que una empresa puede manejar eventos que pueden causar crisis para la mayoría de las organizaciones de manera rutinaria y mundana. ¶ Ver Apéndice 9.

INSTITUTO JUEGO DÍAS PARA ENSAYAR FALLAS

En esta sección, describimos ensayos específicos de recuperación ante desastres llamados Game Days, un término popularizado por Jesse Robbins, uno de los fundadores de la comunidad Velocity Conference y cofundador de Chef, por el trabajo que hizo en Amazon, donde fue responsable de programas para garantizar la disponibilidad del sitio y fue ampliamente conocido internamente como el "Maestro de Desastre ". El concepto de Game Days proviene de la disciplina de la *ingeniería* de *resiliencia* . Robbins define la ingeniería de resiliencia como "un ejercicio diseñado para aumentar la resiliencia a través de la inyección de fallas a gran escala en los sistemas críticos ".

Robbins observa que "cada vez que se propone diseñar un sistema a escala, lo mejor que puede la esperanza es construir una plataforma de software confiable sobre componentes que sean completamente no fidedigno. Eso lo coloca en un entorno donde las fallas complejas son inevitables e impredecible ".

En consecuencia, debemos asegurarnos de que los servicios continúen operando cuando ocurran fallas, potencialmente en todo nuestro sistema, idealmente sin crisis o incluso intervención manual. Como Robbins bromea: "un servicio no se prueba realmente hasta que lo rompemos en la producción".

Nuestro objetivo para Game Day es ayudar a los equipos a simular y ensayar accidentes para darles el Habilidad para practicar. Primero, programamos un evento catastrófico, como el simulado destrucción de un centro de datos completo, que sucederá en algún momento en el futuro. Luego damos los equipos deben prepararse, eliminar todos los puntos únicos de falla y crear el procedimientos de monitoreo necesarios, procedimientos de conmutación por error, etc.

Nuestro equipo de Game Day define y ejecuta simulacros, como realizar failovers de bases de datos (es decir, simulando una falla de la base de datos y asegurando que la base de datos secundaria funcione) o convirtiendo fuera de una conexión de red importante para exponer problemas en los procesos definidos. Ninguna Los problemas o dificultades que se encuentran se identifican, abordan y prueban nuevamente.

A la hora programada, ejecutamos la interrupción. Como describe Robbins, en Amazon ellos "Literalmente apagaría una instalación, sin previo aviso, y luego dejaría que los sistemas fallaran naturalmente y [permitir] que las personas sigan sus procesos donde sea que lo guíen ".

Al hacer esto, comenzamos a exponer los *defectos latentes* en nuestro sistema, que son los problemas que aparecen solo por haber inyectado fallas en el sistema. Robbins explica: "Usted podría descubrir que ciertos sistemas de monitoreo o administración son cruciales para la recuperación el proceso termina apagándose como parte de la falla que ha orquestado. [O] lo harías encuentre algunos puntos únicos de falla que no conocía de esa manera ". Estos ejercicios son entonces

conducido de una manera cada vez más intensa y compleja con el objetivo de hacerlos sentir como solo otra parte de un día promedio.

Al ejecutar Game Days, creamos progresivamente un servicio más resistente y mejor grado de seguridad de que podemos reanudar las operaciones cuando ocurren eventos inoportunos, también crear más aprendizajes y una organización más resistente.

Un excelente ejemplo de simulación de desastres es el Programa de recuperación de desastres de Google (DiRT). Kripa Krishnan es directora técnica de programas en Google y, al momento de escribir esto, ha liderado el programa por más de siete años. Durante ese tiempo, han simulado un terremoto en Silicon Valley, que resultó en todo el campus de Mountain View desconectado de Google; principales centros de datos que tienen pérdida total de energía; e incluso Extranjeros que atacan ciudades donde residían ingenieros.

Como Krishnan escribió: “Un área de prueba que a menudo se pasa por alto es el proceso comercial y comunicaciones Los sistemas y procesos están altamente entrelazados y separan las pruebas de los sistemas de prueba de procesos comerciales no son realistas: una falla de un sistema comercial afectar el proceso de negocio y, a la inversa, un sistema de trabajo no es muy útil sin el personal adecuado ".

Algunos de los aprendizajes obtenidos durante estos desastres incluyen:

> Cuando se perdió la conectividad, la conmutación por error a las estaciones de trabajo del ingeniero no funcionó
>
> Los ingenieros no sabían cómo acceder a un puente de llamada de conferencia o el puente solo tenía capacidad para cincuenta personas o necesitaban un nuevo proveedor de llamadas de conferencia que permitiera ellos para iniciar ingenieros que habían sometido a toda la conferencia a celebrar música
>
> Cuando los centros de datos se quedaron sin diesel para los generadores de respaldo, nadie sabía procedimientos para realizar compras de emergencia a través del proveedor, lo que resulta en alguien usando una tarjeta de crédito personal para comprar diesel por un valor de $ 50,000.

Al crear el fracaso en una situación controlada, podemos practicar y crear los libros de jugadas que necesitar. Una de las otras salidas de Game Days es que las personas realmente saben a quién llamar y saber con quién hablar; al hacer esto, desarrollan relaciones con personas de otros departamentos para que puedan trabajar juntos durante un incidente, convirtiendo acciones conscientes en acciones inconscientes que pueden convertirse en rutina.

CONCLUSIÓN

Para crear una cultura justa que permita el aprendizaje organizacional, tenemos que volver a contextualizar Los llamados fracasos. Cuando se trata adecuadamente, los errores inherentes a sistemas complejos pueden crear un entorno de aprendizaje dinámico donde todos los accionistas se sientan lo suficientemente seguros como para presentar ideas y observaciones, y dónde los grupos se recuperan más fácilmente de proyectos que no funcionan como se esperaba.

Tanto las autopsias sin culpa como las fallas en la producción de inyecciones refuerzan una cultura que todos deben sentirse cómodos y responsables de salir a la superficie y aprender de los fracasos De hecho, cuando reducimos suficientemente el número de accidentes, disminuimos nuestro tolerancia para que podamos seguir aprendiendo. Como se sabe que Peter Senge dice: "El único

La ventaja competitiva sostenible es la capacidad de una organización para aprender más rápido que competencia."

† En enero de 2013 en re: Invent, James Hamilton, vicepresidente e ingeniero distinguido de Amazon Web Services, dijo que la región este de EE. UU. Tenía más de diez centros de datos por sí mismos, y agregó que un centro de datos típico tiene entre cincuenta mil y ochenta mil servidores. Según estas matemáticas, la interrupción del EC2 de 2011 clientes afectados en más de medio millón de servidores.

‡ Esta práctica también se ha denominado *revisiones irreprensibles posteriores al incidente* , así como *retrospectivas posteriores al evento* . También hay una notable similitud con la rutina. retrospectivas que forman parte de muchas prácticas de desarrollo iterativas y ágiles.

§ También podemos optar por extender las filosofías del tiempo de actividad transparente a nuestros informes post mortem y, además de hacer un panel de servicio disponible para el público, podemos optar por publicar (tal vez desinfectadas) reuniones post mortem para el público. Algunas de las publicaciones públicas más admiradas las mortems incluyen las publicadas por el equipo de Google App Engine después de una interrupción significativa de 2010, así como la autopsia de Amazon DynamoDB 2015 corte. Curiosamente, Chef publica sus notas de la reunión post mortem en su blog, así como videos grabados de las reuniones post mortem reales.

¶ Los patrones arquitectónicos específicos que implementaron incluyeron ayunos fallidos (estableciendo tiempos de espera agresivos para que los componentes defectuosos no hagan que todo sistema se detiene), fallos (diseño de cada característica para degradar o retroceder a una representación de menor calidad) y eliminación de características (eliminación de características críticas cuando se ejecutan lentamente desde cualquier página para evitar que afecten la experiencia del miembro). Otro ejemplo sorprendente de la la resistencia que el equipo de Netflix creó más allá de preservar la continuidad del negocio durante la interrupción de AWS, fue que Netflix pasó más de seis horas en AWS corte de energía antes de declarar un incidente de Sev 1, suponiendo que el servicio de AWS eventualmente se restablecería (es decir, "AWS volverá ... generalmente lo hace, ¿verdad?"). Solo después de seis horas de la interrupción activaron los procedimientos de continuidad comercial.

20 Convertir local Descubrimientos en Global Mejoras

En el capítulo anterior, discutimos el desarrollo de una cultura de aprendizaje segura alentando a todos para hablar sobre errores y accidentes a través de autopsias sin culpa. Nosotros también exploró encontrar y corregir señales de falla cada vez más débiles, así como reforzar y Experimentación gratificante y toma de riesgos. Además, ayudamos a hacer nuestro sistema de trabaje más resistente programando y probando de manera proactiva los escenarios de falla, haciendo que nuestro sistemas más seguros al encontrar defectos latentes y arreglarlos.

En este capítulo, crearemos mecanismos que posibiliten nuevos aprendizajes y mejoras descubiertas localmente para ser capturadas y compartidas globalmente a lo largo de todo organización, multiplicando el efecto del conocimiento global y la mejora. Al hacer esto, Elevamos el estado de la práctica de toda la organización para que todos los que trabajen se beneficia de la experiencia acumulativa de la organización.

USE SALAS DE CHAT Y CHAT BOTS PARA AUTOMATIZAR Y CAPTURA CONOCIMIENTO ORGANIZACIONAL

Muchas organizaciones han creado salas de chat para facilitar la comunicación rápida dentro equipos Sin embargo, las salas de chat también se utilizan para activar la automatización.

Esta técnica fue pionera en el viaje de ChatOps en GitHub. El objetivo era poner herramientas de automatización en el medio de la conversación en sus salas de chat, lo que ayuda a crear transparencia y documentación de su trabajo. Como Jesse Newland, ingeniero de sistemas en GitHub describe: "Incluso cuando eres nuevo en el equipo, puedes mirar en los registros de chat y mira cómo se hace todo. Es como si estuvieras programando en pareja con ellos todo el tiempo ".

Crearon *Hubot* , una aplicación de software que interactuó con el equipo de Ops en su chat

habitaciones, donde se le podría indicar que realice acciones simplemente enviándole un comando (p. ej., "@hubot despliega el búho en producción"). Los resultados también se enviarán de vuelta al sala de chat.

Hacer que este trabajo se realice por automatización en la sala de chat (en lugar de ejecutarse los scripts automatizados a través de la línea de comando) tuvieron numerosos beneficios, que incluyen:

> Todos vieron todo lo que estaba sucediendo.
>
> Los ingenieros en su primer día de trabajo pudieron ver cómo era el trabajo diario y cómo fue realizado.
>
> Las personas eran más propensas a pedir ayuda cuando veían a otros que se ayudaban entre sí.

Página 308

> El aprendizaje organizacional rápido fue habilitado y acumulado.

Además, más allá de los beneficios probados anteriormente, las salas de chat registran y crean inherentemente todas las comunicaciones públicas; en contraste, los correos electrónicos son privados por defecto, y la información en ellos no se puede descubrir o propagar fácilmente dentro de una organización.

La integración de nuestra automatización en las salas de chat ayuda a documentar y compartir nuestras observaciones. y la resolución de problemas como parte inherente de la realización de nuestro trabajo. Esto refuerza una cultura. de transparencia y colaboración en todo lo que hacemos.

Esta es también una forma extremadamente efectiva de convertir el aprendizaje local en conocimiento global. A Github, todo el personal de Operaciones trabajaba de forma remota; de hecho, no había dos ingenieros trabajando en el misma ciudad. Como Mark Imbriaco, ex vicepresidente de operaciones de GitHub, recuerda: "No hubo enfriador de agua físico en GitHub. La sala de chat era el refrigerador de agua.

Github permitió a Hubot activar sus tecnologías de automatización, incluidas Puppet, Capistrano, Jenkins, resque (una biblioteca respaldada por Redis para crear trabajos en segundo plano), y Graphme (que genera gráficos a partir de grafito).

Las acciones realizadas a través de Hubot incluyeron verificar la salud de los servicios, hacer títeres empuja o implementa los códigos en la producción, y silencia las alertas a medida que los servicios entran modo de mantenimiento Acciones que se realizaron varias veces, como levantar el registros de prueba de humo cuando falla una implementación, sacando de rotación los servidores de producción, volviendo a master para servicios front-end de producción, o incluso disculpándose con los ingenieros quienes estaban de guardia, también se convirtieron en acciones de Hubot. †

Del mismo modo, se compromete con el repositorio de código fuente y los comandos que activan Las implementaciones de producción emiten mensajes a la sala de chat. Además, como cambios moverse a través de la canalización de implementación, su estado se publica en la sala de chat.

Un típico intercambio rápido de sala de chat podría verse así:

> *"@Sr: @jnewland, ¿cómo obtienes esa lista de grandes repositorios? ¿Disk_hogs o algo así?*

"@Jnewland: / disk-hogs"

Newland observa que ciertas preguntas que se hicieron previamente durante el curso de un proyecto rara vez se preguntan ahora. Por ejemplo, los ingenieros pueden preguntarse unos a otros: "¿Cómo es eso? ¿desplegarse? o "¿Estás desplegando eso, o debería?" o "¿Cómo se ve la carga?"

Entre todos los beneficios que describe Newland, que incluyen la incorporación más rápida de nuevos ingenieros y haciendo que todos los ingenieros sean más productivos, el resultado que sintió fue más importante fue que el trabajo de Ops se volvió más humano a medida que los ingenieros de Ops pudieron descubrir problemas y ayudarse mutuamente rápida y fácilmente.

GitHub creó un entorno para el aprendizaje local colaborativo que podría transformarse en aprendizajes en toda la organización. Durante el resto de este capítulo exploraremos formas de crear y acelerar la difusión de nuevos aprendizajes organizacionales.

AUTOMATIZAR PROCESOS ESTANDARIZADOS EN SOFTWARE PARA REUTILIZAR

Con demasiada frecuencia, codificamos nuestros estándares y procesos para arquitectura, pruebas, implementación, y gestión de infraestructura en prosa, almacenándolos en documentos de Word que son subido a alguna parte. El problema es que los ingenieros que están creando nuevas aplicaciones o los entornos a menudo no saben que existen estos documentos, o no tienen tiempo para implementar los estándares documentados. El resultado es que crean sus propias herramientas y procesos, con todos los resultados decepcionantes que esperaríamos: frágil, inseguro y aplicaciones y entornos no mantenibles que son caros de ejecutar, mantener y evolucionar.

En lugar de poner nuestra experiencia en documentos de Word, necesitamos transformar estos estándares y procesos documentados, que abarcan la suma de nuestra organización aprendizajes y conocimientos, en una forma ejecutable que los hace más fáciles de reutilizar. Uno de La mejor manera de hacer que este conocimiento sea reutilizable es ponerlo en un lugar centralizado. repositorio de código fuente, haciendo que la herramienta esté disponible para que todos puedan buscar y usar.

Justin Arbuckle fue arquitecto jefe de GE Capital en 2013 cuando dijo: "Necesitábamos crear un mecanismo que permita a los equipos cumplir fácilmente con la política nacional, regulaciones regionales e industriales en docenas de marcos regulatorios, que abarcan miles de aplicaciones que se ejecutan en decenas de miles de servidores en decenas de centros de datos ".

El mecanismo que crearon se llamó ArchOps, que "permitió a nuestros ingenieros constructores, no albañiles. Al poner nuestros estándares de diseño en planos automatizados que pudimos ser utilizados fácilmente por cualquier persona, logramos la consistencia como subproducto ".

Al codificar nuestros procesos manuales en código automatizado y ejecutado, habilitamos
El proceso debe ser ampliamente adoptado, dando valor a cualquiera que los use. Arbuckle
concluyó que "el cumplimiento real de una organización está en proporción directa con el
grado en que sus políticas se expresan como código ".

Al hacer de este proceso automatizado el medio más fácil para lograr el objetivo, permitimos
prácticas para ser ampliamente adoptadas, incluso podemos considerar convertirlas en servicios compartidos
apoyado por la organización.

CREE UN CÓDIGO DE FUENTE ÚNICO COMPARTIDO REPOSITORIO PARA NUESTRA ORGANIZACIÓN COMPLETA

Un repositorio de código fuente compartido de toda la empresa es uno de los mecanismos más poderosos utilizados
integrar descubrimientos locales en toda la organización. Cuando actualizamos algo en
el repositorio de código fuente (p. ej., una biblioteca compartida), se propaga rápida y automáticamente
a cualquier otro servicio que use esa biblioteca, y se integra a través de cada equipo
tubería de despliegue.

Google es uno de los mayores ejemplos de uso de un código fuente compartido en toda la organización
repositorio. Para 2015, Google tenía un único repositorio de código fuente compartido con más de uno
mil millones de archivos y más de dos mil millones de líneas de código. Este repositorio es utilizado por cada uno de sus
veinticinco mil ingenieros y abarca todas las propiedades de Google, incluido Google
Búsqueda, Google Maps, Google Docs, Google+, Google Calendar, Gmail y YouTube. [1]

Uno de los valiosos resultados de esto es que los ingenieros pueden aprovechar la experiencia diversa de
todos en la organización. Rachel Potvin, gerente de ingeniería de Google que supervisa

Page 310

El grupo de Infraestructura de desarrolladores le dijo a *Wired* que cada ingeniero de Google puede acceder a "un
riqueza de bibliotecas "porque" casi todo ya se ha hecho ".

Además, como Eran Messeri, ingeniero del grupo de Infraestructura de desarrolladores de Google,
explica, una de las ventajas de usar un único repositorio es que permite a los usuarios
Acceda a todo el código en su forma más actualizada, sin necesidad de coordinación.

Ponemos en nuestro repositorio de código fuente compartido no solo el código fuente, sino también otros
artefactos que codifican el conocimiento y el aprendizaje, que incluyen:

- Estándares de configuración para nuestras bibliotecas, infraestructura y entornos (Chef
  recetas, manifiestos de marionetas, etc.)
- Herramientas de implementación
- Prueba de estándares y herramientas, incluida la seguridad

Herramientas de canalización de implementación

Herramientas de monitoreo y análisis.

Tutoriales y estándares

Codificar el conocimiento y compartirlo a través de este repositorio es uno de los más poderosos mecanismos que tenemos para propagar el conocimiento. Como describe Randy Shoup, "La mayoría El poderoso mecanismo para prevenir fallas en Google es el repositorio de código único. Cada vez que alguien registra algo en el repositorio, se genera una nueva compilación, que siempre usa la última versión de todo. Todo está construido desde la fuente en lugar de vinculado dinámicamente en tiempo de ejecución: siempre hay una única versión de una biblioteca que es actual en uso, que es lo que se vincula estáticamente durante el proceso de compilación ".

Tom Limoncelli es coautor de *The Practice of Cloud System Administration: Diseño y operación de grandes sistemas distribuidos* y una antigua confiabilidad del sitio Ingeniero en Google. En su libro, afirma que el valor de tener un único repositorio para Toda una organización es tan poderosa que es difícil incluso explicarla.

> Puede escribir una herramienta exactamente una vez y hacer que sea utilizable para todos los proyectos. Tienes 100% conocimiento exacto de quién depende de una biblioteca; por lo tanto, puedes refactorizarlo y ser 100% seguro de quién se verá afectado y quién debe probar la rotura. Probablemente podría enumera cien ejemplos más. No puedo expresar con palabras cuánto de competitivo ventaja esto es para Google.

En Google, todas las bibliotecas (por ejemplo, libc, OpenSSL, así como las bibliotecas desarrolladas internamente, como Bibliotecas de subprocesos de Java) tiene un propietario responsable de garantizar que la biblioteca no solo compila, pero también pasa con éxito las pruebas para todos los proyectos que dependen de él, como un bibliotecario del mundo real. Ese propietario también es responsable de migrar cada proyecto de una versión a la siguiente.

Considere el ejemplo de la vida real de una organización que ejecuta ochenta y una versiones diferentes de la biblioteca de framework Java Struts en producción; todas menos una de esas versiones tienen vulnerabilidades críticas de seguridad y mantenimiento de todas esas versiones, cada una con su propia peculiaridades e idiosincrasias, crea una carga operativa y estrés significativos. Además,

Página 311

Toda esta variación hace que las versiones de actualización sean riesgosas e inseguras, lo que a su vez desalienta desarrolladores de la actualización. Y el ciclo continúa.

El repositorio de fuente única resuelve gran parte de este problema, además de haber automatizado pruebas que permiten a los equipos migrar a nuevas versiones de forma segura y confiable.

Si no podemos construir todo a partir de un solo árbol de origen, debemos encontrar otros medios para mantener buenas versiones conocidas de las bibliotecas y sus dependencias. Por ejemplo, nosotros puede tener un repositorio de toda la organización como Nexus, Artifactory, o un Debian o RPM

repositorio, que luego debemos actualizar donde haya vulnerabilidades conocidas, tanto en estos repositorios y en sistemas de producción.

EXPANDA EL CONOCIMIENTO UTILIZANDO PRUEBAS AUTOMATIZADAS COMO DOCUMENTACIÓN Y COMUNIDADES DE PRÁCTICA

Cuando hemos compartido bibliotecas que se utilizan en toda la organización, debemos habilitar rápidamente propagación de experiencia y mejoras. Asegurar que cada una de estas bibliotecas tenga Se incluyen cantidades significativas de pruebas automatizadas, lo que significa que estas bibliotecas se vuelven autónomas documentar y mostrar a otros ingenieros cómo usarlos.

Este beneficio será casi automático si tenemos prácticas de desarrollo basadas en pruebas (TDD) en su lugar, donde se escriben las pruebas automatizadas antes de que escribamos el código. Esta disciplina gira nuestras suites de prueba en una especificación actualizada y viva del sistema. Cualquier ingeniero que lo desee para comprender cómo usar el sistema puede mirar el conjunto de pruebas para encontrar ejemplos de trabajo de cómo usar la API del sistema.

Idealmente, cada biblioteca tendrá un único propietario o un único equipo que lo respalde, lo que representa donde reside el conocimiento y la experiencia para la biblioteca. Además, deberíamos (idealmente) solo permite que se use una versión en producción, asegurando que todo lo que esté en producción aprovecha el mejor conocimiento colectivo de la organización.

En este modelo, el propietario de la biblioteca también es responsable de migrar de forma segura cada grupo utilizando el repositorio de una versión a la siguiente. Esto a su vez requiere una detección rápida de errores de regresión a través de pruebas automatizadas integrales e integración continua para todos los sistemas que usan la biblioteca.

Para propagar más rápidamente el conocimiento, también podemos crear grupos de discusión o salas de chat para cada biblioteca o servicio, para que cualquier persona que tenga preguntas pueda obtener respuestas de otros usuarios, que a menudo responden más rápido que los desarrolladores.

Mediante el uso de este tipo de herramienta de comunicación en lugar de tener focos de experiencia aislados distribuidos por toda la organización, facilitamos un intercambio de conocimientos y experiencia, asegurando que los trabajadores puedan ayudarse mutuamente con problemas y nuevas patrones.

DISEÑO PARA OPERACIONES A TRAVÉS DE CODIFICADOS NO REQUERIMIENTOS FUNCIONALES

Cuando el desarrollo sigue su trabajo río abajo y participa en la producción actividades de resolución de incidentes, la aplicación está cada vez mejor diseñada para

Operaciones Además, a medida que comenzamos a diseñar nuestro código y aplicación deliberadamente que puede acomodar el flujo rápido y la capacidad de implementación, es probable que identifiquemos un conjunto de requisitos funcionales que queremos integrar en todos nuestros servicios de producción.

La implementación de estos requisitos no funcionales permitirá que nuestros servicios sean fáciles de implementar y seguir funcionando en producción, donde podemos detectar y corregir rápidamente problemas y asegúrese de que se degrada con gracia cuando fallan los componentes. Ejemplos de no Los requisitos funcionales incluyen garantizar que tenemos:

- Telemetría de producción suficiente en nuestras aplicaciones y entornos.
- La capacidad de rastrear dependencias con precisión
- Servicios que son resistentes y se degradan con gracia
- Compatibilidad hacia adelante y hacia atrás entre versiones
- La capacidad de archivar datos para administrar el tamaño del conjunto de datos de producción
- La capacidad de buscar y comprender fácilmente los mensajes de registro en todos los servicios
- La capacidad de rastrear solicitudes de usuarios a través de múltiples servicios
- Configuración de tiempo de ejecución simple y centralizada mediante indicadores de funciones, etc.

Al codificar este tipo de requisitos no funcionales, lo hacemos más fácil para todos nuestros nuevos y servicios existentes para aprovechar el conocimiento colectivo y la experiencia de la organización. Estas son todas las responsabilidades del equipo que construye el servicio.

CONSTRUIR HISTORIAS DE USUARIOS DE OPERACIONES REUTILIZABLES EN DESARROLLO

Cuando hay un trabajo de Operaciones que no puede ser totalmente automatizado o hecho autoservicio, nuestro El objetivo es hacer que este trabajo recurrente sea lo más repetible y determinista posible. Nosotros hacemos esto estandarizando el trabajo necesario, automatizando tanto como sea posible y documentando nuestro trabaje para que podamos habilitar mejor a los equipos de productos para planificar y utilizar mejor esta actividad.

En lugar de construir servidores manualmente y luego ponerlos en producción de acuerdo con listas de verificación manuales, debemos automatizar la mayor cantidad posible de este trabajo. Donde cierto los pasos no se pueden automatizar (p. ej., instalar manualmente un servidor y tener otro cable de equipo), deberíamos definir colectivamente las transferencias lo más claramente posible para reducir los plazos de entrega y errores Esto también nos permitirá planificar y programar mejor estos pasos en el futuro. Por ejemplo, podemos usar herramientas como Rundeck para automatizar y ejecutar flujos de trabajo, o sistemas de tickets de trabajo como JIRA o ServiceNow.

Idealmente, para todo nuestro trabajo recurrente Ops sabremos lo siguiente: qué trabajo se requiere, quién se necesita para realizarlo, cuáles son los pasos para completarlo, y así sucesivamente. Por ejemplo, "Sabemos que una implementación de alta disponibilidad requiere catorce pasos, que requieren trabajo de cuatro

diferentes equipos, y las últimas cinco veces que realizamos esto, tomó un promedio de tres dias."

Del mismo modo que creamos historias de usuarios en Desarrollo que colocamos en la cartera y luego extraemos en el trabajo, podemos crear "historias de usuario de Ops" bien definidas que representen actividades de trabajo que se puede reutilizar en todos nuestros proyectos (por ejemplo, implementación, capacidad, seguridad, etc.). Por creando estas historias de usuario de Ops bien definidas, exponemos el trabajo repetido de operaciones de TI en un manera en que aparece junto con el trabajo de desarrollo, lo que permite una mejor planificación y Resultados más repetibles.

ASEGURAR QUE LAS OPCIONES DE TECNOLOGÍA AYUDEN A ALCANZAR METAS ORGANIZACIONALES

Cuando uno de nuestros objetivos es maximizar la productividad del desarrollador y tenemos servicios orientados arquitecturas, pequeños equipos de servicio pueden potencialmente construir y ejecutar su servicio en cualquier lenguaje o marco que mejor satisfaga sus necesidades específicas. En algunos casos, esto es lo que Lo mejor nos permite alcanzar nuestras metas organizacionales.

Sin embargo, hay situaciones en las que ocurre lo contrario, como cuando la experiencia para un el servicio crítico reside solo en un equipo, y solo ese equipo puede hacer cambios o arreglar problemas, creando un cuello de botella. En otras palabras, podemos haber optimizado para el equipo productividad pero sin darse cuenta impidió el logro de los objetivos de la organización.

Esto sucede a menudo cuando tenemos un grupo de operaciones funcionalmente orientado que es responsable de cualquier aspecto del servicio de soporte. En estos escenarios, para garantizar que habilitamos los profundos conjuntos de habilidades en tecnologías específicas, queremos asegurarnos de que las operaciones puedan influir en qué componentes se utilizan en la producción, o darles la capacidad de no ser responsable de plataformas no compatibles.

Si no tenemos una lista de tecnologías que admitirá Operations, generadas colectivamente por Desarrollo y Operaciones, debemos pasar sistemáticamente por la producción infraestructura y servicios, así como todas sus dependencias que se admiten actualmente, para encontrar cuáles están creando una cantidad desproporcionada de demanda de fallas y trabajo no planificado Nuestro objetivo es identificar las tecnologías que:

- Impedir o ralentizar el flujo de trabajo
- Crear desproporcionadamente altos niveles de trabajo no planificado
- Crear desproporcionadamente un gran número de solicitudes de soporte
- Son más inconsistentes con nuestros resultados arquitectónicos deseados (por ejemplo, rendimiento, estabilidad, seguridad, confiabilidad, continuidad del negocio)

Al eliminar estas infraestructuras y plataformas problemáticas de las tecnologías con el apoyo de Ops, les permitimos centrarse en la infraestructura que mejor ayuda a lograr el objetivos globales de la organización.

Como describe Tom Limoncelli, "cuando estaba en Google, teníamos un compilado oficial

idioma, un idioma oficial de secuencias de comandos y un idioma oficial de interfaz de usuario. Sí, otro

Página 314

los idiomas eran compatibles de una forma u otra, pero quedarse con 'los tres grandes' significaba admite bibliotecas, herramientas y una forma más fácil de encontrar colaboradores ".§ Estas normas fueron también reforzado por el proceso de revisión de código, así como también qué idiomas fueron compatibles con sus plataformas internas

En una presentación que dio con Olivier Jacques y Rafael García en el DevOps 2015 Enterprise Summit, Ralph Loura, CIO de HP, declaró:

> Internamente, describimos nuestro objetivo como crear "boyas, no límites". En lugar de trazando límites difíciles en los que todos deben permanecer, colocamos boyas que indican áreas profundas del canal donde estás seguro y apoyado. Puedes pasar las boyas siempre y cuando sigas los principios organizacionales. Después de todo, ¿cómo vamos a vea la próxima innovación que nos ayuda a ganar si no estamos explorando y probando en el bordes? Como líderes, debemos navegar por el canal, marcarlo y permitir personas para explorar más allá.

En muchas organizaciones que adoptan DevOps, una historia común que los desarrolladores cuentan es: "Ops no nos proporcionaron lo que necesitábamos, así que solo lo construimos y lo apoyamos nosotros mismos ". Sin embargo, en las primeras etapas de la transformación de Etsy, el liderazgo tecnológico tomó El enfoque opuesto, que reduce significativamente la cantidad de tecnologías compatibles en producción.

En 2010, después de una temporada de vacaciones pico casi desastrosa, el equipo de Etsy decidió reducir masivamente la cantidad de tecnologías utilizadas en la producción, eligiendo algunas que toda la organización podría apoyar completamente y erradicar el resto. ¶

Su objetivo era estandarizar y reducir deliberadamente el apoyo infraestructura y configuraciones. Una de las primeras decisiones fue migrar a Etsy plataforma completa para PHP y MySQL. Esta fue principalmente una decisión filosófica en lugar de una decisión tecnológica: querían que tanto Dev como Ops pudieran entender la pila de tecnología completa para que todos puedan contribuir a una sola plataforma, así como permitir que todos puedan leer, reescribir y arreglar el uno al otro código. En los próximos años, como Michael Rembetsy, quien fue Director de Etsy Las operaciones en ese momento, recuerda, “Retiramos algunas tecnologías excelentes, tomándolas completamente fuera de producción ", incluidos lighttpd, Postgres, MongoDB, Scala, CoffeeScript, Python y muchos otros.

Del mismo modo, Dan McKinley, desarrollador del equipo de funciones que presentó MongoDB en Etsy en 2010, escribe en su blog que todos los beneficios de tener un esquema sin la base de datos fue negada por todos los problemas operativos que el equipo tuvo que resolver.

Estos incluyeron problemas relacionados con el registro, la representación gráfica, el monitoreo y la producción.
telemetría y copias de seguridad y restauración, así como muchos otros problemas que
Los desarrolladores generalmente no necesitan preocuparse. El resultado fue
abandonar MongoDB, portar el nuevo servicio para usar el MySQL ya compatible
infraestructura de base de datos.

CONCLUSIÓN

Página 315

Las técnicas descritas en este capítulo permiten incorporar cada nuevo aprendizaje.
en el conocimiento colectivo de la organización, multiplicando su efecto. Hacemos esto por
Comunicar de manera activa y amplia nuevos conocimientos, como a través de salas de chat y
a través de tecnología como arquitectura como código, repositorios de código fuente compartido,
estandarización tecnológica, y así sucesivamente. Al hacer esto, elevamos el estado de la práctica.
no solo de Dev y Ops, sino también de toda la organización, para que todos los que realizan el trabajo
lo hace con la experiencia acumulativa de toda la organización.

† Hubot a menudo realizaba tareas llamando a scripts de shell, que luego se podían ejecutar desde la sala de chat en cualquier lugar, incluso desde el teléfono de un ingeniero.

‡ Los proyectos de Chrome y Android residen en un repositorio de código fuente separado, y ciertos algoritmos que se mantienen en secreto, como PageRank, son disponible solo para ciertos equipos.

§ Google usó C ++ como su lenguaje compilado oficial, Python (y más tarde Go) como su lenguaje de scripting oficial, y Java y JavaScript a través de Google Web Kit de herramientas como sus idiomas oficiales de UI.

¶ En ese momento, Etsy usaba PHP, lighttp, Postgres, MongoDB, Scala, CoffeeScript, Python, así como muchas otras plataformas e idiomas.

21 Reserve tiempo para crear Aprendizaje organizacional y mejora

Una de las prácticas que forma parte del sistema de producción de Toyota se llama *bombardeo de mejora* (o a veces un *bombardeo de Kaizen*), definido como un dedicado y período de tiempo concentrado para abordar un problema en particular, a menudo en el transcurso de un varios dias. El Dr. Spear explica: "... los bombardeos a menudo toman esta forma: un grupo se reúne para centrarse intensamente en un proceso con problemas ... El bombardeo dura unos días, el objetivo es mejora de procesos, y los medios son el uso concentrado de personas externas el proceso para asesorar a aquellos que normalmente están dentro del proceso ".

Spear observa que la salida del equipo de mejora de bombardeos a menudo será un nuevo enfoque para resolver un problema, como nuevos diseños de equipos, nuevos medios de transmitir material e información, un espacio de trabajo más organizado o estandarizado trabajo. También pueden dejar una lista de tareas pendientes que se realizarán en el futuro.

Un ejemplo de bombardeo de mejora DevOps es el programa Reto mensual en el Objetivo DevOps Dojo. Ross Clanton, Director de Operaciones en Target, es responsable de acelerando la adopción de DevOps. Uno de sus mecanismos principales para esto es el

Centro de Innovación Tecnológica, más popularmente conocido como DevOps Dojo.
El DevOps Dojo ocupa aproximadamente dieciocho mil pies cuadrados de espacio de oficina abierto, donde los entrenadores de DevOps ayudan a los equipos de toda la organización de tecnología Target elevar el estado de su práctica. El formato más intensivo es lo que llaman "30 días Desafíos ", donde los equipos de desarrollo interno entran por un mes y trabajan juntos con entrenadores e ingenieros de Dojo dedicados. El equipo trae su trabajo con ellos, con el objetivo de resolver un problema interno con el que han estado luchando y para crear un avance en treinta días

Durante los treinta días, trabajan intensamente con los entrenadores de Dojo en el problema: planificar, trabajar y hacer demostraciones en sprints de dos días. Cuando los 30 días El desafío está completo, los equipos internos vuelven a sus líneas de negocio, no solo habiendo resuelto un problema importante, pero volviendo a sus nuevos aprendizajes equipos

Clanton describe: "Actualmente tenemos la capacidad de tener ocho equipos haciendo 30 días Desafíos simultáneos, por lo que estamos enfocados en los proyectos más estratégicos de la organización. Hasta ahora, hemos tenido algunas de nuestras capacidades más críticas a través de Dojo, incluidos los equipos de punto de venta (POS), inventario, precios y promoción ".

Al asignar personal Dojo a tiempo completo y centrarse en un solo objetivo, los equipos pasar un desafío de 30 días hace mejoras increíbles.

Página 318

Ravi Pandey, gerente de desarrollo de Target que realizó este programa, explica:
“En los viejos tiempos, tendríamos que esperar seis semanas para obtener un entorno de prueba. Ahora nosotros obténgalo en minutos, y estamos trabajando codo a codo con los ingenieros de Ops que nos están ayudando aumentar nuestra productividad y herramientas de construcción para ayudarnos a alcanzar nuestros objetivos ".
Clanton amplía esta idea: "No es raro que los equipos logren en días lo que generalmente les llevaría de tres a seis meses. Hasta ahora, han llegado doscientos alumnos a través del Dojo, habiendo completado catorce desafíos ".

Dojo también admite modelos de interacción menos intensivos, incluidas las compilaciones Flash, donde los equipos se unen para eventos de uno a tres días, con el objetivo de enviar un producto viable mínimo (MVP) o una capacidad al final del evento. También alojan Open Labs cada dos semanas, donde cualquiera puede visitar el Dojo para hablar con el Dojo entrenadores, asistir a demostraciones o recibir capacitación.

En este capítulo, describiremos esta y otras formas de reservar tiempo para la organización aprendizaje y mejora, institucionalizando aún más la práctica de dedicar tiempo a mejorando el trabajo diario.

INSTITUCIONALIZAR RITUALES PARA PAGAR TÉCNICOS

En esta sección, programamos rituales que ayudan a reforzar la práctica de reservar Dev y
Tiempo de operación para el trabajo de mejora, como requisitos no funcionales, automatización, etc.
Una de las formas más fáciles de hacer esto es programar y llevar a cabo el día o la semana.
bombardeos de mejora, donde todos en un equipo (o en toda la organización)
se organiza para solucionar los problemas que les interesan; no se permite el trabajo de características. Podría ser un
área problemática del código, entorno, arquitectura, herramientas, etc. Estas
los equipos abarcan todo el flujo de valor, a menudo combinando Desarrollo, Operaciones y
Ingenieros de Infosec. Los equipos que generalmente no trabajan juntos combinan sus habilidades y
esfuerzo para mejorar un área elegida y luego demostrar su mejora al resto de
la compañia.

Además de los términos orientados a Lean kaizen blitz y mejora blitz, el
La técnica de rituales dedicados para el trabajo de mejora también se ha llamado *primavera* u *otoño*
limpiezas y *entradas de cola semanas de inversión* . También se han utilizado otros términos, como
días de pirateo , *hackatones* y *20% de tiempo de innovación* . Desafortunadamente, estos rituales específicos
a veces se centran en la innovación de productos y la creación de prototipos de nuevas ideas de mercado, en lugar de
en el trabajo de mejora, y lo que es peor, a menudo se limitan a los desarrolladores, que es
considerablemente diferente a los objetivos de un bombardeo de mejora.†

Nuestro objetivo durante estos bombardeos no es simplemente experimentar e innovar por el bien de
probar nuevas tecnologías, pero para mejorar nuestro trabajo diario, como resolver nuestro trabajo diario
soluciones alternativas. Mientras que los experimentos también pueden conducir a mejoras, la mejora bombardea
estamos muy enfocados en resolver problemas específicos que encontramos en nuestro trabajo diario.

Podemos programar bombardeos de mejora de una semana que prioricen el trabajo de Dev y Ops
juntos hacia objetivos de mejora. Estos bombardeos de mejora son simples de

Página 319

administrar: se selecciona una semana donde trabajan todos los miembros de la organización tecnológica
en una actividad de mejora al mismo tiempo. Al final del período, cada equipo
hace una presentación a sus compañeros que discute el problema que estaban abordando y
lo que construyeron Esta práctica refuerza una cultura en la que los ingenieros trabajan en todo el
flujo de valor completo para resolver problemas. Además, refuerza la solución de problemas a medida que
parte de nuestro trabajo diario y demuestra que valoramos pagar la deuda técnica.

Lo que hace que los bombardeos de mejora sean tan poderosos es que estamos capacitando a los más cercanos
al trabajo para identificar y resolver continuamente sus propios problemas. Considerar para un
momento en que nuestro complejo sistema es como una telaraña, con hilos entrelazados que
están constantemente debilitándose y rompiéndose. Si se rompe la combinación correcta de hilos, el
toda la web se derrumba. No hay una cantidad de gestión de comando y control que

puede ordenar a los trabajadores que arreglen cada capítulo uno por uno. En cambio, debemos crear el cultura organizacional y normas que llevan a todos a encontrar y arreglar continuamente hebras rotas como parte de nuestro trabajo diario. Como observa el Dr. Spear: "No es de extrañar entonces que las arañas reparan rasgaduras y rasgaduras en la red a medida que ocurren, sin esperar a que las fallas acumular."

Mark describe un gran ejemplo del éxito del concepto de mejora blitz Zuckerberg, CEO de Facebook. En una entrevista con Jessica Stillman de Inc.com, dice: "Cada pocos meses tenemos un hackathon, donde todos construyen prototipos para nuevos ideas que tienen. Al final, todo el equipo se reúne y mira todo lo que ha sido construido. Muchos de nuestros productos más exitosos salieron de hackatones, incluyendo Timeline, chat, video, nuestro marco de desarrollo móvil y algunos de nuestros infraestructura más importante como el compilador HipHop ".

De particular interés es el compilador PHP HipHop. En 2008, Facebook se enfrentaba problemas de capacidad significativos, con más de cien millones de usuarios activos y rápidamente creciendo, creando tremendos problemas para todo el equipo de ingeniería. Durante un hack día, Haiping Zhao, ingeniero senior de servidores en Facebook, comenzó a experimentar con convertir código PHP a código C ++ compilable, con la esperanza de aumentar significativamente la capacidad de su infraestructura existente. Durante los siguientes dos años, un pequeño equipo fue ensamblado para construir lo que se conoció como el compilador HipHop, convirtiendo todos Servicios de producción de Facebook desde PHP interpretado hasta binarios compilados de C ++. Hip hop permitió que la plataforma de Facebook manejara cargas de producción seis veces más altas que las PHP nativo.

En una entrevista con Cade Metz de *Wired* , Drew Paroski, uno de los ingenieros que trabajó en el proyecto y señaló: "Hubo un momento en el que, si HipHop no hubiera estado allí, hubiéramos estado en agua caliente. Probablemente hubiéramos necesitado más máquinas para servir al sitio de lo que podríamos haber llegado a tiempo. Fue un pase de Ave María eso funcionó ".

Más tarde, Paroski y sus colegas ingenieros Keith Adams y Jason Evans decidieron que podría superar el rendimiento del compilador HipHop y reducir algunos de sus limitaciones que redujeron la productividad del desarrollador. El proyecto resultante fue el HipHop proyecto de máquina virtual ("HHVM"), adoptando un enfoque de compilación justo a tiempo. Por

2012, HHVM había reemplazado completamente el compilador HipHop en producción, con casi veinte ingenieros que contribuyen al proyecto.

Al realizar bombardeos de mejora programados regularmente y piratear semanas, permitimos todos en la cadena de valor para enorgullecerse y apropiarse de las innovaciones que creamos, e integramos continuamente mejoras en nuestro sistema, permitiendo aún más

seguridad, confiabilidad y aprendizaje.

HABILITAR A TODOS PARA ENSEÑAR Y APRENDER

Una cultura dinámica de aprendizaje crea condiciones para que todos no solo puedan aprender, pero también enseñar, ya sea a través de métodos didácticos tradicionales (por ejemplo, personas que toman clases, asistir a capacitación) o más métodos experimentales o abiertos (por ejemplo, conferencias, talleres, tutorías). Una forma de fomentar esta enseñanza y aprendizaje es dedicarle tiempo organizacional.

Steve Farley, vicepresidente de tecnología de la información en Nationwide Insurance, dijo: "Tenemos cinco mil profesionales de la tecnología, a los que llamamos 'asociados'. Desde 2011, tenemos comprometido a crear una cultura de aprendizaje, parte de eso es algo que llamamos Jueves de enseñanza, donde cada semana creamos tiempo para que nuestros asociados aprendan. Para dos horas, se espera que cada asociado enseñe o aprenda. Los temas son cualesquiera que sean nuestros los asociados quieren aprender, algunos de ellos están en tecnología, en software nuevo técnicas de desarrollo o mejora de procesos, o incluso sobre cómo gestionar mejor su carrera Lo más valioso que cualquier asociado puede hacer es ser mentor o aprender de otros asociados ".

Como se ha hecho evidente a lo largo de este libro, ciertas habilidades se están volviendo cada vez más lo necesitan todos los ingenieros, no solo los desarrolladores. Por ejemplo, es Cada vez es más importante que todos los ingenieros de operaciones y pruebas estén familiarizados con Técnicas de desarrollo, rituales y habilidades, como control de versiones, pruebas automatizadas, tuberías de implementación, gestión de configuración y creación de automatización. La familiaridad con las técnicas de desarrollo ayuda a los ingenieros de operaciones a seguir siendo relevantes A medida que más flujos de valor tecnológico adopten los principios y patrones de DevOps.

Aunque la posibilidad de aprender algo nuevo puede ser intimidante o tener sentido de vergüenza o vergüenza, no debería. Después de todo, todos somos aprendices de por vida, y uno Una de las mejores maneras de aprender es de nuestros compañeros. Karthik Gaekwad, quien fue parte de la La transformación de DevOps de National Instruments dijo: "Para las personas de operaciones que son tratando de aprender la automatización, no debería dar miedo, solo pregúntele a un desarrollador amigable, porque les encantaría ayudar ".

Podemos ayudar más a enseñar habilidades a través de nuestro trabajo diario mediante la ejecución conjunta de código revisiones que incluyen a ambas partes para que podamos aprender haciendo, así como al tener Desarrollo y Operaciones trabajan juntos para resolver pequeños problemas. Por ejemplo, nosotros podría hacer que Development muestre a Operations cómo autenticar una aplicación, y Inicie sesión y ejecute pruebas automatizadas en varias partes de la aplicación para asegurarse de que los componentes críticos funcionan correctamente (p. ej., funcionalidad clave de la aplicación, base de datos

transacciones, colas de mensajes). Luego integraríamos esta nueva prueba automatizada en nuestro canal de implementación y ejecutarlo periódicamente, enviando los resultados a nuestro monitoreo y sistemas de alerta para que podamos obtener una detección más temprana cuando fallan los componentes críticos.

Como Glenn O'Donnell de Forrester Research bromeó en su DevOps Enterprise 2014 Presentación de la cumbre, "Para todos los profesionales de la tecnología que aman innovar, amar cambio, hay un futuro maravilloso y vibrante por delante ".

COMPARTE TUS EXPERIENCIAS DE DEVOPS
Conferencias

En muchas organizaciones centradas en los costos, los ingenieros a menudo se desaniman de asistir conferencias y aprendiendo de sus compañeros. Para ayudar a construir una organización de aprendizaje, nosotros debería alentar a nuestros ingenieros (tanto de Desarrollo como de Operaciones) a asistir conferencias, dar charlas en ellos y, cuando sea necesario, crear y organizar interna o conferencias externas en sí.

DevOpsDays sigue siendo una de las series de conferencias autoorganizadas más vibrantes de la actualidad. Muchas prácticas de DevOps se han compartido y promulgado en estos eventos. Tiene permaneció libre o casi libre, con el apoyo de una vibrante comunidad de profesionales comunidades y vendedores.

La Cumbre empresarial de DevOps se creó en 2014 para que los líderes tecnológicos compartan sus experiencias adoptando principios y prácticas de DevOps en grandes y complejos organizaciones. El programa se organiza principalmente en torno a informes de experiencia de líderes tecnológicos en el viaje de DevOps, así como expertos en la materia sobre temas seleccionado por la comunidad.

Además de asistir a conferencias externas, muchas empresas, incluidas aquellas descrito en esta sección, tener conferencias internas para su tecnología empleados.

Nationwide Insurance es un proveedor líder de seguros y servicios financieros, y opera en industrias muy reguladas. Sus muchas ofertas incluyen auto y seguros de vivienda, y son el principal proveedor de servicios públicos planes de jubilación y seguro de mascotas. A partir de 2014, $ 195 mil millones en activos, con $ 24 mil millones en ingresos. Desde 2005, Nationwide ha estado adoptando Agile and Lean principios para elevar el estado de práctica de sus cinco mil tecnologías profesionales, permitiendo la innovación de base.

Steve Farley, vicepresidente de tecnología de la información, recuerda: "Tecnología emocionante Las conferencias comenzaban a aparecer alrededor de esa época, como la Agile national conferencia. En 2011, el liderazgo tecnológico en Nationwide acordó que nosotros debería crear una conferencia tecnológica, llamada TechCon. Al celebrar este evento, Queríamos crear una mejor manera de enseñarnos a nosotros mismos, así como asegurarnos de que

todo tenía un contexto nacional, en lugar de enviar a todos a un
conferencia externa ".

Capital One, uno de los bancos más grandes de los EE. UU. Con más de $ 298 mil millones en activos y $ 24 mil millones en ingresos en 2015, celebraron su primera ingeniería interna de software conferencia en 2015 como parte de su objetivo de crear una tecnología de clase mundial organización. La misión era promover una cultura de intercambio y colaboración, y para construir relaciones entre los profesionales de la tecnología y permitir aprendizaje. La conferencia tuvo trece pistas de aprendizaje y cincuenta y dos sesiones, y Asistieron más de 1.200 empleados internos.

Dr. Tapabrata Pal, miembro técnico de Capital One y uno de los organizadores de el evento describe, "Incluso tuvimos una sala de exposiciones, donde tuvimos veintiocho cabinas, donde los equipos internos de Capital One mostraban toda la increíble capacidades en las que estaban trabajando. Incluso decidimos muy deliberadamente que allí no habría vendedores allí, porque queríamos mantener el foco en Capital One metas."

Target es el sexto minorista más grande de los EE. UU., Con $ 72 mil millones en ingresos en 2014 y 1,799 tiendas minoristas y 347,000 empleados en todo el mundo. Heather Mickman, una director de desarrollo, y Ross Clanton han celebrado seis DevOpsDays internos eventos desde 2014 y tienen más de 975 seguidores dentro de su tecnología interna comunidad, inspirada en los DevOpsDays celebrados en ING en Amsterdam en 2013.

Después de que Mickman y Clanton asistieron a la Cumbre empresarial de DevOps en 2014, celebraron su propia conferencia interna, invitando a muchos oradores de empresas externas para que puedan recrear su experiencia para sus superiores liderazgo. Clanton describe: "2015 fue el año en que obtuvimos el poder ejecutivo atención y cuando creamos impulso. Después de ese evento, vinieron toneladas de personas depende de nosotros, preguntando cómo pueden involucrarse y cómo pueden ayudar ".

CREAR CONSULTORÍA INTERNA Y ENTRENADORES PARA PRÁCTICAS DE DIFUSIÓN

La creación de una organización interna de coaching y consultoría es un método comúnmente utilizado para difundir la experiencia en una organización. Esto puede venir en muchas formas diferentes. A Capital One, expertos en la materia designados tienen horarios de oficina donde cualquiera puede consultar con ellos, hacer preguntas, etc.

Anteriormente en el libro, comenzamos la historia de cómo el Testing Grouplet construyó una clase mundial cultura de pruebas automatizadas en Google a partir de 2005. Su historia continúa aquí, como intentan mejorar el estado de las pruebas automatizadas en todo Google utilizando bombardeos de mejora dedicados, entrenadores internos e incluso una certificación interna

programa.
Bland dijo que, en ese momento, había una política de tiempo de innovación del *20%* en Google, lo que permitía los desarrolladores gastan aproximadamente un día por semana en un proyecto relacionado con Google fuera de Su principal área de responsabilidad. Algunos ingenieros optaron por formar *grupos* , ad hoc

Página 323

equipos de ingenieros de ideas afines que querían unir su 20% de tiempo, lo que les permite hacer bombardeos de mejora enfocada.

Bharat Mediratta y Nick Lesiecki formaron un grupo de prueba con la misión. de impulsar la adopción de pruebas automatizadas en Google. A pesar de que no tenían presupuesto o autoridad formal, como describió Mike Bland: "No hubo restricciones impuestas sobre nosotros, tampoco. Y nos aprovechamos de eso ".

Utilizaron varios mecanismos para impulsar la adopción, pero uno de los más famosos fue *Pruebas en el inodoro* (o TotT), su prueba semanal periódica. Cada semana, ellos publicó un boletín informativo en casi todos los baños en casi todas las oficinas de Google en todo el mundo. Bland dijo: "El objetivo era aumentar el grado de prueba de conocimiento y sofisticación en toda la empresa. Es dudoso una publicación solo en línea habría involucrado a las personas en el mismo grado ".

Bland continúa: "Uno de los episodios de TotT más significativos fue el titulado 'Prueba Certificado: nombre pésimo, excelentes resultados 'porque describió dos iniciativas que tenían éxito significativo en el avance del uso de pruebas automatizadas ".

Test Certified (TC) proporcionó una hoja de ruta para mejorar el estado de las pruebas automatizadas. Como Bland describe: "Tenía la intención de piratear las prioridades centradas en la medición de Google cultura ... y superar el primer obstáculo aterrador de no saber dónde o cómo comenzar. El nivel 1 era establecer rápidamente una métrica de referencia, el nivel 2 establecía una política y alcanzar una meta de cobertura de prueba automatizada, y el Nivel 3 se esforzaba por alcanzar un largo plazo objetivo de cobertura ".

La segunda capacidad era proporcionar mentores certificados de prueba a cualquier equipo que quisiera consejos o ayuda, y Test Mercenaries (es decir, un equipo de tiempo completo de entrenadores internos y consultores) para trabajar con equipos para mejorar sus prácticas de prueba y código calidad. Los mercenarios lo hicieron aplicando los conocimientos, las herramientas del grupo de prueba, y técnicas para el propio código de un equipo, utilizando TC como guía y como objetivo. Soso era finalmente, un líder de Testing Grouplet de 2006 a 2007, y miembro del Prueba de mercenarios de 2007 a 2009.

Bland continúa: "Nuestro objetivo era llevar a todos los equipos a TC Nivel 3, ya sea que estuvieran inscrito en nuestro programa nuestro no. También colaboramos estrechamente con las pruebas internas. equipos de herramientas, proporcionando comentarios mientras abordamos los desafíos de prueba con el producto equipos Fuimos botas en el suelo, aplicando las herramientas que construimos, y eventualmente, nosotros

pudimos eliminar 'No tengo tiempo para probar' como una excusa legítima ".
Él continúa: "Los niveles de CT explotaron la cultura basada en métricas de Google: los tres
los niveles de prueba eran algo de lo que la gente podía hablar y presumir en
tiempo de revisión de desempeño. El grupo de prueba finalmente obtuvo fondos para la prueba
Mercenarios, un equipo de consultores internos a tiempo completo. Este fue un importante
paso, porque ahora la administración estaba completamente integrada, no con edictos, sino por
fondos."

Otra construcción importante fue el aprovechamiento de la mejora "Fixit" en toda la empresa
bombardeos Bland describe a Fixits como "cuando los ingenieros comunes tienen una idea y un sentido de
misión reclutar toda la ingeniería de Google para sprints intensivos de un día de reforma de código

Página 324

y adopción de herramientas ". Organizó cuatro Fixits para toda la empresa, dos Fixits puros de prueba
y dos que estaban más relacionadas con las herramientas, la última con más de cien
Voluntarios en más de veinte oficinas en trece países. También dirigió el Fixit Grouplet
De 2007 a 2008.

Estos Fixits, como describe Bland, significa que debemos proporcionar misiones enfocadas en
puntos críticos en el tiempo para generar entusiasmo y energía, lo que ayuda a avanzar
lo último. Esto ayudará a que la misión de cambio cultural a largo plazo alcance una nueva
meseta con cada gran esfuerzo visible.

Los resultados de la cultura de prueba son evidentes en los sorprendentes resultados que Google tiene
logrado, presentado a lo largo del libro.

CONCLUSIÓN

Este capítulo describe cómo podemos instituir rituales que ayuden a reforzar la cultura que
todos somos aprendices de por vida y valoramos la mejora del trabajo diario sobre el día a día
trabajar en sí. Hacemos esto reservando tiempo para pagar la deuda técnica, crear foros que
Permitir que todos aprendan y se enseñen unos a otros, tanto dentro de nuestra organización como
fuera de ella Y ponemos a disposición expertos para ayudar a los equipos internos, ya sea mediante coaching o
consultoría o incluso solo horas de oficina para responder preguntas.

Al hacer que todos se ayuden unos a otros a aprender en nuestro trabajo diario, superamos el
competencia, ayudándonos a ganar en el mercado. Pero también nos ayudamos mutuamente a lograr nuestro
pleno potencial como seres humanos.

CONCLUSIÓN A LA PARTE V

A lo largo de la Parte V, exploramos las prácticas que ayudan a crear una cultura de aprendizaje y experimentación en su organización. Aprendiendo de incidentes, creando compartidos repositorios, y compartir aprendizajes es esencial cuando trabajamos en sistemas complejos, ayudando a que nuestra cultura laboral sea más justa y nuestros sistemas más seguros y resistentes.

En la Parte VI, exploraremos cómo extender el flujo, la retroalimentación y el aprendizaje y experimentación usándolos para ayudarnos simultáneamente a lograr nuestra información Objetivos de seguridad.

† De aquí en adelante, los términos "semana de pirateo" y "hackathon" se usan indistintamente con "bombardeo de mejora", y no en el contexto de "puedes trabajar en lo que quieras ".

‡ Por cierto, el primer evento DevOpsDays interno de Target se modeló después de los primeros DevOpsDays de ING que fue organizado por Ingrid Algra, Jan-Joost Bouwman, Evelijn Van Leeuwen y Kris Buytaert en 2013, después de que parte del equipo de ING asistió a los Paris DevOpsDays 2013.

22 Seguridad de la información como El trabajo de todos, todos los días

Una de las principales objeciones a la implementación de los principios y patrones de DevOps ha sido, "El cumplimiento y la seguridad de la información no nos permitirán". Y, sin embargo, DevOps puede ser uno de los mejores formas de integrar mejor la seguridad de la información en el trabajo diario de todos en el flujo de valor tecnológico.

Cuando Infosec se organiza como un silo fuera de Desarrollo y Operaciones, muchos surgen problemas. James Wickett, uno de los creadores de la herramienta de seguridad Gauntlt y organizador de DevOpsDays Austin y la conferencia Lonestar Application Security,

observado:

> Una interpretación de DevOps es que surgió de la necesidad de habilitar a los desarrolladores productividad, porque a medida que crecía el número de desarrolladores, no había suficientes operaciones personas para manejar todo el trabajo de implementación resultante. Esta escasez es aún peor en Infosec: la proporción de ingenieros en Desarrollo, Operaciones e Infosec en un La organización tecnológica es 100: 10: 1. Cuando Infosec es superado en número, sin automatización e integración de la seguridad de la información en el trabajo diario de Dev y Ops, Infosec solo puede hacer verificaciones de cumplimiento, que es lo opuesto a la ingeniería de seguridad Y además, también hace que todos nos odien.

James Wickett y Josh Corman, ex CTO de Sonatype e información respetada investigador de seguridad, ha escrito sobre la incorporación de objetivos de seguridad de la información en DevOps, un conjunto de prácticas y principios denominados *Rugged DevOps* . Ideas similares fueron creado por el Dr. Tapabrata Pal, Director y Miembro Técnico de Ingeniería de Plataformas en Capital One y el equipo de Capital One, que describen sus procesos como *DevOpsSec* , donde Infosec está integrado en todas las etapas del SDLC. Robusto DevOps rastrea parte de su historia a *Visible Ops Security* , escrito por Gene Kim, Paul Love y George Spafford.

A lo largo de *The DevOps Handbook* , hemos explorado cómo integrar completamente el control de calidad y Objetivos de operaciones en todo nuestro flujo de valor tecnológico. En este capítulo, nosotros Describir cómo integrar de manera similar los objetivos de Infosec en nuestro trabajo diario, donde podamos aumentar la productividad operativa y del desarrollador, aumentar la seguridad y aumentar nuestra seguridad.

INTEGRAR LA SEGURIDAD EN LA ITERACIÓN DEL DESARROLLO MANIFESTACIONES

Uno de nuestros objetivos es que los equipos principales participen con Infosec lo antes posible, ya que en oposición a participar principalmente al final del proyecto. Una forma de hacerlo es mediante invitando a Infosec a las demostraciones de productos al final de cada intervalo de desarrollo que puedan comprender mejor los objetivos del equipo en el contexto de los objetivos de la organización, observar sus implementaciones a medida que se construyen y proporcionar orientación y comentarios

en las primeras etapas del proyecto, cuando hay la mayor cantidad de tiempo y libertad para hacer correcciones.

Justin Arbuckle, ex arquitecto jefe de GE Capital, observa: “Cuando se trataba de seguridad de la información y cumplimiento, encontramos que los bloqueos al final del proyecto eran mucho más caros que al principio, y los bloqueos de Infosec estaban entre los peor. El 'cumplimiento por demostración' se convirtió en uno de los rituales que usamos para cambiar todo esto complejidad al principio del proceso ".

Él continúa: “Al involucrar a Infosec durante la creación de cualquier nueva capacidad, pudimos reducir drásticamente nuestro uso de listas de verificación estáticas y confiar más en el uso

su experiencia en todo el proceso de desarrollo de software ".

Esto ayudó a la organización a alcanzar sus objetivos. Snehal Antani, ex CIO de Enterprise Arquitectura en GE Capital Americas, describió sus tres principales medidas comerciales clave fueron "velocidad de desarrollo (es decir, velocidad de entrega de funciones al mercado), cliente fallido interacciones (es decir, interrupciones, errores) y tiempo de respuesta de cumplimiento (es decir, tiempo de espera de la auditoría Solicitar la entrega de toda la información cuantitativa y cualitativa requerida para cumplir con solicitud)."

Cuando Infosec es una parte asignada del equipo, incluso si solo se les mantiene informados y observando el proceso, obtienen el contexto comercial que necesitan para mejorar su gestión basada en el riesgo decisiones Además, Infosec puede ayudar a los equipos destacados a aprender lo que se necesita para cumplir objetivos de seguridad y cumplimiento.

INTEGRAR LA SEGURIDAD EN EL SEGUIMIENTO DE DEFECTOS Y DESPUÉS DE MUERTE

Cuando sea posible, queremos realizar un seguimiento de todos los problemas de seguridad abiertos en el mismo sistema de seguimiento del trabajo que Desarrollo y Operaciones están utilizando, asegurando que el trabajo sea visible y pueda priorizado contra todo otro trabajo. Esto es muy diferente de cómo Infosec ha tradicionalmente funcionó, donde todas las vulnerabilidades de seguridad se almacenan en un GRC (gobierno, riesgo y cumplimiento) herramienta a la que solo Infosec tiene acceso. En cambio, pondremos todo el trabajo necesario en los sistemas que usan Dev y Ops.

En una presentación en los Austin DevOpsDays de 2012, Nick Galbreath, quien encabezó La seguridad de la información en Etsy durante muchos años describe cómo trataron los problemas de seguridad, “Ponemos todos los problemas de seguridad en JIRA, que todos los ingenieros usan en su trabajo diario, y ellos eran 'P1' o 'P2', lo que significa que tenían que repararse inmediatamente o al final de la semana, incluso si el problema es solo una aplicación interna ".

Además, afirma: "Cada vez que tuvimos un problema de seguridad, llevaríamos a cabo una publicación mortem, porque resultaría en una mejor educación de nuestros ingenieros sobre cómo prevenirlo volver a suceder en el futuro, así como un mecanismo fantástico para transferir conocimiento de seguridad para nuestros equipos de ingeniería ".

INTEGRAR CONTROLES PREVENTIVOS DE SEGURIDAD EN CÓDIGO DE FUENTE COMPARTIDO REPOSITORIOS Y COMPARTIDO SERVICIOS

En el capítulo 20, creamos un repositorio de código fuente compartido que facilita que cualquiera pueda descubrir y reutilizar el conocimiento colectivo de nuestra organización, no solo para nuestro código, sino también para nuestras cadenas de herramientas, canalización de implementación, estándares, etc. Al hacer esto, cualquiera puede

beneficiarse de la experiencia acumulativa de todos en la organización.

Ahora agregaremos a nuestro repositorio de código fuente compartido cualquier mecanismo o herramienta que ayude permitirnos garantizar que nuestras aplicaciones y entornos sean seguros. Agregaremos bibliotecas que están pre-bendecidos por la seguridad para cumplir objetivos específicos de Infosec, como la autenticación y bibliotecas y servicios de encriptación. Porque todos en el flujo de valor de DevOps usan control de versiones para cualquier cosa que construyan o admitan, poniendo nuestra seguridad de la información los artefactos allí hacen que sea mucho más fácil influir en el trabajo diario de Dev y Ops, porque todo lo que creamos está disponible, se puede buscar y es reutilizable. El control de versiones también sirve como mecanismo de comunicación omnidireccional para mantener a todas las partes al tanto de los cambios hecho.

Si tenemos una organización centralizada de servicios compartidos, también podemos colaborar con ellos para crear y operar plataformas compartidas relevantes para la seguridad, como la autenticación, autorización, registro y otros servicios de seguridad y auditoría que requieren Dev y Ops. Cuando los ingenieros usan una de estas bibliotecas o servicios predefinidos, no necesitarán programar una revisión de diseño de seguridad por separado para ese módulo; usarán la guía hemos creado con respecto al fortalecimiento de la configuración, la configuración de seguridad de la base de datos, las longitudes de las claves, Etcétera.

Para aumentar aún más la probabilidad de que se utilicen los servicios y bibliotecas que proporcionamos correctamente, podemos proporcionar capacitación en seguridad a Dev y Ops, así como revisar lo que han creado para ayudar a garantizar que los objetivos de seguridad se implementen correctamente, especialmente para equipos que usan estas herramientas por primera vez.

En última instancia, nuestro objetivo es proporcionar las bibliotecas o servicios de seguridad que todo moderno la aplicación o el entorno requieren, como habilitar la autenticación del usuario, la autorización, gestión de contraseñas, cifrado de datos, etc. Además, podemos proporcionar Dev y operaciones con ajustes de configuración específicos de seguridad efectivos para los componentes que usan en sus pilas de aplicaciones, como el registro, la autenticación y el cifrado. Podemos incluir elementos como:

- Bibliotecas de códigos y sus configuraciones recomendadas (por ejemplo, 2FA [dos factores biblioteca de autenticación], hash de contraseña de bcrypt, registro)

- Gestión secreta (p. Ej., Configuración de conexión, claves de cifrado) utilizando herramientas como Bóveda, zapatilla de deporte, Keywhiz, credstash, Trousseau, Red October, etc.

- Paquetes y compilaciones del sistema operativo (por ejemplo, NTP para sincronización horaria, versiones seguras de OpenSSL con configuraciones correctas, OSSEC o Tripwire para monitoreo de integridad de archivos, syslog configuración para garantizar el registro de la seguridad crítica en nuestra pila ELK centralizada)

Al poner todo esto en nuestro repositorio de código fuente compartido, lo hacemos fácil para cualquier ingeniero para crear y usar correctamente los estándares de registro y cifrado en sus aplicaciones y entornos, sin más trabajo de nuestra parte.

También debemos colaborar con los equipos de Ops para crear un libro de cocina base o crear una imagen de nuestro SO, bases de datos y otra infraestructura (p. Ej., NGINX, Apache, Tomcat), lo que demuestra que son

en un estado conocido, seguro y de riesgo reducido. Nuestro repositorio compartido no solo se convierte en el lugar donde podemos obtener las últimas versiones, pero también se convierte en un lugar donde podemos colaborar con otros ingenieros y supervisar y alertar sobre los cambios realizados en la seguridad Módulos sensibles.

INTEGRAR LA SEGURIDAD EN NUESTRA TUBERÍA DE DESPLIEGUE

En épocas anteriores, para fortalecer y proteger nuestra aplicación, comenzaríamos nuestra seguridad revisión después de que se completó el desarrollo. A menudo, el resultado de esta revisión sería cientos de páginas de vulnerabilidades en un PDF, que daríamos a Desarrollo y Operaciones, que estarían completamente sin abordar debido a la presión de la fecha de vencimiento del proyecto o problemas que se encuentran demasiado tarde en el ciclo de vida del software para poder corregirlos fácilmente.

En este paso, automatizaremos tantas pruebas de seguridad de la información como sea posible, para que se ejecutan junto con todas nuestras otras pruebas automatizadas en nuestra tubería de implementación, siendo realizado (idealmente) en cada código confirmado por Dev u Ops, e incluso en las primeras etapas de un proyecto de software.

Nuestro objetivo es proporcionar a Dev y Ops comentarios rápidos sobre su trabajo para que puedan notificados cada vez que cometen cambios que son potencialmente inseguros. Al hacer esto, nosotros permitirles detectar y corregir rápidamente problemas de seguridad como parte de su trabajo diario, que permite el aprendizaje y evita futuros errores.

Idealmente, estas pruebas de seguridad automatizadas se ejecutarán en nuestro canal de implementación junto con el otras herramientas de análisis de código estático.

Herramientas como Gauntlt han sido diseñadas para integrarse en las tuberías de implementación, que ejecutar pruebas de seguridad automatizadas en nuestras aplicaciones, nuestras dependencias de aplicaciones, nuestras entorno, etc. Sorprendentemente, Gauntlt incluso pone todas sus pruebas de seguridad en la sintaxis de Gherkin scripts de prueba, que es ampliamente utilizado por los desarrolladores para pruebas unitarias y funcionales. Haciendo esto coloca las pruebas de seguridad en un marco con el que probablemente ya estén familiarizados. Esto también permite pruebas de seguridad para ejecutar fácilmente en una tubería de implementación en cada cambio comprometido, como análisis de código estático, comprobación de dependencias vulnerables o pruebas dinámicas.

Figura 43: *Jenkins ejecuta pruebas de seguridad automatizadas (Fuente: James Wicket y Gareth Rushgrove, "Código probado en batalla sin la batalla", presentación de la conferencia Velocity 2014, publicado en Speakerdeck.com, 24 de junio de 2014, https://speakerdeck.com/garethr/battle-tested-código sin batalla)*

Al hacer esto, brindamos a todos los que están en la cadena de valor la retroalimentación más rápida posible sobre la seguridad de lo que están creando, permitiendo a los ingenieros de Dev y Ops encontrar y solucionar Problemas rápidamente.

ASEGURAR LA SEGURIDAD DE LA APLICACIÓN

A menudo, las pruebas de desarrollo se centran en la corrección de la funcionalidad, mirando a lo positivo
La lógica fluye. Este tipo de prueba a menudo se conoce como la *ruta feliz* , que valida al usuario
viajes (y a veces rutas alternativas) donde todo sale como se esperaba, sin
excepciones o condiciones de error.

Por otro lado, los profesionales eficaces de control de calidad, Infosec y Fraude a menudo se centrarán en
caminos tristes , que suceden cuando las cosas van mal, especialmente en relación con la seguridad
condiciones de error (Estos tipos de condiciones específicas de seguridad a menudo se mencionan en broma
como los *malos caminos*)

Por ejemplo, supongamos que tenemos un sitio de comercio electrónico con un formulario de entrada del cliente que acepta
números de tarjeta de crédito como parte de la generación de un pedido del cliente. Queremos definir a todos los tristes
y rutas de baño necesarias para garantizar que las tarjetas de crédito no válidas se rechacen correctamente para evitar
fraudes y vulnerabilidades de seguridad, como inyecciones de SQL, desbordamientos de búfer y otros elementos no deseados
resultados.

En lugar de realizar estas pruebas manualmente, idealmente las generaríamos como parte de nuestro
pruebas unitarias o funcionales automatizadas para que puedan ejecutarse continuamente en nuestra implementación
tubería. Como parte de nuestras pruebas, querremos incluir lo siguiente:

Análisis estático: esta es una prueba que realizamos en un entorno sin tiempo de ejecución, idealmente
en la tubería de despliegue. Por lo general, una herramienta de análisis estático inspeccionará el código del programa para
todos los posibles comportamientos de tiempo de ejecución y buscar fallas de codificación, puertas traseras y potencialmente
código malicioso (esto a veces se conoce como "prueba de adentro hacia afuera"). Ejemplos de
Las herramientas incluyen Brakeman, Code Climate y la búsqueda de funciones de código prohibido (por ejemplo,
"Exec ()").

Análisis dinámico: a diferencia de las pruebas estáticas, el análisis dinámico consiste en pruebas
ejecutado mientras un programa está en funcionamiento. Las pruebas dinámicas monitorean elementos como el sistema
memoria, comportamiento funcional, tiempo de respuesta y rendimiento general del sistema.
Este método (a veces conocido como "prueba desde afuera hacia adentro") es similar al
manera en que un tercero malintencionado podría interactuar con una aplicación. Ejemplos
incluyen Arachni y OWASP ZAP (Zed Attack Proxy).† Algunos tipos de penetración
las pruebas también se pueden realizar de manera automatizada y se deben incluir como parte de
Análisis dinámico utilizando herramientas como Nmap y Metasploit. Idealmente, debemos realizar
prueba dinámica automatizada durante la fase de prueba funcional automatizada de nuestro
canalización de implementación, o incluso en contra de nuestros servicios mientras están en producción. A
garantizar un manejo correcto de la seguridad, herramientas como OWASP ZAP se pueden configurar para atacar a nuestros
servicios a través de un proxy de navegador web e inspeccionar el tráfico de red dentro de nuestra prueba
aprovechar.

Escaneo de dependencia: otro tipo de prueba estática que normalmente realizaríamos en el tiempo de construcción dentro de nuestra tubería de implementación implica inventariar todas nuestras dependencias para binarios y ejecutables, y asegurando que estas dependencias, que a menudo no tienen control, están libres de vulnerabilidades o binarios maliciosos. Ejemplos incluyen Gemnasium y auditorías de paquetes para Ruby, Maven para Java y OWASP Verificación de dependencia.

Página 332

Integridad del código fuente y firma del código: todos los desarrolladores deben tener su propio PGP clave, tal vez creada y administrada en un sistema como keybase.io. Todos se comprometen a el control de versión debe estar firmado, eso es sencillo de configurar con el comando abierto herramientas de origen gpg y git. Además, todos los paquetes creados por el proceso de CI deben ser firmado, y su hash registrado en el servicio de registro centralizado para fines de auditoría.

Además, debemos definir patrones de diseño para ayudar a los desarrolladores a escribir código para evitar abuso, como establecer límites de tarifas para nuestros servicios y atenuar los botones de envío después de han sido presionados. OWASP publica una gran cantidad de orientación útil, como el Serie de hojas de trucos, que incluye:

Cómo almacenar contraseñas

Cómo manejar contraseñas olvidadas

Cómo manejar el registro

Cómo evitar vulnerabilidades de scripting entre sitios (XSS)

La presentación de "10 implementaciones por día: cooperación entre desarrolladores y operaciones en Flickr" de John Allspaw y Paul Hammond son famosos por catalizar la comunidad de Dev y Ops en 2009. El equivalente para la comunidad de seguridad de la información es probablemente la presentación que Justin Collins, Alex Smolen y Neil Matatall dieron sobre su seguridad de la información trabajo de transformación en Twitter en la conferencia AppSecUSA en 2012.

Twitter tuvo muchos desafíos debido al hipercrecimiento. Durante años, la famosa Fail Whale la página de error se mostraría cuando Twitter no tuviera la capacidad suficiente para mantener con la demanda del usuario, mostrando un gráfico de una ballena levantada por ocho pájaros. los la escala de crecimiento de usuarios fue impresionante: entre enero y marzo de 2009, el El número de usuarios activos de Twitter pasó de 2.5 millones a 10 millones.

Twitter también tuvo problemas de seguridad durante este período. A principios de 2009, dos graves Se produjeron violaciones de seguridad. Primero, en enero, la cuenta de Twitter @BarackObama fue hackeado. Luego, en abril, las cuentas administrativas de Twitter se vieron comprometidas

a través de un ataque de diccionario de fuerza bruta. Estos eventos llevaron al Comercio Federal Comisión para juzgar que Twitter estaba engañando a sus usuarios para que creyeran que sus las cuentas estaban seguras y emitieron una orden de consentimiento de la FTC.

La orden de consentimiento requiere que Twitter cumpla dentro de los sesenta días instituyendo un conjunto de procesos que debían hacerse cumplir durante los siguientes veinte años y harían lo siguiendo:

- Designar a un empleado o empleados para que sean responsables de la información de Twitter plan de seguridad
- Identifique riesgos razonablemente previsibles, tanto internos como externos, que podrían conducir a un incidente de intrusión y crear e implementar un plan para abordar estos riesgos ↑

Página 333

- Mantener la privacidad de la información del usuario, no solo de fuentes externas, sino también internamente, con un resumen de las posibles fuentes de verificación y prueba de seguridad y corrección de estas implementaciones

El grupo de ingenieros asignado para resolver este problema tuvo que integrar la seguridad en el trabajo diario de Dev y Ops y cerrar los agujeros de seguridad que permitieron las violaciones pasar en primer lugar.

En su presentación mencionada anteriormente, Collins, Smolen y Matatall identificaron varios problemas que necesitaban abordar:

- **Evite que se repitan los errores de seguridad:** descubrieron que estaban reparando los mismos defectos y vulnerabilidades una y otra vez. Necesitaban modificar el sistema de trabajo y herramientas de automatización para evitar que los problemas vuelvan a ocurrir.
- **Integre los objetivos de seguridad en las herramientas de desarrollador existentes:** identificaron temprano en eso la principal fuente de vulnerabilidades fueron los problemas de código. No pudieron ejecutar una herramienta que generó un gran informe PDF y luego lo envió por correo electrónico a alguien en Desarrollo o Operaciones En cambio, tenían que proporcionar al desarrollador que había creado el vulnerabilidad con la información exacta necesaria para solucionarlo.
- **Preservar la confianza del desarrollo:** necesitaban ganarse y mantener la confianza de Desarrollo. Eso significaba que necesitaban saber cuándo enviaron Desarrollo falso positivos, para que puedan corregir el error que provocó el falso positivo y evitar Perder el tiempo del desarrollo.
- **Mantenga un flujo rápido a través de Infosec a través de la automatización:** incluso cuando el código el escaneo de vulnerabilidades fue automatizado, Infosec aún tenía que hacer mucho trabajo manual y esperando. Tuvieron que esperar a que se completara la exploración, recuperar la gran pila de informes,

interprete los informes y luego encuentre a la persona responsable de arreglarlo. Y cuando el código cambiado, tuvo que hacerse de nuevo. Al automatizar el trabajo manual, hicieron menos tareas tontas de "presionar botones", lo que les permitió usar más creatividad y juicio.

Haga que todo sea autoservicio relacionado con la seguridad, si es posible: confiaron en que la mayoría la gente quería hacer lo correcto, por lo que era necesario proporcionarles todos los contexto e información que necesitaban para solucionar cualquier problema.

Adopte un enfoque holístico para lograr los objetivos de Infosec: su objetivo era hacer análisis desde todos los ángulos: código fuente, entorno de producción e incluso Lo que sus clientes estaban viendo.

El primer gran avance para el equipo de Infosec ocurrió durante un hack en toda la compañía semana cuando integraron el análisis de código estático en el proceso de construcción de Twitter. los El equipo usó Brakeman, que escanea las aplicaciones de Ruby on Rails en busca de vulnerabilidades. los El objetivo era integrar el escaneo de seguridad en las primeras etapas del desarrollo proceso, no solo cuando el código se confirmó en el repositorio de código fuente.

Página 334

Figura 44: *Número de vulnerabilidades de seguridad de Brakeman detectadas*

Los resultados de integrar las pruebas de seguridad en el proceso de desarrollo fueron asombroso. A lo largo de los años, creando comentarios rápidos para los desarrolladores cuando escribir código inseguro y mostrarles cómo solucionar las vulnerabilidades, Brakeman ha redujo la tasa de vulnerabilidades encontradas en un 60%, como se muestra en la figura 44. (Los picos

generalmente están asociados con nuevos lanzamientos de Brakeman).

Este estudio de casos ilustra cuán necesario es integrar la seguridad en el día a día.
trabajo y herramientas de DevOps y qué tan efectivamente puede funcionar. Hacerlo mitiga
riesgo de seguridad, reduce la probabilidad de vulnerabilidades en el sistema y ayuda a enseñar
desarrolladores para escribir código más seguro.

ASEGURAR LA SEGURIDAD DE NUESTRA CADENA DE SUMINISTRO DE SOFTWARE

Josh Corman observó que como desarrolladores "ya no estamos escribiendo software personalizado:
en su lugar, reunimos lo que necesitamos a partir de piezas de código abierto, que se ha convertido en el software
cadena de suministro de la que dependemos mucho ". En otras palabras, cuando usamos
componentes o bibliotecas, ya sean comerciales o de código abierto, en nuestro software, no solo
heredan su funcionalidad, pero también cualquier vulnerabilidad de seguridad que contengan.

Al seleccionar el software, detectamos cuándo nuestros proyectos de software dependen de componentes
o bibliotecas que tienen vulnerabilidades conocidas y ayudan a los desarrolladores a elegir los componentes
usan deliberadamente y con el debido cuidado, seleccionando esos componentes (por ejemplo, código abierto
proyectos) que tienen un historial demostrado de reparación rápida de vulnerabilidades de software. Nosotros
También busque múltiples versiones de la misma biblioteca que se utilizan en nuestra producción
paisaje, particularmente la presencia de versiones anteriores de bibliotecas que contienen conocidos
vulnerabilidades

El examen de las infracciones de datos del titular de la tarjeta muestra la importancia de la seguridad del código abierto
Los componentes que elegimos pueden ser. Desde 2008, el incumplimiento anual de datos de Verizon PCI
El Informe de Investigación (DBIR) ha sido la voz más autorizada en violaciones de datos donde
los datos del titular de la tarjeta se perdieron o fueron robados. En el informe de 2014, estudiaron más de ochenta y cinco

mil infracciones para comprender mejor de dónde venían los ataques, cómo el titular de la tarjeta
los datos fueron robados y los factores que llevaron a la violación.

El DBIR descubrió que diez vulnerabilidades (es decir, CVE) representaban casi el 97% de la
exploits utilizados en violaciones de datos de titulares de tarjetas estudiadas en 2014. De estas diez vulnerabilidades,
ocho de ellos tenían más de diez años.

El *Informe de estado de la cadena de suministro de software de Sonatype de 2015* analizó aún más el
datos de vulnerabilidad del repositorio central de Nexus. En 2015, este repositorio proporcionó el
construir artefactos para más de 605,000 proyectos de código abierto, atendiendo a más de diecisiete mil millones
descargar solicitudes de artefactos y dependencias principalmente para la plataforma Java,
originario de 106,000 organizaciones.

El informe incluía estos hallazgos sorprendentes:

La organización típica dependía de 7,601 artefactos de construcción (es decir, proveedores de software o componentes) y se utilizaron 18,614 versiones diferentes (es decir, partes de software).

De esos componentes utilizados, el 7.5% tenía vulnerabilidades conocidas, con más del 66% de esas vulnerabilidades tienen más de dos años sin haberse resuelto.

La última estadística confirma otro estudio de seguridad de la información realizado por el Dr. Dan Geer y Josh. Corman, que mostró el de los proyectos de código abierto con vulnerabilidades conocidas registrada en la Base de datos de vulnerabilidad nacional, solo el 41% alguna vez fueron reparados y requeridos, en promedio, 390 días para publicar una solución. Para aquellas vulnerabilidades que fueron etiquetadas en severidad más alta (es decir, aquellos calificados como CVSS nivel 10), las correcciones requieren 224 días.§

ASEGURAR LA SEGURIDAD DEL MEDIO AMBIENTE

En este paso, debemos hacer lo que sea necesario para ayudar a garantizar que los entornos estén en un estado endurecido, de riesgo reducido. Aunque es posible que hayamos creado configuraciones buenas y conocidas ya, debemos poner controles de monitoreo para asegurar que todas las instancias de producción coincidan estos buenos estados conocidos.

Hacemos esto mediante la generación de pruebas automatizadas para garantizar que se hayan realizado todas las configuraciones apropiadas aplicado correctamente para el endurecimiento de la configuración, la configuración de seguridad de la base de datos, las longitudes de clave y etc. Además, utilizaremos pruebas para escanear nuestros entornos en busca de vulnerabilidades¶

Otra categoría de verificación de seguridad es comprender los entornos reales (es decir, "como en realidad lo son "). Ejemplos de herramientas para esto incluyen Nmap para asegurar que solo se espera los puertos están abiertos y Metasploit para garantizar que hemos reforzado adecuadamente nuestros entornos contra vulnerabilidades conocidas, como el escaneo con ataques de inyección SQL. La salida de Estas herramientas deben colocarse en nuestro repositorio de artefactos y compararse con las anteriores versión como parte de nuestro proceso de prueba funcional. Hacer esto nos ayudará a detectar cualquier cambios indeseables tan pronto como ocurran.

Se proyectaba que las agencias del gobierno federal de los Estados Unidos gastarían casi $ 80 mil millones en TI en 2016, apoyando la misión de todas las agencias del poder ejecutivo. A pesar de agencia, para llevar cualquier sistema de "desarrollo completo" a "vivir en producción" requiere obtener una Autoridad para Operar (ATO) de una Autoridad de Aprobación Designada (DAA) Las leyes y políticas que rigen el cumplimiento en el gobierno están comprendidas de decenas de documentos que suman más de cuatro mil páginas, llenas de siglas como FISMA, FedRAMP y FITARA. Incluso para sistemas que solo

requieren bajos niveles de confidencialidad, integridad y disponibilidad, más de cien los controles deben implementarse, documentarse y probarse. Normalmente toma entre ocho y catorce meses para que se otorgue una ATO después de "dev complete".

El equipo 18F en la Administración de Servicios Generales del gobierno federal ha tomado Un enfoque múltiple para resolver este problema. Mike Bland explica: "18F fue creado dentro de la Administración de Servicios Generales para capitalizar el impulso generado por la recuperación de Healthcare.gov para reformar cómo el gobierno construye y compra software ".

Un esfuerzo de 18F es una plataforma como un servicio llamado Cloud.gov, creado a partir de código abierto componentes. Cloud.gov se ejecuta actualmente en AWS GovCloud. No solo el la plataforma maneja muchas de las preocupaciones operativas que los equipos de entrega podrían de otra manera tener que cuidar, como el registro, la supervisión, las alertas y el ciclo de vida del servicio gestión, también maneja la mayor parte de las preocupaciones de cumplimiento. Al correr en esto plataforma, una gran mayoría de los controles que los sistemas gubernamentales deben implementar puede ser atendido a nivel de infraestructura y plataforma. Entonces, solo el resto los controles que están dentro del alcance en la capa de aplicación deben documentarse y probarse, reduciendo significativamente la carga de cumplimiento y el tiempo que lleva recibir una ATO.

AWS GovCloud ya ha sido aprobado para su uso en sistemas del gobierno federal de todos los tipos, incluidos aquellos que requieren altos niveles de confidencialidad, integridad y disponibilidad. Para cuando lea este libro, se espera que Cloud.gov sea aprobado para todos los sistemas que requieren niveles moderados de confidencialidad, integridad y disponibilidad.** **

Además, el equipo de Cloud.gov está creando un marco para automatizar la creación de planes de seguridad del sistema (SSP), que son "descripciones completas de la arquitectura del sistema, controles implementados y postura de seguridad general ... [que son] a menudo increíblemente complejas, con varios cientos de páginas de longitud ". Ellos desarrolló una herramienta prototipo llamada mampostería de cumplimiento para que los datos del SSP se almacenen en YAML legible por máquina y luego se convirtió en GitBooks y PDF automáticamente.

18F se dedica a trabajar al aire libre y publica su trabajo de código abierto en el dominio publico. Puede encontrar la mampostería de cumplimiento y los componentes que conforman Cloud.gov en los repositorios de GitHub de 18F, incluso puede defender su propia instancia de Cloud.gov. El trabajo sobre la documentación abierta para los SSP se está realizando de cerca. asociación con la comunidad OpenControl.

INTEGRAR LA SEGURIDAD DE LA INFORMACIÓN EN LA PRODUCCIÓN
Telemetría

Marcus Sachs, uno de los investigadores de Verizon Data Breach, observó en 2010: "Año después
año, en la gran mayoría de las infracciones de datos del titular de la tarjeta, la organización detectó el
violación de seguridad meses o trimestres después de la violación. Peor aún, la forma en la brecha
se detectó que no era un control de monitoreo interno, pero era mucho más probable que alguien
fuera de la organización, generalmente un socio comercial o el cliente que nota
transacciones fraudulentas Una de las razones principales para esto es que nadie en el
la organización revisaba regularmente los archivos de registro ".

En otras palabras, los controles de seguridad internos a menudo son ineficaces para detectar con éxito
infracciones de manera oportuna, ya sea por puntos ciegos en nuestro monitoreo o porque no
uno en nuestra organización está examinando la telemetría relevante en su trabajo diario.

En el capítulo 14, discutimos la creación de una cultura en Dev y Ops donde todos en el valor
Stream está creando telemetría y métricas de producción, haciéndolas visibles en prominentes
lugares públicos para que todos puedan ver cómo están funcionando nuestros servicios en la producción.
Además, exploramos la necesidad de buscar implacablemente señales de falla cada vez más débiles
para que podamos encontrar y solucionar problemas antes de que resulten en una falla catastrófica.

Aquí, implementamos el monitoreo, el registro y las alertas necesarias para cumplir con nuestra información
objetivos de seguridad en todas nuestras aplicaciones y entornos, así como garantizar que
está adecuadamente centralizado para facilitar un análisis y una respuesta fáciles y significativos.

Hacemos esto integrando nuestra telemetría de seguridad en las mismas herramientas que Desarrollo, QA,
y Operaciones están utilizando, dando a todos en el flujo de valor visibilidad de cómo sus
las aplicaciones y los entornos funcionan en un entorno de amenazas hostiles donde
los atacantes constantemente intentan explotar vulnerabilidades, obtener acceso no autorizado,
plantar puertas traseras, cometer fraude, realizar denegaciones de servicio, etc.

Al irradiar cómo se atacan nuestros servicios en el entorno de producción, nosotros
Reforzar que todos deben pensar en los riesgos de seguridad y diseñar
contramedidas en su trabajo diario.

CREANDO TELEMETRÍA DE SEGURIDAD EN NUESTRAS APLICACIONES

Para detectar comportamientos problemáticos del usuario que podrían ser un indicador o un facilitador de fraude
y acceso no autorizado, debemos crear la telemetría relevante en nuestras aplicaciones.

Los ejemplos pueden incluir:

- Inicios de sesión de usuario exitosos y fallidos
- La contraseña del usuario se restablece
- La dirección de correo electrónico del usuario se restablece
- Cambios de tarjeta de crédito del usuario

Por ejemplo, como un indicador temprano de intentos de inicio de sesión de fuerza bruta para obtener ganancias no autorizadas
acceso, podríamos mostrar la proporción de intentos de inicio de sesión fallidos a inicios de sesión exitosos. Y,
por supuesto, debemos crear alertas sobre eventos importantes para asegurarnos de que podamos detectar y
Corrija los problemas rápidamente.

CREANDO TELEMETRÍA DE SEGURIDAD EN NUESTRO ENTORNO

Además de instrumentar nuestra aplicación, también necesitamos crear suficiente telemetría en nuestros entornos para que podamos detectar los primeros indicadores de acceso no autorizado, especialmente en los componentes que se ejecutan en infraestructura que no controlamos (por ejemplo, hosting ambientes, en la nube).

Necesitamos monitorear y potencialmente alertar sobre artículos, incluidos los siguientes:

Cambios en el sistema operativo (por ejemplo, en producción, en nuestra infraestructura de compilación)

Cambios de grupo de seguridad

Cambios en las configuraciones (p. Ej., OSSEC, Puppet, Chef, Tripwire)

Cambios en la infraestructura de la nube (por ejemplo, VPC, grupos de seguridad, usuarios y privilegios)

Intentos de XSS (es decir, "ataques de secuencias de comandos entre sitios")

Intentos de SQLi (es decir, "ataques de inyección SQL")

Errores del servidor web (p. Ej., Errores 4XX y 5XX)

También queremos confirmar que hemos configurado correctamente nuestro registro para que toda la telemetría sea ser enviado al lugar correcto. Cuando detectamos ataques, además de registrar que sucedió, también podemos optar por bloquear el acceso y almacenar información sobre la fuente para ayúdenos a elegir las mejores acciones de mitigación.

En 2010, Nick Galbreath fue director de ingeniería en Etsy y responsable de seguridad de la información, control de fraudes y privacidad. Galbreath definió el *fraude* como cuando "el el sistema funciona incorrectamente, permitiendo entradas no válidas o no inspeccionadas en el sistema, causando pérdida financiera, pérdida / robo de datos, tiempo de inactividad del sistema, vandalismo o un ataque a otro sistema ".

Para lograr estos objetivos, Galbreath no creó un control de fraude o departamento de seguridad de la información; en cambio, incrustó esas responsabilidades en todo el flujo de valor de DevOps.

Galbreath creó la telemetría relacionada con la seguridad que se mostró junto con todos los otras métricas más orientadas a Dev y Ops, que cada ingeniero de Etsy veía habitualmente:

> **Terminaciones anormales del programa de producción (p. Ej., Fallas de segmentación, núcleo vertederos, etc.):** "De particular preocupación era por qué ciertos procesos seguían volcando el núcleo en todo nuestro entorno de producción, provocado por el tráfico proveniente del Dirección IP, una y otra vez. De igual preocupación fueron los HTTP '500 Internal Errores del servidor. Estos son indicadores de que una vulnerabilidad estaba siendo explotada para ganar acceso no autorizado a nuestros sistemas, y que un parche debe aplicarse con urgencia ".

Error de sintaxis de la base de datos: "Siempre buscamos errores de sintaxis de la base de datos en el interior nuestro código: estos ataques de inyección SQL habilitados o eran ataques reales en

Progreso. Por esta razón, teníamos tolerancia cero para los errores de sintaxis de la base de datos en nuestro código, porque sigue siendo uno de los principales vectores de ataque utilizados para comprometer sistemas ".

Indicaciones de ataques de inyección SQL: "Esta fue una prueba ridículamente simple: simplemente alerta cuando aparezca 'UNION ALL' en los campos de entrada del usuario, ya que casi siempre indica un ataque de inyección SQL. También agregamos pruebas unitarias para asegurarnos de que este tipo de entrada de usuario no controlada nunca podría permitirse en nuestra base de datos consultas ".

Figura 45: Los *desarrolladores verían los intentos de inyección SQL en Graphite en Etsy (Fuente: "DevOpsSec: Aplicación de los principios de DevOps a la seguridad, DevOpsDays Austin 2012", SlideShare.net, publicado por Nick Galbreath, 12 de abril de 2012, http://www.slideshare.net/nickgsuperstar/devopssec-apply-devops-principles-to-security.)*

La Figura 45 es un ejemplo de un gráfico que vería cada desarrollador, que muestra el Número de posibles ataques de inyección SQL que se intentaron en la producción ambiente. Como observó Galbreath, "Nada ayuda a los desarrolladores a comprender cómo hostil el entorno operativo es que ver su código siendo atacado en tiempo real hora."

Galbreath observó: “Uno de los resultados de mostrar este gráfico fue que los desarrolladores ¡Me di cuenta de que estaban siendo atacados todo el tiempo! Y eso fue increíble, porque cambió la forma en que los desarrolladores pensaban sobre la seguridad de su código tal como estaban escribiendo el código ".

PROTEJA NUESTRA TUBERÍA DE DESPLIEGUE

La infraestructura que soporta nuestra integración continua y despliegue continuo Los procesos también presentan una nueva superficie vulnerable al ataque. Por ejemplo, si alguien

compromete los servidores que ejecutan la canalización de implementación que tiene las credenciales para nuestro sistema de control de versiones, podría permitir que alguien robe el código fuente. Peor si el la canalización de implementación tiene acceso de escritura, un atacante también podría inyectar cambios maliciosos en nuestro repositorio de control de versiones y, por lo tanto, inyectar cambios maliciosos en nuestra aplicación y servicios.

Como Jonathan Claudius, ex Tester de Seguridad Senior en TrustWave SpiderLabs, observó:
“Los servidores de prueba y compilación continuos son increíbles, y los uso yo mismo. Pero empecé pensando en formas de usar CI / CD como una forma de inyectar código malicioso. Lo que condujo a la pregunta de dónde sería un buen lugar para ocultar código malicioso? La respuesta fue obvia:

en las pruebas unitarias. En realidad, nadie mira las pruebas unitarias, y se ejecutan cada vez que alguien confirma el código al repositorio ".

Esto demuestra que para proteger adecuadamente la integridad de nuestras aplicaciones y entornos, también debemos mitigar los vectores de ataque en nuestra tubería de implementación. Riesgos incluir desarrolladores que introducen código que permite el acceso no autorizado (que hemos mitigado a través de controles como pruebas de código, revisiones de código y pruebas de penetración) y usuarios no autorizados que obtienen acceso a nuestro código o entorno (que hemos mitigado a través de controles como garantizar que las configuraciones coincidan con estados conocidos, buenos y efectivos parches).

Sin embargo, para proteger nuestro desarrollo continuo, integración o canalización de implementación, nuestro Las estrategias de mitigación pueden incluir:

- Fortalecer los servidores de integración y compilación continuos y garantizar que podamos reproducir ellos de manera automatizada, tal como lo haríamos para la infraestructura que soporta Servicios de producción orientados al cliente, para evitar nuestra construcción e integración continua servidores comprometidos

- Revisión de todos los cambios introducidos en el control de versiones, ya sea a través del par programación en tiempo de confirmación o mediante un proceso de revisión de código entre confirmación y fusión en el tronco, para evitar que los servidores de integración continua ejecuten código no controlado (por ejemplo, las pruebas unitarias pueden contener código malicioso que permite o habilita el acceso no autorizado)

- Instrumentar nuestro repositorio para detectar cuándo el código de prueba contiene llamadas API sospechosas (por ejemplo, las pruebas unitarias que acceden al sistema de archivos o la red) se registran en el repositorio, quizás ponerlo en cuarentena y desencadenar una revisión inmediata del código

- Asegurar que cada proceso de CI se ejecute en su propio contenedor aislado o VM

- Garantizar que las credenciales de control de versiones utilizadas por el sistema CI sean de solo lectura

CONCLUSIÓN

A lo largo de este capítulo hemos descrito formas de integrar la seguridad de la información.
objetivos en todas las etapas de nuestro trabajo diario. Hacemos esto integrando controles de seguridad en
los mecanismos que ya hemos creado, asegurando que todos los entornos bajo demanda también
en un estado endurecido y de riesgo reducido, integrando pruebas de seguridad en la implementación
canalización y garantizar la creación de telemetría de seguridad en preproducción y producción
ambientes. Al hacerlo, permitimos que la productividad operativa y de los desarrolladores aumente
mientras que simultáneamente aumenta nuestra seguridad general. Nuestro siguiente paso es proteger el
tubería de despliegue.

† El Open Web Application Security Project (OWASP) es una organización sin fines de lucro enfocada en mejorar la seguridad del software.

‡ Las estrategias para gestionar estos riesgos incluyen proporcionar capacitación y gestión a los empleados; repensar el diseño de sistemas de información, incluyendo red y software; e instituir procesos diseñados para prevenir, detectar y responder a los ataques.

§ Las herramientas que pueden ayudar a garantizar la integridad de nuestras dependencias de software incluyen OWASP Dependency Check y Sonatype Nexus Lifecycle.

¶ Ejemplos de herramientas que pueden ayudar con las pruebas de corrección de seguridad (es decir, "como debería ser") incluyen sistemas de gestión de configuración automatizados (por ejemplo, Puppet, Chef, Ansible, Salt), así como herramientas como ServerSpec y Netflix Simian Army (por ejemplo, Conformity Monkey, Security Monkey, etc.).

** Estas aprobaciones se conocen como FedRAMP JAB P-ATO.

23 Protegiendo el Tubería de implementación

A lo largo de este capítulo también veremos cómo proteger nuestra tubería de implementación
como lograr objetivos de seguridad y cumplimiento en nuestro entorno de control,
incluyendo la gestión del cambio y la separación del deber.

INTEGRAR SEGURIDAD Y CUMPLIMIENTO EN CAMBIO PROCESOS DE APROBACIÓN

Casi cualquier organización de TI de cualquier tamaño significativo tendrá una gestión de cambios existente
procesos, que son los controles principales para reducir las operaciones y los riesgos de seguridad.
El gerente de cumplimiento y los gerentes de seguridad confían en la gestión del cambio
procesos para los requisitos de cumplimiento, y generalmente requieren evidencia de que todos
los cambios han sido debidamente autorizados.

Si hemos construido nuestra tubería de implementación correctamente para que las implementaciones sean bajas,
riesgo, la mayoría de nuestros cambios no necesitarán pasar por una aprobación de cambio manual

proceso, porque habremos depositado nuestra confianza en controles como las pruebas automatizadas y monitoreo proactivo de la producción.

En este paso, haremos lo que sea necesario para garantizar que podamos integrar con éxito seguridad y cumplimiento en cualquier proceso de gestión de cambios existente. Cambio efectivo Las políticas de gestión reconocerán que existen diferentes riesgos asociados con diferentes tipos de cambios y que esos cambios se manejan de manera diferente. Estas Los procesos se definen en ITIL, que divide los cambios en tres categorías:

Cambios estándar: estos son **cambios de** bajo riesgo que siguen a un sistema establecido y proceso aprobado, pero también puede ser preaprobado. Incluyen actualizaciones mensuales de tablas de impuestos de aplicación o códigos de país, contenido del sitio web y cambios de estilo, y ciertos tipos de aplicaciones o parches del sistema operativo que tienen un buen conocimiento impacto. El proponente de cambio no requiere aprobación antes de implementar el cambio, y las implementaciones de cambios pueden ser completamente automatizadas y deben registrarse de modo que es la trazabilidad

Cambios normales: estos son **cambios de** mayor riesgo que requieren revisión o aprobación de la autoridad de cambio acordada. En muchas organizaciones, esta responsabilidad es colocado de manera inapropiada en la junta asesora de cambio (CAB) o cambio de emergencia consejo asesor (ECAB), que puede carecer de la experiencia requerida para comprender el pleno impacto del cambio, que a menudo conduce a tiempos de entrega inaceptablemente largos. Este problema es especialmente relevante para grandes implementaciones de código, que pueden contener cientos de

miles (o incluso millones) de líneas de código nuevo, presentadas por cientos de desarrolladores en el transcurso de varios meses. Para que los cambios normales sean autorizado, el CAB seguramente tendrá una solicitud de cambio bien definida (RFC) formulario que define qué información se requiere para la decisión de ir / no ir. los El formulario RFC generalmente incluye los resultados comerciales deseados, la utilidad planificada y garantía, † un caso de negocios con riesgos y alternativas, y un cronograma propuesto.‡

Cambios urgentes: son emergencias y, en consecuencia, un riesgo potencialmente alto, cambios que deben ponerse en producción de inmediato (por ejemplo, parche de seguridad urgente, restablecer el servicio). Estos cambios a menudo requieren la aprobación de la alta gerencia, pero permiten documentación a realizar después del hecho. Un objetivo clave de las prácticas de DevOps es racionalice nuestro proceso de cambio normal de modo que también sea adecuado para emergencias cambios

RE-CATEGORIZAR LA MAYORÍA DE NUESTRO MENOR RIESGO CAMBIOS COMO CAMBIOS ESTÁNDAR

Idealmente, al contar con una tubería de implementación confiable, ya habremos ganado un reputación por implementaciones rápidas, confiables y no dramáticas. En este punto, deberíamos tratar de obtener un acuerdo de las Operaciones y las autoridades de cambio relevantes que nuestro Se ha demostrado que los cambios son de bajo riesgo suficiente para ser definidos como estándar cambios, preaprobados por el CAB. Esto nos permite desplegarnos en producción sin necesita una mayor aprobación, aunque los cambios aún deben registrarse correctamente.

Una forma de respaldar una afirmación de que nuestros cambios son de bajo riesgo es mostrar un historial de cambios durante un período de tiempo significativo (por ejemplo, meses o trimestres) y proporcionar un completo lista de problemas de producción durante ese mismo período. Si podemos mostrar un gran éxito de cambio tasas y MTTR bajo, podemos afirmar que tenemos un entorno de control que es efectivamente previniendo errores de implementación, así como demostrando que podemos detectar de manera efectiva y rápida y corrija cualquier problema resultante.

Incluso cuando nuestros cambios se clasifican como cambios estándar, aún deben ser visuales y registrado en nuestros sistemas de gestión de cambios (por ejemplo, Remedy o ServiceNow). Idealmente, los despliegues serán realizados automáticamente por nuestra gestión de configuración y herramientas de canalización de implementación (por ejemplo, Puppet, Chef, Jenkins) y los resultados serán Grabado automáticamente. Al hacer esto, todos en nuestra organización (DevOps o no) lo harán tener visibilidad de nuestros cambios además de todos los otros cambios que ocurren en el organización.

Podemos vincular automáticamente estos registros de solicitud de cambio a elementos específicos en nuestro trabajo herramientas de planificación (por ejemplo, JIRA, Rally, LeanKit, ThoughtWorks Mingle), lo que nos permite crear más contexto para nuestros cambios, como la vinculación a defectos de características, incidentes de producción o historias de usuarios. Esto se puede lograr de una manera liviana al incluir números de boletos desde herramientas de planificación en los comentarios asociados con el control de versiones.§ Por Al hacer esto, podemos rastrear una implementación de producción a los cambios en el control de versiones y, a partir de ahí, rastrearlos hasta los tickets de herramientas de planificación.

Crear esta trazabilidad y contexto debería ser fácil y no crear demasiado carga onerosa o que consume mucho tiempo para los ingenieros. Vinculación a historias de usuarios, requisitos, o defectos es casi seguro suficiente; cualquier otro detalle, como abrir un boleto para cada compromiso con el control de versiones, probablemente no sea útil y, por lo tanto, innecesario y no deseado, ya que impondrá un nivel significativo de fricción en su trabajo diario.

QUÉ HACER CUANDO LOS CAMBIOS SE CATEGORIZAN COMO CAMBIOS NORMALES

Para aquellos cambios que no podemos clasificar como cambios estándar, serán

consideró *cambios normales* y requerirá la aprobación de al menos un subconjunto del CAB
antes del despliegue En este caso, nuestro objetivo sigue siendo garantizar que podamos implementar rápidamente,
incluso si no está completamente automatizado.

En este caso, debemos asegurarnos de que las solicitudes de cambio enviadas sean tan completas y
lo más preciso posible, dándole al CAB todo lo que necesitan para evaluar adecuadamente nuestro
cambio: después de todo, si nuestra solicitud de cambio tiene un formato incorrecto o está incompleta, será rechazada
de nuevo a nosotros, aumentando el tiempo requerido para que entremos en producción y arrojemos dudas
sobre si realmente entendemos los objetivos del proceso de gestión del cambio.

Es casi seguro que podemos automatizar la creación de RFC completos y precisos,
llenar el ticket con detalles de exactamente lo que se va a cambiar. Por ejemplo, podríamos
cree automáticamente un ticket de cambio de ServiceNow con un enlace a la historia de usuario de JIRA, junto
con los manifiestos de compilación y la salida de prueba de nuestra herramienta de canalización de implementación y enlaces a
los scripts de Puppet / Chef que se ejecutarán.

Debido a que nuestros cambios enviados serán evaluados manualmente por personas, es aún más
importante que describamos el contexto del cambio. Esto incluye identificar por qué
están realizando el cambio (por ejemplo, proporcionando un enlace a las características, defectos o incidentes), quién
se ve afectado por el cambio y lo que se va a cambiar.

Nuestro objetivo es compartir la evidencia y los artefactos que nos dan confianza de que el cambio
operará en producción según lo diseñado. Aunque los RFC suelen tener texto de forma libre
campos, debemos proporcionar enlaces a datos legibles por máquina para permitir que otros se integren
y procesar nuestros datos (por ejemplo, enlaces a archivos JSON).

En muchas cadenas de herramientas, esto se puede hacer de una manera compatible y totalmente automatizada. por
ejemplo, Mingle and Go de ThoughtWorks puede vincular automáticamente esta información
juntos, como una lista de defectos corregidos y nuevas características completadas que están asociadas
con el cambio, y ponerlo en un RFC.

Tras la presentación de nuestro RFC, los miembros relevantes del CAB revisarán, procesarán y
apruebe estos cambios como lo harían con cualquier otra solicitud de cambio presentada. Si todo va
bueno, las autoridades de cambio apreciarán la minuciosidad y el detalle de nuestro envío
cambios, porque les hemos permitido validar rápidamente la corrección de la
información que hemos proporcionado (por ejemplo, ver los enlaces a artefactos de nuestra implementación
herramientas de tubería). Sin embargo, nuestro objetivo debe ser mostrar continuamente una pista ejemplar

registro de cambios exitosos, para que eventualmente podamos obtener su acuerdo de que nuestro
Los cambios automatizados se pueden clasificar de forma segura como cambios estándar.

Salesforce fue fundada en 2000 con el objetivo de establecer una relación con el cliente. gestión fácilmente disponible y entregable como servicio. Ofertas de Salesforce fueron ampliamente adoptados por el mercado, lo que condujo a una OPI exitosa en 2004. Por 2007, la compañía tenía más de cincuenta y nueve mil clientes empresariales, procesando cientos de millones de transacciones por día, con ingresos anuales de $ 497 millones.

Sin embargo, más o menos al mismo tiempo, su capacidad para desarrollar y lanzar nuevos La funcionalidad para sus clientes parecía detenerse. En 2006, tenían cuatro principales lanzamientos de clientes, pero en 2007 solo pudieron hacer un cliente liberar a pesar de haber contratado más ingenieros. El resultado fue que el número de las funciones entregadas por equipo se mantuvieron decrecientes y los días entre lanzamientos principales siguió aumentando

Y debido a que el tamaño del lote de cada versión se hizo cada vez mayor, la implementación Los resultados también empeoraron. Karthik Rajan, entonces vicepresidente de infraestructura Ingeniería, informa en una presentación de 2013 que 2007 marcó "el último año cuando El software fue creado y enviado utilizando un proceso en cascada y cuando hicimos nuestro cambio a un proceso de entrega más incremental ".

En la Cumbre de DevOps Enterprise 2014, Dave Mangot y Reena Mathew describió la transformación DevOps multianual resultante que comenzó en 2009. Según Mangot y Mathew, al implementar los principios de DevOps y prácticas, la compañía redujo los plazos de implementación de seis días a cinco minutos para 2013. Como resultado, pudieron escalar la capacidad más fácilmente, permitiéndoles procesar más de mil millones de transacciones por día.

Uno de los temas principales de la transformación de Salesforce fue hacer que la calidad diseñar el trabajo de todos, independientemente de si formaban parte de Desarrollo, Operaciones, o Infosec. Para hacer esto, integraron pruebas automatizadas en todas las etapas de la creación de aplicaciones y entornos, así como en el continuo proceso de integración y despliegue, y creó la herramienta de código abierto Rouster para realizar pruebas funcionales de sus módulos Puppet.

También comenzaron a realizar *pruebas destructivas de* forma rutinaria , un término utilizado en fabricación para referirse a la realización de pruebas de resistencia prolongadas bajo la mayoría condiciones de funcionamiento severas hasta que se destruya el componente que se está probando. los El equipo de Salesforce comenzó a probar sus servicios de forma rutinaria bajo un nivel cada vez mayor carga hasta que el servicio se rompió, lo que les ayudó a comprender sus modos de falla y hacer las correcciones apropiadas. Como era de esperar, el resultado fue significativamente Mayor calidad de servicio con cargas de producción normales.

Information Security también trabajó con Quality Engineering en las primeras etapas de su proyecto, colaborando continuamente en fases críticas como la arquitectura y

diseño de prueba, así como la integración adecuada de herramientas de seguridad en las pruebas automatizadas proceso.

Para Mangot y Mathew, uno de los éxitos clave de toda la repetibilidad y el rigor que diseñaron en el proceso les decía su gestión de cambios grupo que "los cambios de infraestructura realizados a través de Puppet ahora se tratarían como 'cambios estándar', que requieren mucho menos o incluso ninguna otra aprobación del CAB ". Además, señalaron que "los cambios manuales en la infraestructura aún requerirían aprobaciones ".

Al hacer esto, no solo habían integrado sus procesos DevOps con el cambio el proceso de gestión, pero también creó más motivación para automatizar el proceso de cambio para más de su infraestructura.

REDUCIR LA CONFIANZA EN LA SEPARACIÓN DEL DERECHO

Durante décadas, hemos utilizado la separación de tareas como uno de nuestros controles principales para reducir El riesgo de fraude o errores en el proceso de desarrollo de software. Ha sido el práctica aceptada en la mayoría de los SDLC para exigir que los cambios de desarrollador se envíen a un código bibliotecario, que revisará y aprobará el cambio antes de que IT Operations promocione cambio a producción.

Hay muchos otros ejemplos menos polémicos de separación de tareas en el trabajo de Ops, tal como los administradores del servidor, idealmente, pueden ver los registros pero no eliminarlos ni modificarlos ellos, para evitar que alguien con acceso privilegiado elimine evidencia de fraude u otros problemas

Cuando realizamos despliegues de producción con menos frecuencia (por ejemplo, anualmente) y cuando nuestro trabajo era menos complejo, compartimentando nuestro trabajo y haciendo transferencias eran formas sostenibles de hacer negocios. Sin embargo, a medida que aumenta la complejidad y la frecuencia de implementación, realizar implementaciones de producción con éxito requiere cada vez más de todos en el flujo de valor para ver rápidamente los resultados de sus acciones.

La separación del deber a menudo puede impedir esto al reducir la velocidad y reducir la retroalimentación Los ingenieros reciben en su trabajo. Esto evita que los ingenieros asuman toda la responsabilidad. por la calidad de su trabajo y reduce la capacidad de una empresa para crear aprendizaje organizacional.

En consecuencia, siempre que sea posible, debemos evitar utilizar la separación de funciones como controlar. En su lugar, debemos elegir controles como la programación de pares, continua inspección de los registros de código y revisión de código. Estos controles nos pueden dar lo necesario tranquilidad sobre la calidad de nuestro trabajo. Además, al poner estos controles en lugar, si se requiere la separación de funciones, podemos demostrar que logramos un equivalente resultados con los controles que hemos creado.

Bill Massie es gerente de desarrollo en Etsy y es responsable del pago aplicación llamada ICHT (una abreviatura de "I Can Haz Tokens"). ICHT toma

órdenes de crédito de clientes a través de un conjunto de procesamiento de pagos desarrollado internamente aplicaciones que manejan la entrada de pedidos en línea tomando el titular de la tarjeta ingresado por el cliente datos, tokenizarlos, comunicarse con el procesador de pagos y completar el orden de transacción. ¶

Debido al alcance de las Normas de seguridad de datos de la industria de tarjetas de pago (PCI DSS) el entorno de datos del titular de la tarjeta (CDE) es "las personas, los procesos y Tecnología que almacena, procesa o transmite datos de titulares de tarjetas o información confidencial datos de autenticación ", incluidos los componentes del sistema conectados, el ICHT La aplicación tiene alcance para PCI DSS.

Para contener el alcance PCI DSS, la aplicación ICHT es física y lógicamente separado del resto de la organización Etsy y es administrado por un equipo de aplicación independiente de desarrolladores, ingenieros de bases de datos, redes ingenieros e ingenieros de operaciones. Cada miembro del equipo recibe dos computadoras portátiles: una para ICHT (que están configurados de manera diferente para cumplir con los requisitos de DSS, así como estar encerrado en una caja fuerte cuando no esté en uso) y una para el resto de Etsy.

Al hacer esto, pudieron desacoplar el entorno CDE del resto de la organización Etsy, limitando el alcance de las regulaciones PCI DSS a uno Área segregada. Los sistemas que forman el CDE están separados (y administrados diferente) del resto de los entornos de Etsy en la fuente física, de red, código y niveles de infraestructura lógica. Además, el CDE se construye y opera por un equipo multifuncional que es el único responsable del CDE.

El equipo de ICHT tuvo que modificar sus prácticas de entrega continua para acomodar la necesidad de aprobaciones de código. De acuerdo con la Sección 6.3.2 de la PCI DSS v3.1, los equipos deben revisar:

Todo el código personalizado antes del lanzamiento a producción o clientes para identificar cualquier vulnerabilidad potencial de codificación (usando procesos manuales o automatizados) como sigue:

> ¿Los cambios de código son revisados por personas distintas del autor del código de origen, y por personas conocedoras de las técnicas de revisión de código y seguras prácticas de codificación?
>
> Las revisiones de código hacen que el código se desarrolle de acuerdo con la codificación segura pautas?
>
> ¿Se implementan las correcciones apropiadas antes del lanzamiento?
>
> ¿Los resultados de la revisión del código son revisados y aprobados por la gerencia antes del lanzamiento?

Para cumplir con este requisito, el equipo inicialmente decidió designar a Massie como aprobador de cambios responsable de implementar cualquier cambio en la producción. Deseado los despliegues se marcarían en JIRA, y Massie los marcaría como revisados y aprobado, y desplegarlos manualmente en la producción de ICHT.

Esto ha permitido a Etsy cumplir con los requisitos de PCI DSS y obtener su firma Informe de cumplimiento de sus asesores. Sin embargo, con respecto al equipo,

Se han producido problemas importantes.

Massie observa que un efecto secundario preocupante "es un nivel de 'compartimentación' eso está sucediendo en el equipo de ICHT que ningún otro grupo está teniendo en Etsy. Nunca desde que implementamos la separación de tareas y otros controles requeridos por el PCI Conformidad con DSS, nadie puede ser un ingeniero de pila completa en este entorno ".

Como resultado, mientras el resto de los equipos de Desarrollo y Operaciones en Etsy trabajan juntos de cerca e implementan cambios sin problemas y con confianza, señala Massie que "dentro de nuestro entorno PCI, hay temor y renuencia en torno a la implementación y mantenimiento porque nadie tiene visibilidad fuera de su parte del software apilar. Los cambios aparentemente menores que hicimos en la forma en que trabajamos parecen tener creó un muro impenetrable entre desarrolladores y operaciones, y crea un tensión innegable que nadie en Etsy ha tenido desde 2008. Incluso si tiene confianza en su porción, es imposible confiar en que alguien más el cambio no va a romper tu parte de la pila ".

Este estudio de caso muestra que el cumplimiento es posible en organizaciones que usan DevOps. Sin embargo, la historia potencialmente precautoria aquí es que todas las virtudes que tenemos asociarse con equipos DevOps de alto rendimiento son frágiles, incluso un equipo que tiene Las experiencias compartidas con alta confianza y objetivos compartidos pueden comenzar a tener dificultades cuando Se implementan mecanismos de control de baja confianza.

ASEGURAR LA DOCUMENTACIÓN Y LA PRUEBA PARA LOS AUDITORES Y OFICIALES DE CUMPLIMIENTO

A medida que las organizaciones tecnológicas adoptan cada vez más los patrones de DevOps, hay más tensión que nunca entre TI y auditoría. Estos nuevos patrones de DevOps desafían a los tradicionales pensando en auditorías, controles y mitigación de riesgos.

Como observa Bill Shinn, arquitecto principal de soluciones de seguridad en Amazon Web Services, “DevOps se trata de cerrar la brecha entre Dev y Ops. De alguna manera, el desafío de cerrar la brecha entre DevOps y los auditores y los oficiales de cumplimiento es aún mayor. Por ejemplo, cuántos auditores pueden leer el código y cuántos desarrolladores han leído NIST 800-37 o la Ley Gramm-Leach-Bliley? Eso crea una brecha de conocimiento, y el

La comunidad DevOps necesita ayudar a cerrar esa brecha ”.

Ayudar a los clientes de grandes empresas a demostrar que aún pueden cumplir con todos los requisitos relevantes. Las leyes y regulaciones están entre las responsabilidades de Bill Shinn como principal seguridad. Arquitecto de soluciones en Amazon Web Services. Con los años, ha pasado tiempo con más de mil clientes empresariales, incluidos Hearst Media, GE, Phillips, y Pacific Life, que han hecho referencia públicamente a su uso de nubes públicas en ambientes regulados.

Shinn señala: "Uno de los problemas es que los auditores han recibido capacitación en métodos que no son muy adecuados para los patrones de trabajo de DevOps. Por ejemplo, si un auditor vio

un entorno con diez mil servidores de producciones, han sido tradicionalmente capacitado para solicitar una muestra de mil servidores, junto con Captura de pantalla de evidencia de gestión de activos, configuración de control de acceso, agente instalaciones, registros del servidor, etc. "

"Eso estuvo bien con los entornos físicos", continúa Shinn. "Pero cuando la infraestructura es código, y cuando el escalado automático hace que los servidores aparezcan y desaparezcan todo el tiempo, ¿cómo muestras eso? Te encuentras con los mismos problemas cuando tener un canal de implementación, que es muy diferente al software tradicional proceso de desarrollo, donde un grupo escribe el código y otro grupo implementa ese código en producción ".

Él explica: "En el trabajo de campo de auditoría, los métodos más comunes de recolección la evidencia sigue siendo capturas de pantalla y archivos CSV llenos de ajustes de configuración y troncos Nuestro objetivo es crear métodos alternativos para presentar los datos que claramente mostrar a los auditores que nuestros controles son operativos y efectivos ".

Para ayudar a cerrar esa brecha, tiene equipos que trabajan con auditores en el diseño de control proceso. Usan un enfoque iterativo, asignando un solo control para cada sprint para determinar lo que se necesita en términos de evidencia de auditoría. Esto ha ayudado a garantizar que los auditores obtienen la información que necesitan cuando el servicio está en producción, enteramente bajo demanda.

Shinn afirma que la mejor manera de lograr esto es "enviar todos los datos a nuestro sistemas de telemetría, como Splunk o Kibana. De esta forma, los auditores pueden obtener lo que necesidad, completamente autoservicio. No necesitan solicitar una muestra de datos: en cambio, inician sesión en Kibana y luego buscan evidencia de auditoría que necesitan para un rango de tiempo dado. Idealmente, verán muy rápidamente que hay evidencia para apoyar que nuestros controles están funcionando ".

Shinn continúa: "Con el registro de auditoría moderno, las salas de chat y la implementación tuberías, hay visibilidad y transparencia sin precedentes de lo que está sucediendo en producción, especialmente en comparación con la forma en que se realizaban las operaciones, con mucho menor probabilidad de que se introduzcan errores y fallas de seguridad. Entonces, el desafío es para convertir toda esa evidencia en algo que un auditor reconoce ".

Eso requiere derivar los requisitos de ingeniería de las regulaciones reales. Shinn explica: "Para descubrir lo que HIPAA requiere de una seguridad de la información perspectiva, debe examinar la legislación de cuarenta y cinco CFR Parte 160, entrar en Subparte A y C de la Parte 164. Aun así, debe seguir leyendo hasta que obtenga en "salvaguardas técnicas y controles de auditoría". Solo allí verás que lo que es es necesario que determinemos las actividades que serán rastreadas y auditadas relevante para la información de atención médica del paciente, documentar e implementar aquellos controles, seleccione herramientas y finalmente revise y capture el apropiado información."

Shinn continúa: "Cómo cumplir ese requisito es la discusión que debe ser sucediendo entre el cumplimiento y los oficiales reguladores, y la seguridad y Equipos de DevOps, específicamente sobre cómo prevenir, detectar y corregir problemas.

349 de 1189.

A veces se pueden cumplir en una configuración en el control de versiones. Otro veces, es un control de monitoreo ".

Shinn da un ejemplo: "Podemos elegir implementar uno de esos controles usando AWS CloudWatch, y podemos probar que el control funciona con uno línea de comando. Además, tenemos que mostrar a dónde van los registros, en el ideal, empujamos todo esto a nuestro marco de registro, donde podemos vincular la auditoría evidencia con el requisito de control real ".

Para ayudar a resolver este problema, *DevOps Audit Defense Toolkit* describe el final narrativa completa del proceso de cumplimiento y auditoría para una organización ficticia (Partes ilimitadas de *The Phoenix Project*). Comienza describiendo la entidad objetivos organizacionales, procesos comerciales, riesgos principales y control resultante entorno, así como cómo la administración podría probar con éxito que los controles existen y son efectivas. También se presenta un conjunto de objeciones de auditoría, así como cómo para superarlos

El documento describe cómo podrían diseñarse los controles en una tubería de implementación para mitigar los riesgos establecidos, y proporciona ejemplos de declaraciones de control y artefactos de control para demostrar la efectividad del control. Estaba destinado a ser general a todos los objetivos de control, incluso en apoyo de informes financieros precisos, cumplimiento normativo (por ejemplo, SEC SOX-404, HIPAA, FedRAMP, modelo de la UE

Contratos, y los reglamentos propuestos SEC Reg-SCI), obligaciones contractuales (por ejemplo, PCI DSS, DOD DISA) y operaciones efectivas y eficientes.

Mary Smith (un seudónimo) encabeza la iniciativa DevOps para el consumidor propiedad bancaria de una gran organización de servicios financieros de los Estados Unidos. Ella hizo el observación de que la seguridad de la información, los auditores y los reguladores a menudo ponen demasiado dependencia de revisiones de código para detectar fraude. En cambio, deberían confiar en controles de monitoreo de producción además de usar pruebas automatizadas, código revisiones y aprobaciones para mitigar efectivamente los riesgos asociados con errores y fraude.

Ella observó:

> Hace muchos años, tuvimos un desarrollador que plantó una puerta trasera en el código que implementamos en nuestros cajeros automáticos. Pudieron poner los cajeros automáticos en modo de mantenimiento en ciertos momentos, lo que les permite sacar efectivo de Las máquinas. Pudimos detectar el fraude muy rápidamente, y no fue a través de una revisión de código. Estos tipos de puertas traseras son difíciles, o incluso imposible, detectar cuándo los perpetradores tienen suficientes medios, motivos, y oportunidad.
>
> Sin embargo, detectamos rápidamente el fraude durante nuestras operaciones regulares reunión de revisión cuando alguien notó que se estaban poniendo cajeros automáticos en una ciudad en modo de mantenimiento en momentos no programados. Encontramos el fraude incluso

> antes del proceso de auditoría de efectivo programado, cuando concilian la cantidad de efectivo en los cajeros automáticos con transacciones autorizadas.

En este estudio de caso, el fraude ocurrió a pesar de la separación de deberes entre Desarrollo y operaciones y un proceso de aprobación de cambios, pero fue rápido detectado y corregido mediante telemetría de producción efectiva.

CONCLUSIÓN

A lo largo de este capítulo, hemos discutido prácticas que hacen que la seguridad de la información el trabajo de todos, donde todos nuestros objetivos de seguridad de la información están integrados en el trabajo diario de todos en la cadena de valor. Al hacer esto, mejoramos significativamente la efectividad de nuestros controles, para que podamos prevenir mejor las brechas de seguridad, así como detectar y recuperarse de ellos más rápido. Y reducimos significativamente el trabajo asociado

con la preparación y aprobación de auditorías de cumplimiento.

PARTE VI CONCLUSIÓN

A lo largo de los capítulos anteriores, exploramos cómo tomar los principios de DevOps y aplicar
ellos a la Seguridad de la Información, ayudándonos a alcanzar nuestros objetivos y asegurándonos de que la seguridad
parte del trabajo de todos, todos los días. Una mejor seguridad garantiza que seamos defendibles y
sensible con nuestros datos, que podemos recuperarnos de los problemas de seguridad antes de que se conviertan

catastrófico y, lo más importante, que podemos hacer la seguridad de nuestros sistemas y datos mejor que nunca.

† ITIL define la utilidad como "lo que hace el servicio", mientras que la garantía se define como "cómo se entrega el servicio y se puede utilizar para determinar si un el servicio es "apto para su uso".

‡ Para gestionar aún más los cambios de riesgo, también podemos tener reglas definidas, como ciertos cambios que solo pueden ser implementados por un determinado grupo o individuo (por ejemplo, solo los DBA pueden implementar cambios en el esquema de la base de datos). Tradicionalmente, las reuniones de CAB se han celebrado semanalmente, donde las solicitudes de cambio son aprobado y programado. A partir de la versión 3 de ITIL, es aceptable que los cambios se aprueben electrónicamente de manera justo a tiempo a través de un herramienta de gestión del cambio. También recomienda específicamente que "los cambios estándar se identifiquen desde el principio al construir el cambio Proceso de gestión para promover la eficiencia. De lo contrario, una implementación de Gestión de cambios puede crear niveles de administración innecesariamente altos y resistencia al proceso de Gestión del Cambio ".

§ El *boleto de* término se usa genéricamente para indicar cualquier elemento de trabajo identificable de forma única.

¶ Los autores agradecen a Bill Massie y John Allspaw por pasar un día entero con Gene Kim compartiendo su experiencia de cumplimiento.

Una llamada a la acción

Conclusión a los DevOps Manual

Hemos llegado al final de una exploración detallada de tanto los principios como las prácticas técnicas de DevOps. A un momento en que cada líder tecnológico es desafiado con permitiendo seguridad, confiabilidad y agilidad, y a la vez cuando fallas de seguridad, tiempo de comercialización y masivo La transformación de la tecnología está teniendo lugar, DevOps ofrece una solución. Con suerte, este libro ha proporcionado una profunda comprensión del problema y una hoja de ruta para creando soluciones relevantes.

Como hemos explorado a lo largo de *The DevOps Handbook* , sabemos que, si no se gestiona, un conflicto inherente puede existe entre Desarrollo y Operaciones que crea problemas cada vez más graves, lo que resulta en un tiempo más lento para mercado de nuevos productos y características, mala calidad, aumento de interrupciones y deuda técnica, reducción productividad de ingeniería, así como un aumento de empleados insatisfacción y agotamiento.

Los principios y patrones de DevOps nos permiten romper esto núcleo, conflicto crónico. Después de leer este libro, esperamos puede ver cómo una transformación DevOps puede habilitar creación de organizaciones dinámicas de aprendizaje, logrando Los sorprendentes resultados del flujo rápido y de clase mundial fiabilidad y seguridad, así como mayor

competitividad y satisfacción de los empleados.

Página 354

DevOps requiere potencialmente nuevas actividades culturales y normas de gestión y cambios en nuestra técnica prácticas y arquitectura. Esto requiere una coalición que abarca liderazgo empresarial, gestión de productos, Desarrollo, QA, Operaciones de TI, Seguridad de la información, e incluso Marketing, donde muchas iniciativas tecnológicas originar. Cuando todos estos equipos trabajan juntos, podemos crear un sistema de trabajo seguro que permita a los equipos pequeños Desarrollar y validar código de forma rápida e independiente que se puede implementar de manera segura a los clientes. Esto resulta en maximizando la productividad del desarrollador, organizacional aprendizaje, alta satisfacción de los empleados y la capacidad de ganar en el mercado

Nuestro objetivo al escribir este libro era codificar suficientemente Principios y prácticas de DevOps para que lo sorprendente los resultados logrados dentro de la comunidad DevOps podrían ser replicado por otros. Esperamos acelerar el adopción de iniciativas DevOps y apoyar sus implementaciones exitosas mientras se reduce la activación energía requerida para que se completen.

Conocemos los peligros de posponer mejoras y conformarse con soluciones diarias, así como las dificultades de cambiar la forma en que priorizamos y realizamos nuestro trabajo diario trabajo. Además, entendemos los riesgos y el esfuerzo.

requerido para que las organizaciones adopten una forma diferente de trabajo, así como la percepción de que DevOps es otra moda pasajera, que pronto será reemplazada por la siguiente palabra pegadiza.

Afirmamos que DevOps es transformador en la forma en que realizar trabajos tecnológicos, como Lean forever

transformado cómo se realizó el trabajo de fabricación en la década de 1980. Aquellos que adopten DevOps ganarán en el mercado, a expensas de los que no lo hacen. Ellos creará energía y aprendizaje continuo organizaciones que superan e innovan sus innovaciones competidores.

Debido a esto, DevOps no es solo una tecnología imperativo, pero también un imperativo organizacional. los En pocas palabras, DevOps es aplicable y relevante para cualquier y todas las organizaciones que deben aumentar el flujo de planificación trabajar a través de la organización tecnológica, mientras manteniendo la calidad, confiabilidad y seguridad para nuestros clientes.

Nuestro llamado a la acción es este: no importa en qué papel juegues su organización, comience a encontrar personas a su alrededor que desea cambiar la forma en que se realiza el trabajo. Muestra este libro a otros y crear una coalición de pensadores de ideas afines para salir de la espiral descendente. Pregunta organizacional líderes para apoyar estos esfuerzos o, mejor aún, patrocinar y lidera estos esfuerzos tú mismo.

Finalmente, ya que has llegado hasta aquí, tenemos un sucio
secreto para revelar. En muchos de nuestros estudios de caso, a continuación
el logro de los resultados innovadores presentados,
muchos de los agentes de cambio fueron promovidos, pero en algunos
casos, luego hubo un cambio de liderazgo que
resultó en que muchas de las personas involucradas se fueran,
acompañado de un retroceso de la organización
cambios que habían creado.

Creemos que es importante no ser cínico sobre esto
posibilidad. Las personas involucradas en estas transformaciones.
sabía de antemano que lo que estaban haciendo tenía un alto
posibilidad de fracaso, y lo hicieron de todos modos. Al hacerlo,
quizás lo más importante, nos inspiraron al resto de nosotros
mostrándonos qué se puede hacer. La innovación es imposible
sin correr riesgos, y si no has logrado molestar
al menos algunas personas en la gerencia, probablemente no
intentando lo suficiente. No dejes que tu organización
El sistema inmunitario lo disuade o distrae de su visión.
Como Jesse Robbins, anteriormente "maestro del desastre" en
A Amazon le gusta decir: "No luches estúpido, haz más
increíble."

DevOps nos beneficia a todos en el flujo de valor de la tecnología,
si somos Dev, Ops, QA, Infosec, propietarios de productos,
o clientes. Trae alegría al desarrollo excelente

productos, con menos marchas de la muerte. Permite humano condiciones de trabajo con menos fines de semana y perdidas Vacaciones con nuestros seres queridos. Permite a los equipos trabajar juntos para sobrevivir, aprender, prosperar, deleitar a nuestros clientes, y ayudar a nuestra organización a tener éxito.

Esperamos sinceramente que *el manual DevOps lo* ayude lograr estos objetivos

Apéndices

APÉNDICE 1 LA CONVERGENCIA DE LOS DEVOPS

Creemos que DevOps se está beneficiando de una increíble convergencia de gestión movimientos, que se refuerzan mutuamente y pueden ayudar a crear un poderoso coalición para transformar cómo las organizaciones desarrollan y entregan productos de TI y servicios.

John Willis llamó a esto "la convergencia de DevOps". Los diversos elementos de esto La convergencia se describe a continuación en orden cronológico aproximado. (Tenga en cuenta que estos las descripciones no pretenden ser una descripción exhaustiva, sino simplemente lo suficiente para mostrar la progresión del pensamiento y las conexiones bastante improbables que llevaron a

DevOps.)

EL MOVIMIENTO LEAN

El movimiento Lean comenzó en la década de 1980 como un intento de codificar el Toyota Sistema de producción con la popularización de técnicas como Value Stream Mapeo, tableros kanban y mantenimiento productivo total.

Dos principios fundamentales de Lean eran la creencia profundamente arraigada de que el tiempo de entrega (es decir, el tiempo requerido para convertir las materias primas en productos terminados) fue el mejor predictor de calidad, satisfacción del cliente y felicidad de los empleados; y ese uno de los mejores los predictores de tiempos de entrega cortos fueron lotes pequeños, siendo el ideal teórico "Flujo de una sola pieza" (es decir, flujo "1x1": inventario de 1, tamaño de lote de 1).

Los principios Lean se centran en crear valor para el cliente, pensando sistemáticamente, creando constancia de propósito, abrazando el pensamiento científico, creando flujo y atracción (versus empuje), asegurando la calidad en la fuente, liderando con humildad y respetando cada individuo.

EL MOVIMIENTO ÁGIL

Comenzado en 2001, el Manifiesto Ágil fue creado por diecisiete de los principales pensadores. en desarrollo de software, con el objetivo de convertir métodos livianos como DP y DSDM en un movimiento más amplio que podría asumir el desarrollo de software pesado procesos como el desarrollo en cascada y metodologías como el Rational Proceso unificado.

Un principio clave era "entregar software de trabajo con frecuencia, desde un par de semanas hasta un par de meses, con preferencia a la escala de tiempo más corta ". Otros dos principios centrarse en la necesidad de equipos pequeños y motivados, que trabajen en un entorno de alta confianza

modelo de gestión y énfasis en lotes pequeños. Ágil también está asociado con un conjunto de herramientas y prácticas como Scrum, Standups, etc.

EL MOVIMIENTO DE LA CONFERENCIA DE VELOCIDAD

Iniciada en 2007, la Conferencia Velocity fue creada por Steve Souders, John Allspaw, y Jesse Robbins para proporcionar un hogar para las operaciones de TI y el rendimiento web tribu. En la conferencia Velocity 2009, John Allspaw y Paul Hammond dieron el seminal "10 implementaciones por día: cooperación entre desarrolladores y operaciones en Flickr".

EL MOVIMIENTO DE INFRAESTRUCTURA ÁGIL

En la conferencia Agile Toronto de 2008, Patrick Dubois y Andrew Schafer realizaron una

Sesión "pájaros de una pluma" sobre la aplicación de los principios ágiles a la infraestructura en lugar de al código de la aplicación. Rápidamente ganaron seguidores de pensadores de ideas afines, incluyendo John Willis Más tarde, Dubois estaba tan emocionado por "10 Deploys por Allspaw y Hammond Día: Dev y Ops Cooperación en Flickr "presentación que creó la primera DevOpsDays en Gante, Bélgica, en 2009, acuñando la palabra "DevOps".

EL MOVIMIENTO CONTINUO DE ENTREGA

Basándose en la disciplina de desarrollo de construcción, prueba e integración continuas, Jez Humble y David Farley ampliaron el concepto de entrega continua, que incluyó una "tubería de implementación" para garantizar que el código y la infraestructura estén siempre en un estado desplegable y que todo el código registrado en el camión se implementa en producción.

Esta idea se presentó por primera vez en Agile 2006 y también fue desarrollada independientemente por Tim Fitz en una publicación de blog titulada "Implementación continua".

EL MOVIMIENTO TOYOTA KATA

En 2009, Mike Rother escribió *Toyota Kata: Gestión de personas para mejorar, Adaptabilidad y resultados superiores* , que describieron los aprendizajes durante sus veinte años viaje para comprender y codificar los mecanismos causales de la producción de Toyota Sistema. *Toyota Kata* describe las "rutinas gerenciales invisibles y pensar que mentir detrás del éxito de Toyota con la mejora continua y la adaptación ... y cómo otras compañías desarrollan rutinas y pensamientos similares en sus organizaciones ".

Su conclusión fue que la comunidad Lean perdió la práctica más importante de todo, que describió como el Kata de mejora. Él explica que cada organización tiene rutinas de trabajo, y el factor crítico en Toyota fue hacer que el trabajo de mejora habitual, y convertirlo en el trabajo diario de todos en la organización. los Toyota Kata instituye un enfoque iterativo, incremental y científico del problema resolviendo en la búsqueda de un verdadero norte organizacional compartido.

EL MOVIMIENTO LEAN STARTUP

En 2011, Eric Ries escribió *The Lean Startup: cómo utilizan los emprendedores de hoy Innovación continua para crear negocios radicalmente exitosos* , codificando su lecciones aprendidas en IMVU, una startup de Silicon Valley, que se basó en el trabajo de Steve En blanco en *Los cuatro pasos para la Epifanía* , así como en el despliegue continuo

técnicas Eric Ries también codificó prácticas y términos relacionados, incluyendo Mínimo Producto viable, el ciclo de construcción-medida-aprendizaje y muchas implementaciones continuas Patrones técnicos.

EL MOVIMIENTO LEAN UX

En 2013, Jeff Gothelf escribió *Lean UX: Aplicación de principios Lean para mejorar el usuario Experiencia* , que codificó cómo mejorar el "front end difuso" y explicó cómo los propietarios de productos pueden enmarcar hipótesis comerciales, experimentar y ganar confianza en esas hipótesis comerciales antes de invertir tiempo y recursos en el resultado caracteristicas. Al agregar Lean UX, ahora tenemos las herramientas para optimizar completamente el flujo entre hipótesis comerciales, desarrollo de características, pruebas, implementación y prestación de servicios al cliente.

EL MOVIMIENTO DE COMPUTACIÓN ROBUSTO

En 2011, Joshua Corman, David Rice y Jeff Williams examinaron la aparente inutilidad. de asegurar aplicaciones y entornos al final del ciclo de vida. En respuesta, ellos creó una filosofia llamada "Computación robusta", que intenta enmarcar la no requisitos funcionales de estabilidad, escalabilidad, disponibilidad, capacidad de supervivencia, sostenibilidad, seguridad, capacidad de soporte, capacidad de administración y defensa.

Debido al potencial de altas tasas de liberación, DevOps puede ejercer una presión increíble sobre QA e Infosec, porque cuando las tasas de implementación van de mensuales o trimestrales a cientos o miles por día, ya no son tiempos de respuesta de dos semanas de Infosec o QA sostenible. El movimiento de Computación Robusta postuló que el enfoque actual para luchando contra el complejo industrial vulnerable empleado por la mayoría de la información Los programas de seguridad no tienen remedio.

APÉNDICE 2 TEORÍA DE RESTRICCIONES Y NÚCLEO, CONFLICTOS CRONICOS

El cuerpo de conocimiento de Theory of Restraints discute ampliamente el uso de crear nubes centrales de conflicto (a menudo denominadas "C3"). Aquí está la nube de conflictos para TI:

Figura 46: *El conflicto central y crónico que enfrenta cada organización de TI*

Durante la década de 1980, hubo un conflicto crónico central muy conocido en fabricación. Cada gerente de planta tenía dos objetivos comerciales válidos: proteger las ventas y reducir costos. El problema era que para proteger las ventas, la gestión de ventas era incentivado a aumentar el inventario para garantizar que siempre sea posible cumplir demanda del cliente

Por otro lado, para reducir los costos, se incentivó la gestión de la producción. para disminuir el inventario para asegurar que el dinero no esté atado en el trabajo en progreso que no se podía enviar inmediatamente al cliente en forma de ventas realizadas.

Pudieron romper el conflicto mediante la adopción de principios Lean, como reducir tamaños de lote, reduciendo el trabajo en proceso y acortando y amplificando los circuitos de retroalimentación. Esto dio lugar a aumentos dramáticos en la productividad de la planta, la calidad del producto y la satisfacción del cliente.

Los principios detrás de los patrones de trabajo de DevOps son los mismos que los que transformaron fabricación, lo que nos permite optimizar el flujo de valor de TI, convirtiendo negocios necesidades en capacidades y servicios que brinden valor a nuestros clientes.

APÉNDICE 3 FORMA TABULAR DE LA ESPIRAL HACIA ABAJO

La forma columnar de la espiral descendente representado en *El Proyecto Phoenix* se muestra abajo:

Tabla 4: *La espiral descendente*

APÉNDICE 4 LOS PELIGROS DE HANDOFFS Y Colas

El problema con altas cantidades de tiempo de cola se exacerba cuando hay muchos transferencias, porque ahí es donde se crean las colas. La Figura 47 muestra el tiempo de espera como función de qué tan ocupado está un recurso en un centro de trabajo. La curva asintótica muestra por qué un El "cambio simple de treinta minutos" a menudo lleva semanas en completarse: ingenieros específicos y los centros de trabajo a menudo se convierten en cuellos de botella problemáticos cuando operan a altas utilización. Cuando un centro de trabajo se acerca al 100% de utilización, cualquier trabajo que se requiera de él languidecerá en las colas y no se trabajará sin alguien acelerar / escalar.

Figura 47: *Tamaño de la cola y tiempos de espera en función del porcentaje de utilización (Fuente: Kim, Behr y Spafford,* The Phoenix Project *, edición ePub, 557.)*

En la figura 47, el eje x es el porcentaje ocupado para un recurso dado en un centro de trabajo, y el eje y es el tiempo de espera aproximado (o, más exactamente, la longitud de la cola). Lo que muestra la forma de la línea es que a medida que la utilización de los recursos supera el 80%, espere el tiempo pasa por el techo.

En *The Phoenix Project* , así es como Bill y su equipo se dieron cuenta de la devastadora consecuencias de esta propiedad en los plazos de entrega de los compromisos que estaban haciendo para la oficina de gestión del proyecto:

> Les cuento sobre lo que Erik me contó en MRP-8, sobre cómo dependen los tiempos de espera utilización de recursos. "El tiempo de espera es el 'porcentaje de tiempo ocupado' dividido por el 'porcentaje de tiempo inactivo'. En otras palabras, si un recurso está ocupado en un cincuenta por ciento, entonces es cincuenta por ciento inactivo. El tiempo de espera es del cincuenta por ciento dividido entre el cincuenta por ciento, por lo que una unidad

de tiempo. Llamémoslo una hora.

Entonces, en promedio, nuestra tarea esperaría en la cola durante una hora antes de que llegue trabajó.

"Por otro lado, si un recurso está ocupado el noventa por ciento, el tiempo de espera es 'noventa por ciento dividido por diez por ciento 'o nueve horas. En otras palabras, nuestra tarea esperaría en cola nueve veces más que si el recurso estuviera inactivo en un cincuenta por ciento ".

Concluyo: "Entonces ... Para la tarea de Phoenix, suponiendo que tengamos siete traspasos, y que cada uno de esos recursos está ocupado el noventa por ciento del tiempo, las tareas pasarían en haga cola un total de nueve horas por los siete pasos ... "

"¿Qué? ¿Sesenta y tres horas, justo a la espera? Wes dice, incrédulo. "Eso es ¡imposible!"

Patty dice con una sonrisa, "Oh, por supuesto. Porque solo faltan treinta segundos para escribir, ¿Derecha?"

Bill y su equipo se dan cuenta de que su "tarea simple de treinta minutos" en realidad requiere siete transferencias (por ejemplo, equipo de servidor, equipo de redes, equipo de base de datos, equipo de virtualización y, por supuesto, Brent, el ingeniero "rockstar").

Suponiendo que todos los centros de trabajo estaban ocupados en un 90%, la figura nos muestra que el promedio el tiempo de espera en cada centro de trabajo es de nueve horas, y porque el trabajo tuvo que pasar por siete centros de trabajo, el tiempo total de espera es siete veces mayor que: sesenta y tres horas.

En otras palabras, el% total del *tiempo* de *valor agregado* (a veces conocido como tiempo de proceso) fue solo el 0.16% del tiempo total de entrega (treinta minutos divididos por sesenta y tres horas). Eso significa que para el 99.8% de nuestro tiempo de entrega total, el trabajo fue simplemente sentarse cola, esperando ser trabajado.

APÉNDICE 5 MITOS DE SEGURIDAD INDUSTRIAL

Décadas de investigación en sistemas complejos muestran que las contramedidas se basan en Varios mitos. En "Algunos mitos sobre la seguridad industrial", por Denis Besnard y Erik Hollnagel, se resumen así:

Mito 1: "El error humano es la mayor causa de accidentes e incidentes".

Mito 2: “Los sistemas estarán seguros si las personas cumplen con los procedimientos que han sido dado."

Mito 3: “La seguridad puede mejorarse mediante barreras y protección; más capas de la protección da como resultado una mayor seguridad ".

Mito 4: “El análisis de accidentes puede identificar la causa raíz (la 'verdad') de por qué el accidente pasó."

Mito 5: "La investigación de accidentes es la identificación lógica y racional de las causas basado en hechos ".

Mito 6: "La seguridad siempre tiene la máxima prioridad y nunca se verá comprometida".

Las diferencias entre lo que es mito y lo que es verdadero se muestran a continuación:

Página 364

Tabla 5: *Dos historias*

APÉNDICE 6 EL CABLE DE TOYOTA ANDON

Muchos preguntan cómo se puede completar cualquier trabajo si el cable de Andon se tira de más de cinco mil veces por día? Para ser precisos, no todos los tirones del cable Andon se detienen Toda la línea de montaje. Más bien, cuando se tira del cordón de Andon, el líder del equipo supervisar el centro de trabajo especificado tiene cincuenta segundos para resolver el problema. Si el el problema no se ha resuelto para el momento en que transcurren los cincuenta segundos, parcialmente el vehículo ensamblado cruzará una línea dibujada físicamente en el piso, y la línea de ensamblaje Será detenido.

Figura 48: *El cable de Toyota Andon*

APÉNDICE 7 SOFTWARE COTS

Actualmente, para obtener software COTS (comercial estándar) complejo (p. Ej.,
SAP, IBM WebSphere, Oracle WebLogic) en el control de versiones, es posible que tengamos que
elimine el uso de herramientas gráficas de instalación de proveedores de apuntar y hacer clic. Para hacer eso, nosotros
necesitamos descubrir qué está haciendo el instalador del proveedor, y es posible que necesitemos hacer una instalación
en una imagen de servidor limpia, diferencie el sistema de archivos y coloque los archivos agregados en la versión
controlar. Los archivos que no varían según el entorno se colocan en un solo lugar ("instalación base"),
mientras que los archivos específicos del entorno se colocan en su propio directorio ("prueba" o
"producción"). Al hacer esto, las operaciones de instalación de software se convierten simplemente en una versión
operación de control, que permite una mejor visibilidad, repetibilidad y velocidad.

Es posible que también tengamos que transformar los ajustes de configuración de cualquier aplicación para que sean
en control de versiones. Por ejemplo, podemos transformar configuraciones de aplicaciones que son
almacenado en una base de datos en archivos XML y viceversa.

APÉNDICE 8 REUNIONES POSTERIORES A MORTEM

A continuación se muestra una agenda de muestra de la reunión post mortem:

> El líder de la reunión o el facilitador harán una declaración inicial para reforzar
> que esta reunión es una autopsia irreprochable y que no nos centraremos en el pasado
> eventos o especular sobre "would haves" o "could haves". Los facilitadores pueden leer el

"Directiva principal retrospectiva" del sitio web Retrospective.com.

Además, el facilitador recordará a todos que cualquier contramedida debe
ser asignado a alguien, y si la acción correctiva no garantiza ser un máximo
prioridad cuando termina la reunión, entonces no es una acción correctiva. (Esto es para
evitar que la reunión genere una lista de buenas ideas que nunca
implementado.)

Aquellos en la reunión llegarán a un acuerdo sobre el cronograma completo del
incidente, incluido cuándo y quién detectó el problema, cómo se descubrió (por ejemplo,
monitoreo automatizado, detección manual, el cliente nos notificó), cuando el servicio fue
restaurado satisfactoriamente, y así sucesivamente. También integraremos en la línea de tiempo todo
Comunicaciones externas durante el incidente.

Cuando usamos la palabra "línea de tiempo", puede evocar la imagen de un conjunto lineal de pasos de
cómo obtuvimos una comprensión del problema y finalmente lo solucionamos. En realidad,
especialmente en sistemas complejos, probablemente habrá muchos eventos que contribuyeron a
el accidente, y se habrán tomado muchas rutas y acciones de solución de problemas en
un esfuerzo por arreglarlo En esta actividad, buscamos hacer una crónica de todos estos eventos y el
perspectivas de los actores y establecer hipótesis sobre causa y efecto
donde sea posible.

Page 366

El equipo creará una lista de todos los factores que contribuyeron al incidente, tanto
humano y técnico Luego pueden clasificarlos en categorías, como 'diseño
decisión, "remediación", "descubrir que había un problema", etc. El equipo
utilizará técnicas como la lluvia de ideas y los "infinitos" para profundizar
factores contribuyentes que consideran particularmente importantes para descubrir niveles más profundos de
factores contribuyentes. Todas las perspectivas deben ser incluidas y respetadas, nadie
debe permitirse discutir o negar la realidad de un factor contribuyente
alguien más ha identificado. Es importante que el facilitador post mortem
asegurarse de que se dedica suficiente tiempo a esta actividad y que el equipo no lo intenta
y participar en un comportamiento convergente, como tratar de identificar una o más 'raíces
causas.

Aquellos en la reunión llegarán a un acuerdo sobre la lista de acciones correctivas que
serán las principales prioridades después de la reunión. El ensamblaje de esta lista requerirá
lluvia de ideas y elegir las mejores acciones potenciales para prevenir el problema

de ocurrir o permitir una detección o recuperación más rápida. Otras formas de mejorar el También se pueden incluir sistemas.

Nuestro objetivo es identificar el menor número de pasos incrementales para lograr el resultados deseados, a diferencia de los cambios de "big bang", que no solo demoran más en implementar, pero retrasar las mejoras que necesitamos.

También generaremos una lista separada de ideas de menor prioridad y asignaremos un propietario. Si problemas similares ocurren en el futuro, estas ideas pueden servir como base para elaboración de futuras contramedidas.

Los asistentes a la reunión llegarán a un acuerdo sobre las métricas del incidente y sus Impacto organizacional. Por ejemplo, podemos elegir medir nuestros incidentes por las siguientes métricas:

- **Gravedad del evento:** ¿Cuán grave fue este problema? Esto se relaciona directamente con el impacto en el servicio y nuestros clientes.
- **Tiempo de inactividad total:** ¿durante cuánto tiempo los clientes no pudieron usar el servicio para cualquier grado?
- **Tiempo para detectar:** ¿Cuánto tiempo nos llevó a nosotros oa nuestros sistemas saber allí? fue un problema?
- **Tiempo para resolver:** ¿Cuánto tiempo tardó después de que supimos que había un problema? para que podamos restaurar el servicio?

Bethany Macri de Etsy observó: "La falta de culpa en una autopsia no significa que nadie se hace responsable Significa que queremos saber cuál es el las circunstancias permitieron que la persona que realizara el cambio o que introdujera el

Página 367

problema para hacer esto. ¿Cuál era el entorno más grande ...? La idea es que al eliminar culpa, eliminas el miedo, y al eliminar el miedo, obtienes honestidad ".

APÉNDICE 9 EL EJÉRCITO SIMIANO

Después de la interrupción de AWS EAST 2011, Netflix tuvo numerosas discusiones sobre diseñando sus sistemas para enfrentar automáticamente las fallas. Estas discusiones tienen evolucionó a un servicio llamado "Chaos Monkey".

Desde entonces, Chaos Monkey se ha convertido en una familia completa de herramientas, conocidas internamente. como el "Ejército Simian de Netflix", para simular niveles cada vez más catastróficos de fallas:

Chaos Gorilla: simula la falla de toda una zona de disponibilidad de AWS

Chaos Kong: simula fallas de regiones enteras de AWS, como Norteamérica o Europa

Otro miembro del Ejército Simian ahora incluye:

Mono de latencia: induce retrasos artificiales o tiempo de inactividad en su cliente RESTful capa de comunicación del servidor para simular la degradación del servicio y garantizar que los servicios dependientes responden adecuadamente

Mono de conformidad: encuentra y cierra instancias de AWS que no se adhieren a mejores prácticas (por ejemplo, cuando las instancias no pertenecen a un grupo de escalado automático o cuando no hay una dirección de correo electrónico de ingeniero de escalamiento en el catálogo de servicios)

Doctor Monkey: aprovecha las comprobaciones de estado que se ejecutan en cada instancia y encuentra instancias poco saludables y las apaga proactivamente si los propietarios no arreglan la raíz causa a tiempo

Janitor Monkey: garantiza que su entorno de nube se ejecute sin desorden y desperdicio; busca recursos no utilizados y los elimina

Security Monkey: una extensión de Conformity Monkey; encuentra y termina instancias con violaciones de seguridad o vulnerabilidades, como configuración incorrecta Grupos de seguridad de AWS

APÉNDICE 10 TIEMPO TRANSPARENTE

Lenny Rachitsky escribió sobre los beneficios de lo que llamó "tiempo de actividad transparente":

1. Sus costos de soporte disminuyen ya que sus usuarios pueden autoidentificarse en todo el sistema problemas sin llamar o enviar un correo electrónico a su departamento de soporte. Los usuarios no Ya no tienen que adivinar si sus problemas son locales o globales, y pueden más rápidamente llegar a la raíz del problema antes de quejarse.

2. Puede comunicarse mejor con sus usuarios durante los eventos de tiempo de inactividad, aprovechando la naturaleza de transmisión de Internet versus el uno a uno

naturaleza del correo electrónico y el teléfono. Pasas menos tiempo comunicando lo mismo una y otra vez y más tiempo resolviendo el problema.

3. Usted crea un lugar único y obvio para que sus usuarios vengan cuando estén experimentando tiempo de inactividad. Ahorras el tiempo que tus usuarios pasan actualmente buscando foros, Twitter o tu blog.

4. La confianza es la piedra angular de cualquier adopción exitosa de SaaS. Sus clientes son apostando su negocio y sus medios de vida en su servicio o plataforma. Ambos Los clientes actuales y potenciales requieren confianza en su servicio. Ambos necesitan sé que no se quedarán en la oscuridad, solos y desinformados, cuando te encuentres con problema. La visión en tiempo real de eventos inesperados es la mejor manera de construir esta confianza. Mantenerlos en la oscuridad y solos ya no es una opción.

5. Es solo cuestión de tiempo antes de que cada proveedor de SaaS serio ofrezca un tablero de instrumentos de salud pública. Tus usuarios lo exigirán.

Recursos adicionales

Muchos de los problemas comunes que enfrentan las organizaciones de TI se discuten en la primera mitad del libro *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win* de Gene Kim, Kevin Behr y George Spafford.

Este video muestra un discurso que Paul O'Neill pronunció sobre su mandato como CEO de Alcoa, incluyendo la investigación en la que participó después de que un trabajador adolescente fue asesinado en una de las casas de Alcoa plantas: https://www.youtube.com/watch?v=tC2ucDs_XJY .

Para obtener más información sobre la asignación de flujo de valor, consulte *Asignación de flujo de valor: cómo visualizar Trabajar y alinear el liderazgo para la transformación organizacional* por Karen Martin y Mike Osterling.

Para obtener más información sobre los ORM, visite Stack Overflow: http://stackoverflow.com/questions/1279613/what-is-an-orm-and-where-can-i-learn-más sobre eso .

Una excelente introducción a muchos rituales de desarrollo ágil y cómo usarlos en TI El trabajo de operaciones se puede encontrar en una serie de publicaciones escritas en el blog Agile Admin: http://theagileadmin.com/2011/02/21/scrum-for-operations-what-is-scrum/ .

Para obtener más información sobre la arquitectura para construcciones rápidas, consulte Daniel Worthington-Bodart's publicación de blog "Tiempos de compilación rápidos y locos (o cuando 10 segundos comienzan a ponerte nervioso)": http://dan.bodar.com/2012/02/28/crazy-fast-build-times-or-when-10-seconds-starts-a-make-you-nervioso / .

Para obtener más detalles sobre las pruebas de rendimiento en Facebook, junto con algunos detalles información sobre el proceso de lanzamiento de Facebook, consulte la presentación de Chuck Rossi "The Proceso de lanzamiento de Facebook ": http://www.infoq.com/presentations/Facebook-Release-Proceso .

Se pueden encontrar muchas más variantes del lanzamiento oscuro en el capítulo 8 de *La práctica de Administración del sistema en la nube: diseño y operación de grandes sistemas distribuidos, Volumen 2* de Thomas A. Limoncelli, Strata R. Chalup y Christina J. Hogan.

Aquí hay una excelente discusión técnica sobre las funciones de alternancia: http://martinfowler.com/articles/feature-toggles.html .

Las versiones se analizan con más detalle en *The Practice of Cloud System Administration: Diseño y operación de grandes sistemas distribuidos, Volumen 2* por Thomas A. Limoncelli, Strata R. Chalup y Christina J. Hogan; *Entrega continua: confiable Lanzamientos de software a través de la automatización* de *compilación, prueba e implementación* de Jez Humble

y David Farley; y *liberarlo! Diseñe e implemente software listo para producción* por Michael T. Nygard.

Una descripción del patrón del interruptor automático se puede encontrar aquí: http://martinfowler.com/bliki/CircuitBreaker.html .

Para obtener más información sobre el costo de la demora, consulte *Los principios del flujo de desarrollo de productos: segundo Desarrollo de productos Lean de generación* por Donald G. Reinertsen.

Se puede encontrar una discusión adicional sobre cómo adelantarse a las fallas del servicio Amazon S3 aquí: https://qconsf.com/sf2010/dl/qcon-sanfran-2009 / diapositivas / JasonMcHugh_AmazonS3ArchitectingForResiliencyInTheFaceOfFailures.pdf .

Para obtener una guía excelente sobre cómo realizar investigaciones de usuarios, consulte *Lean UX: Aplicación de Lean Principios para mejorar la experiencia del usuario* por Jeff Gothelf y Josh Seiden.

¿Qué prueba ganó? es un sitio que muestra cientos de pruebas A / B de la vida real y pregunta espectador para adivinar qué variante funcionó mejor, reforzando la clave que a menos que en realidad prueba, simplemente estamos adivinando. Visítala aquí:http://whichtestwon.com/ .

Se puede encontrar una lista de patrones arquitectónicos en *Release It! Diseñar e implementar Software listo para producción* de Michael T. Nygard.

Puede encontrar un ejemplo de las notas de la reunión post-mortem de Chef publicadas aquí: https://www.chef.io/blog/2014/08/14/cookbook-dependency-api-postmortem/ . UN El video de la reunión se puede encontrar aquí: https://www.youtube.com/watch?v = Rmi1Tn5oWfI .

Puede encontrar un cronograma actual de los próximos DevOpsDays en el sitio web de DevOpsDays: http://www.devopsdays.org/ . Las instrucciones para organizar un nuevo DevOpsDays pueden ser que se encuentra en la página de la Guía de organización de DevOpsDay: http://www.devopsdays.org/pages/organizing/ .

Puede encontrar más información sobre el uso de herramientas para administrar secretos en la publicación de Noah Kantrowitz "Secretos Management and Chef "en su blog: https://coderanger.net/chef-secrets/ .

James Wickett y Gareth Rushgrove han puesto todos sus ejemplos de tuberías seguras en el sitio web de GitHub: https://github.com/secure-pipeline .

El sitio web de la Base de datos de vulnerabilidad nacional y las fuentes de datos XML se pueden encontrar en: https://nvd.nist.gov/ .

Un escenario concreto que involucra la integración entre Puppet y ThoughtWorks 'Go y Mingle (una aplicación de gestión de proyectos) se puede encontrar en una publicación de blog de Puppet Labs por Andrew Cunningham y Andrew Myers y editado por Jez Humble: https://puppetlabs.com/blog/a-deployment-pipeline-for-infrastructure .

La preparación y aprobación de auditorías de cumplimiento se explora más a fondo en 2015 de Jason Chan presentación "SEC310: División del control de cumplimiento y seguridad: mantenimiento

Desarrolladores y auditores felices en la nube ": https://www.youtube.com/watch?v = Io00_K4v12Y & feature = youtu.be .

La historia de cómo Jez Humble transformó los ajustes de configuración de la aplicación y David Farley para Oracle WebLogic se describió en el libro *Entrega continua: Versiones de software confiables a través de la automatización de compilación, prueba e implementación* . Mirco Hering describió un enfoque más genérico para este proceso aquí: http://notafactoryanymore.com/2015/10/19/devops-for-systems-of-record-a-new-hope-preview-of-does-talk / .

Una lista de muestra de los requisitos operativos de DevOps se puede encontrar aquí: http://blog.devopsguys.com/2013/12/19/the-top-ten-devops-operational-requisitos / .

Notas finales

INTRODUCCIÓN

Antes de la revolución ... Eliyahu M. Goldratt, más *allá de la meta: Eliyahu Goldratt habla sobre The Theory of Restraints (Your Coach in a Box)* (Prince Frederick, Maryland: Gildan Media, 2005), Audiolibro.

Pon aún más ... Jeff Immelt, “CEO de GE, Jeff Immelt: finalmente terminemos el debate Si estamos en una burbuja tecnológica ", *Business Insider* , 9 de diciembre de 2015, http://www.businessinsider.com/ceo-of-ge-lets-finally-end-the-debate-over-whether-we-are-in-a-tech-bubble-2015-12 .

O como Jeffrey ... "Weekly Top 10: Your DevOps Flavor", *Electric Cloud* , 1 de abril de 2016, http://electric-cloud.com/blog/2016/04/weekly-top-10-devops-flavor/ .

Dr. Eliyahu M. Goldratt ... Goldratt, más *allá de la meta.*

Como Christopher Little ... Christopher Little, correspondencia personal con Gene Kim, 2010.

Como Steven J. Spear ... Steven J. Spear, *The High-Velocity Edge: How Market Leaders Aproveche la excelencia operativa para vencer a la competencia* (Nueva York, NY: McGraw Hill Educación), edición Kindle, cap. 3)

En 2013, el ... Chris Skinner, "Los bancos tienen tiendas de desarrollo más grandes que Microsoft", Chris

Skinner's Blog, consultado el 28 de julio de 2016, http://thefinanser.com/2011/09/banks-have-mayor-desarrollo-tiendas-que-microsoft.html / .

Los proyectos son típicamente ... Nico Stehr y Reiner Grundmann, *Conocimiento: Conceptos críticos, Volumen 3* (Londres: Routledge, 2005), 139.

Dr. Vernon Richardson ... A. Masli, V. Richardson, M. Widenmier y R. Zmud, "Senior Responsabilidades de la gerencia de TI del ejecutivo: serias deficiencias de TI y CEO-CFO Volumen de negocios ", *MIS Quaterly* (publicado electrónicamente el 21 de junio de 2016).

Considere lo siguiente ... "IDC prevé que el gasto mundial en TI crecerá un 6% en 2012, A pesar de la incertidumbre económica ", *Business Wire* , 10 de septiembre de 2012, http://www.businesswire.com/news/home/20120910005280/en/IDC-Forecasts-Worldwide-Spending-Grow-6-2012 .

La primera sorpresa ... Nigel Kersten, IT Revolution y PwC, *Informe de estado de DevOps 2015* (Portland, OR: Puppet Labs, 2015), https://puppet.com/resources/white-paper/2015-state-of-devops-report? _ga = 1.6612658.168869.1464412647 & link = blog .

Esto se destaca ... Frederick P. Brooks, Jr., *The Mythical Man-Month: Ensayos sobre Ingeniería de Software, Edición Aniversario* (Upper Saddle River, NJ: Addison-Wesley, 1995).

Como Randy Shoup ... Gene Kim, Gary Gruver, Randy Shoup y Andrew Phillips, "Explorando el territorio desconocido de microservicios ", XebiaLabs.com, seminario web, 20 de febrero de 2015,

Page 373

https://xebialabs.com/community/webinars/exploring-the-uncharted-territory-of-microservicios / .

The State 2015 ... Kersten, IT Revolution y PwC, *2015 State of DevOps Report* .

Otro más extremo ... "Velocity 2011: Jon Jenkins, 'Velocity Culture'", video de YouTube, 15:13, publicado por O'Reilly, 20 de junio de 2011, https://www.youtube.com/watch? v = dxk8b9rSKOo ; "Transformando el desarrollo de software", video de YouTube, 40 : 57, publicado por Amazon Web Service, 10 de abril de 2015, https://www.youtube.com/watch? v = YCrhemssYuI & feature = youtu.be .

Más tarde en su ... Eliyahu M. Goldratt, *Más allá de la meta* .

Al igual que con The ... JGFLL, revisión de *The Phoenix Project: A Novel About IT, DevOps y Ayudando a que su negocio gane* , por Gene Kim, Kevin Behr y George Spafford, Amazon.com revisión, 4 de marzo de 2013, http://www.amazon.com/review/R1KSSPTEGLWJ23 ; Mark L Townsend, revisión del *Proyecto Phoenix: una novela sobre TI, DevOps y cómo ayudar a Business Win* , por Gene Kim, Kevin Behr y George Spafford, revisión de Amazon.com, marzo 2, 2013, http://uedata.amazon.com/gp/customer-opiniones / R1097DFODM12VD / ref = cm_cr_getr_d_rvw_ttl? ie = UTF8 & ASIN = B00VATFAMI ; Scott Van Den Elzen, revisión de *The Phoenix Project: A Novel About IT, DevOps y Ayudando a que su negocio gane* , por Gene Kim, Kevin Behr y George Spafford, Amazon.com

revisión, 13 de marzo de 2013, http://uedata.amazon.com/gp/customer-opiniones / R2K95XEH5OL3Q5 / ref = cm_cr_getr_d_rvw_ttl? ie = UTF8 & ASIN = B00VATFAMI .

PARTE I INTRODUCCIÓN

Un principio clave ... Kent Beck, et al., "Doce principios del software ágil" AgileManifesto.org, 2001, http://agilemanifesto.org/principles.html .

Llegó a la conclusión de que ... Mike Rother, *Toyota Kata: Gestión de personas para mejorar, Adaptabilidad y resultados superiores* (Nueva York: McGraw Hill, 2010), edición Kindle, parte III.

CAPÍTULO 1

Karen Martin y ... Karen Martin y Mike Osterling, *Value Stream Mapping: How to Visualice el trabajo y alinee el liderazgo para la transformación organizacional* (Nueva York: McGraw Hill, 2013), edición Kindle, cap.1.

En este libro ... Ibid., Cap. 3)

Karen Martin y ... Ibid.

CAPITULO 2

Los estudios han demostrado ... Joshua S. Rubinstein, David E. Meyer y Jeffrey E. Evans, "Control ejecutivo de procesos cognitivos en el cambio de tareas", *Journal of Experimental Psicología: percepción y rendimiento humanos* 27, no. 4 (2001): 763-797, doi:

Página 374

10.1037 // 0096-1523.27.4.763, http://www.umich.edu/~bcalab/documents/RubinsteinMeyerEvans2001.pdf .

Dominica DeGrandis, uno ... "DOES15 — Dominica DeGrandis — La forma de la incertidumbre" Video de YouTube, 22:54, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015, https://www.youtube.com/watch?v=Gp05i0d34gg .

Taiichi Ohno comparó ... Sami Bahri, "Pocos pacientes en proceso y menos seguridad Planificación; Los suministros entrantes son secundarios ", The Deming Institute Blog, 22 de agosto, 2013, https://blog.deming.org/2013/08/fewer-patients-in-process-and-less-safety-programación-suministros-entrantes-son-secundarios / .

En otras palabras ... Reunión entre David J. Andersen y el equipo de Motorola con Daniel S. Vacanti, 24 de febrero de 2004; historia contada en USC CSSE Research Review con Barry Boehm en marzo de 2004.

Las diferencias dramáticas ... James P. Womack y Daniel T. Jones, *Lean Thinking: Banish Desperdicia y crea riqueza en tu corporación* (Nueva York: Free Press, 2010), Kindle edición, cap. 1)

Hay muchos ... Eric Ries, "Trabaja en pequeños lotes", StartupLessonsLearned.com, 20 de febrero de 2009, http://www.startuplessonslearned.com/2009/02/work-in-small-batches.html .

En Beyond the ... Goldratt, *Beyond the Goal* .

Como solución ... Eliyahu M. Goldratt, *El objetivo: un proceso de mejora continua* (excelente Barrington, MA: North River Press, 2014), edición Kindle, "Cinco pasos de enfoque".

Shigeo Shingo, uno ... Shigeo Shingo, *un estudio del sistema de producción de Toyota: de un Industrial Engineering Viewpoint* (Londres: Productivity Press, 1989); "Los 7 desechos (Siete formas de Muda) ", BeyondLean.com, consultado el 28 de julio de 2016, http://www.beyondlean.com/7-wastes.html .

En el libro ... Mary Poppendieck y Tom Poppendieck, *Implementando Lean Software: Del concepto al efectivo* (Upper Saddle River, NJ: Addison-Wesley, 2007), 74.

Las siguientes categorías ... Adaptado de Damon Edwards, "DevOps Kaizen: Find and Fix Qué está realmente detrás de sus problemas ", Slideshare.net, publicado por dev2ops, 4 de mayo de 2015, http://www.slideshare.net/dev2ops/dev-ops-kaizen-damon-edwards .

CAPÍTULO 3

Dr. Charles Perrow ... Charles Perrow, *Accidentes normales: viviendo con alto riesgo Tecnologías* (Princeton, NJ: Princeton University Press, 1999).

Dr. Sidney Dekker ... Dr. Sidney Dekker, *La guía de campo para comprender el error humano* (Universidad de Lund, Suecia: Ashgate, 2006).

Después de descifrar ... Spear, *The High-Velocity Edge* , cap. 8)

El Dr. Spear extendió ... Ibid.

Dr. Peter Senge ... Peter M. Senge, *La quinta disciplina: el arte y la práctica del aprendizaje Organización* (Nueva York: Doubleday, 2006), edición Kindle, cap. 5)

En un bien documentado ... "NUMMI", *This American Life* , 26 de marzo de 2010, http://www.thisamericanlife.org/radio-archives/episode/403/transcript .

Como Elisabeth Hendrickson ... "DOES15 - Elisabeth Hendrickson - Todo se trata de comentarios" Video de YouTube, 34:47, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015, https://www.youtube.com/watch?v=r2BFTXBundQ .

"Al hacerlo ... Spear, *The High-Velocity Edge* , cap. 1)
Como el Dr. Spear ... Ibid., Cap. 4)

Ejemplos de ineficaces ... Jez Humble, Joanne Molesky y Barry O'Reilly, *Lean Empresa: cómo las organizaciones de alto rendimiento innovan a escala* (Sebastopol, CA: O'Reilly Media, 2015), edición Kindle, Parte IV.

En la década de 1700 ... Dr. Thomas Sowell, *Conocimiento y decisiones* (Nueva York: Basic Books, 1980), 222.

Como Gary Gruver ... Gary Gruver, correspondencia personal con Gene Kim, 2014.

CAPÍTULO 4

Por ejemplo, en ... Paul Adler, "Tiempo y movimiento recuperados", *Harvard Business Review* , Enero-febrero 1993, https://hbr.org/1993/01/time-and-motion-regained .

El "nombre, culpa ... Dekker, *La guía de campo para comprender el error humano,* cap. 1)

Dr. Sidney Dekker ... "Cultura justa: equilibrio entre seguridad y responsabilidad", Universidad de Lund, Sitio web de Factores humanos y seguridad del sistema, 6 de noviembre de 2015, http://www.humanfactors.lth.se/sidney-dekker/books/just-culture/ .

Observó que ... Ron Westrum, "El estudio del flujo de información: un viaje personal" *Actas de Safety Science* 67 (agosto de 2014): 58-63, https://www.researchgate.net/publication/261186680_ The_study_of_information_flow_A_personal_journey .

Así como el Dr. Westrum ... Nicole Forsgren Velásquez, Gene Kim, Nigel Kersten y Jez Humilde, *Informe del Estado de DevOps 2014* (Portland, OR: Puppet Labs, IT Revolution Press, y ThoughtWorks, 2014), http://puppetlabs.com/2014-devops-report .

Como Bethany Macri ... Bethany Macri, "Morgue: ayudando a comprender mejor los eventos Construyendo una herramienta Post Mortem - Bethany Macri ", video de Vimeo, 33:34, publicado por info@devopsdays.org, 18 de octubre de 2013, http://vimeo.com/77206751 .

El Dr. Spear observa ... Spear, *The High-Velocity Edge* , cap. 1)

En The Fifth ... Senge, *The Fifth Discipline, cap. 1* .

Mike Rother observó ... Mike Rother, *Toyota Kata* , 12.

Es por eso ... Mike Orzen, correspondencia personal con Gene Kim, 2012.

Considere lo siguiente ... "Paul O'Neill" , *Forbes* , 11 de octubre de 2001, http://www.forbes.com/2001/10/16/poneill.html .

En 1987, Alcoa ... Spear, *The High-Velocity Edge* , cap. 4)

Como el Dr. Spear ... Ibid.

Un ejemplo notable ... Ibid., Cap. 5)

Este proceso de ... Nassim Nicholas Taleb, *Antifragile: Cosas que se obtienen del desorden* (Incerto), (Nueva York: Random House, 2012).

Según Womack ... Jim Womack, Gemba Walks (Cambridge, MA: Lean Enterprise Institute, 2011), edición Kindle, ubicación 4113.

Mike Rother formalizó ... Rother, *Toyota Kata* , Parte IV.

Mike Rother observa ... Ibid., Conclusión.

CAPÍTULO 5

Por lo tanto, debemos ... Michael Rembetsy y Patrick McDonnell, "Implementación continua Cultura [en Etsy] ", Slideshare.net, 4 de octubre de 2012, publicado por Patrick McDonnel.bl, http://www.slideshare.net/mcdonnps/continuously-deploying-culture-scaling-culture-at-etsy-14588485 .

En 2015, Nordstrom ... "Nordstrom, Inc.", perfil de la compañía en Vault.com, http://www.vault.com/company-profiles/retail/nordstrom,-inc/company-overview.aspx .

El escenario para ... Courtney Kissler, "DOES14 - Courtney Kissler - Nordstrom - Transformando a una cultura de mejora continua ", video de YouTube, 29:59, publicado por DevOps Enterprise Summit 2014, 29 de octubre de 2014, https://www.youtube.com/watch?v = 0ZAcsrZBSlo .

Estas organizaciones fueron ... Tom Gardner, "Barnes & Noble, Blockbuster, Borders: The El asesino B está muriendo ", *The Motley Fool* , 21 de julio de 2010, http://www.fool.com/investing/general/2010/07/21/barnes-noble-blockbuster-borders-the-killer-bs-are.aspx .

Como Kissler describió ... Kissler, "DOES14 - Courtney Kissler - Nordstrom".

Como Kissler dijo ... Ibid; Modificaciones a la cita hechas por Courtney Kissler a través de personal correspondencia con Gene Kim, 2016.

Como Kissler declaró ... Ibid; Modificaciones a la cita hechas por Courtney Kissler a través de personal correspondencia con Gene Kim, 2016.

En 2015, Kissler ... Ibid.

Ella continuó: "Esto ... Ibid.

Kissler concluyó: "De ... Ibid.

Un ejemplo de ... Ernest Mueller, "La adopción de la nube impulsada por el modelo de negocio: lo que está haciendo NI en la nube ", Slideshare.net, 28 de junio de 2011, publicado por Ernest Mueller, http://www.slideshare.net/mxyzplk/business-model-driven-cloud-adoption-what-ni-is-haciendo en la nube .

Aunque muchos creen ... Cálculo no publicado por Gene Kim después del DevOps 2014 Cumbre empresarial.

De hecho, uno de ... Kersten, IT Revolution y PwC, *Informe del estado de DevOps 2015* .

CSG (2013): En ... Prugh, "DOES14: Scott Prugh, CSG - DevOps y Lean in Legacy Entornos ", Slideshare.net, 14 de noviembre de 2014, publicado por DevOps Enterprise Summit, http://www.slideshare.net/DevOpsEnterpriseSummit/scott-prugh .

Etsy (2009): En ... Rembetsy y McDonnell, "Implementación continua de la cultura [en Etsy]".

La investigación de Gartner ... Bernard Golden, "Qué significa el modelo bimodal de TI de Gartner Enterprise CIOs ", *CIO Magazine* , 27 de enero de 2015, http://www.cio.com/article/2875803/cio-role/what-gartner-s-bimodal-it-model-means-to-enterprise-cios.html .

Sistemas de registro ... Ibid.

Sistemas de compromiso ... Ibid.

Los datos de ... Kersten, IT Revolution, y PwC, *2015 State of DevOps Report* .

Scott Prugh, VP ... Scott Prugh, correspondencia personal con Gene Kim, 2014.

Geoffrey A. Moore ... Geoffrey A. Moore y Regis McKenna, *Cruzando el abismo: Comercialización y venta de productos de alta tecnología a clientes convencionales* (Nueva York: HarperCollins, 2009), 11.

Big Bang, de arriba hacia abajo ... Linda Tucci, “Cuatro pilares del ágil 'Big Bang' de PayPal Transformación ", *TechTarget* , agosto de 2014, http://searchcio.techtarget.com/feature/Four-pillars-of-PayPals-big-bang-Agile-la transformación .

La siguiente lista ... "Creación de organizaciones de alta velocidad", descripción, por supuesto, de Roberto Fernández y Steve Spear, sitio web de MIT Sloan Executive Education, accedieron 30 de mayo de 2016, http://executivo.mit.edu/openenrollment/program/organizational-organizaciones de desarrollo de alta velocidad .

Pero como Ron van Kemenade ... Ron Van Kemande, "Nada supera el talento de ingeniería: el Transformación ágil en ING ", presentación en DevOps Enterprise Summit, Londres, Reino Unido, del 30 de junio al 1 de julio de 2016.

Peter Drucker, un ... Leigh Buchanan, "La sabiduría de Peter Drucker de la A a la Z", *Inc.* , 19 de noviembre de 2009, http://www.inc.com/articles/2009/11/drucker.html .

CAPÍTULO 6

Con los años ... Kissler, "DOES14 - Courtney Kissler - Nordstrom".

Kissler explicó: ... Ross Clanton y Michael Ducy, entrevista de Courtney Kissler y Jason Josephy, "Mejora continua en Nordstrom", *The Goat Farm* , podcast audio, 25 de junio de 2015, http://goatcan.do/2015/06/25/the-goat-farm-episode-7-continuous-mejora-en-nordstrom / .

Ella dijo con orgullo ... Ibid.

Los ejecutivos de tecnología o ... Brian Maskell, "¿Qué hace este tipo? Rol del flujo de valor

Gerente ", *Maskell* , 3 de julio de 2015, http://blog.maskell.com/?
p = 2106http: //www.lean.org/common/display/? o = 221 .

Página 378

Damon Edwards observó ... Damon Edwards, "DevOps Kaizen: Encuentra y repara lo que es Realmente detrás de sus problemas ", Slideshare.net, publicado por dev2ops, 4 de mayo de 2015, http://www.slideshare.net/dev2ops/dev-ops-kaizen-damon-edwards .

En su libro ... Vijay Govindarajan y Chris Trimble, *El otro lado de la innovación: Resolviendo el desafío de ejecución* (Boston, MA: Harvard Business Review, 2010) Kindle edición.

Basado en su ... Ibid., Parte I.

Después de la muerte cercana ... Marty Cagan, *inspirado: cómo crear productos que los clientes adoren* (Saratoga, CA: SVPG Press, 2008), 12.

Cagan señala que ... Ibid.

Seis meses después de ... Ashlee Vance, "LinkedIn: una historia sobre posiblemente de Silicon Valley Necesidad poco saludable de velocidad " , *Bloomberg* , 30 de abril de 2013, http://www.bloomberg.com/bw/articles/2013-04-29/linkedin-a-story-about-silicon-valles-posiblemente-insalubres-necesidad-de-velocidad .

LinkedIn fue creado ... "LinkedIn comenzó en 2003 - LinkedIn - Una breve historia" Slideshare.net, publicado por Josh Clemm, 9 de noviembre de 2015, http://www.slideshare.net/joshclemm/how-linkedin-scaled-a-brief-history/3-LinkedIn_started_back_in_2003 .

Un año después ... Jonas Klit Nielsen, "8 años con LinkedIn: mirando el crecimiento [Infografía], "MindJumpers.com, 10 de mayo de 2011, http://www.mindjumpers.com/blog/2011/05/linkedin-growth-infographic/ .

Para noviembre de 2015 ... "LinkedIn comenzó en 2003", Slideshare.net.

El problema era ... "De un monolito a microservicios + RESTO: La evolución de Arquitectura de LinkedIn ", Slideshare.net, publicado por Karan Parikh, 6 de noviembre, 2014, http://www.slideshare.net/parikhk/restli-and-deco .

Josh Clemm, un ... "LinkedIn comenzó en 2003", Slideshare.net.

En 2013, periodista ... Vance, "LinkedIn: una historia sobre", *Bloomberg* .

Scott lanzó Operation ... "Cómo estructuré los equipos de ingeniería en LinkedIn y AdMob para el éxito ", *First Round Review* , 2015, http://firstround.com/review/how-i-structured-ingenieros-equipos-en-linkedin-y-admob-for-success / .

Scott describió uno ... Ashlee Vance, "Dentro de la Operación Inversión, el Código Congelado que LinkedIn guardado ", *Bloomberg* , 11 de abril de 2013,

http://www.bloomberg.com/news/articles/2013-04-10/inside-operation-inversion-the-código-congelar-que-guardado-linkedin .

Sin embargo, Vance describió ... Vance, "LinkedIn: una historia sobre", *Bloomberg* .

Como Josh Clemm ... "LinkedIn comenzó en 2003", Slideshare.net.

Kevin Scott declaró ... "Cómo estructuré los equipos de ingeniería", *Revisión de la primera ronda* .

Como Christopher Little ... Christopher Little, correspondencia personal con Gene Kim, 2011.

Como Ryan Martens ... Ryan Martens, correspondencia personal con Gene Kim, 2013.

Página 379

CAPÍTULO 7

Observó: "Después de ... Dr. Melvin E. Conway," ¿Cómo inventan los comités? " MelConway.com, http://www.melconway.com/research/committees.html , anteriormente publicado en *Datamation* , abril de 1968.

Estas observaciones llevaron ... Ibid.

Eric S. Raymond, autor ... Eric S. Raymond, "Conway's Law", catb.org, consultado el 31 de mayo, 2016, http://catb.org/~esr/jargon/ .

El viaje de DevOps de Etsy ... Sarah Buhr, "Etsy cierra el 86 por ciento en el primer día de negociación" *Tech Crunch* , 16 de abril de 2015, http://techcrunch.com/2015/04/16/etsy-stock-surges-86-Porcentaje al cierre del primer día de negociación a 30 por acción .

Como Ross Snyder ... "Escalando Etsy: lo que salió mal, lo que salió bien", Slideshare.net, publicado por Ross Snyder, 5 de octubre de 2011, http://www.slideshare.net/beamrider9/scaling-etsy-qué-salió-mal-qué-salió-bien .

Como observó Snyder ... Ibid.

En otras palabras ... Sean Gallagher, "Cuando 'inteligente' va mal: cómo Etsy superó a los pobres Architectural Choices ", *Arstechnica* , 3 de octubre de 2011, http://arstechnica.com/business/2011/10/when-clever-goes-wrong-how-etsy-overcame-malas elecciones arquitectónicas .

Snyder explicó que ... "Scaling Etsy" Slideshare.net.

Etsy inicialmente tenía ... Ibid.

En la primavera ... Ibid.

Como lo describió Snyder ... Ross Snyder, "Surge 2011 — Scaling Etsy: What Wer Wrong, What Went Right ", video de YouTube, publicado por Surge Conference, 23 de diciembre de 2011, https://www.youtube.com/watch?v=eenrfm50mXw .

Como dijo Snyder ... Ibid.

Sprouter era uno ... "Despliegue continuo de la cultura: Escalando la cultura en Etsy - Velocidad Europa 2012 ", Slideshare.net, publicado por Patrick McDonnell, 4 de octubre de 2012, http://www.slideshare.net/mcdonnps/continuously-deploying-culture-scaling-culture-at-etsy-14588485 .

Están definidos ... "Creación de organizaciones de alta velocidad", descripción, por supuesto, por Roberto Fernández y Steven Spear.

Adrian Cockcroft comentó ... Adrian Cockcroft, correspondencia personal con Gene Kim, 2014.

In the Lean ... Spear, *The High-Velocity Edge* , cap. 8)

Como Mike Rother ... Rother, *Toyota Kata* , 250.

Reflexionando sobre lo compartido ... "DOES15 - Jody Mulkey - DevOps en la empresa: A Transformation Journey ", video de YouTube, 28:22, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015 https://www.youtube.com/watch?v=USYrDaPEFtM .

Él continuó: "El ... Ibid.

Page 380

Pedro Canahuati, su ... Pedro Canahuati, "Creciendo de Pocos a Muchos: Escalando la Organización de Operaciones en Facebook " , *InfoQ* , 16 de diciembre de 2013, http://www.infoq.com/presentations/scaling-operations-facebook .

Cuando los departamentos se especializan en exceso ... Spear, *The High-Velocity Edge* , cap. 1)

Scott Prugh escribe ... Scott Prugh, "Entrega continua", Scaled Agile Framework, actualizado el 14 de febrero de 2013, http://www.scaledagileframework.com/continuous-delivery/ .

" Por entrenamiento cruzado ... Ibid.

"Los gerentes tradicionales ... ibid.

Además, como Prugh ... Ibid.

Cuando valoramos ... Dra. Carol Dweck, "Carol Dweck revisita la 'mentalidad de crecimiento'" *Semana de la Educación* , 22 de septiembre de 2015, http://www.edweek.org/ew/articles/2015/09/23/carol-dweck-revisits-the-growth-mindset.html .

Como Jason Cox ... Jason Cox, "Disney DevOps: To Infinity and Beyond", presentación en DevOps Enterprise Summit 2014, San Francisco, CA, octubre de 2014.

Como John Lauderbach ... John Lauderbach, conversación personal con Gene Kim, 2001.

Estas propiedades son ... Tony Mauro, "Adopción de microservicios en Netflix: lecciones para Diseño arquitectónico ", *NGINX* , 19 de febrero de 2015, https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/ .; Adam Wiggins, "La aplicación de los doce factores", 12Factor.net, 30 de enero de 2012,

http://12factor.net/ .

Randy Shoup, ex ... "Explorando el territorio desconocido de microservicios", YouTube video, 56:50, publicado por XebiaLabs, Inc., 20 de febrero de 2015, https://www.youtube.com/watch?v=MRa21icSIQk .

Como parte de ... Humble, O'Reilly y Molesky, *Lean Enterprise* , Parte III.

En Netflix ... Reed Hastings, "Cultura de Netflix: libertad y responsabilidad" Slideshare.net, 1 de agosto de 2009, http://www.slideshare.net/reed2001/culture-1798664 .

CTO de Amazon Werner ... Larry Dignan, "Little Things Add Up" , *línea de base* , 19 de octubre de 2005, http://www.baselinemag.com/c/a/Projects-Management/Profiles-Lessons-From-the-Líderes en la línea de base i500 / 3 .

El objetivo es ... Heather Mickman y Ross Clanton, "DOES15 - Heather Mickman & Ross Clanton - (Re) construyendo una cultura de ingeniería: DevOps at Target ", video de YouTube, 33:39, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015, https://www.youtube.com/watch?v=7s-VbB1fG5o .

Como describió Mickman ... Ibid.

En un intento ... Ibid.

Porque nuestro equipo ... Ibid.

En lo siguiente ... Ibid.

Estos cambios tienen ... Ibid.

---

381 de 1189.

La habilitación de API ... Ibid.

CAPÍTULO 8

En Big Fish ... "Big Fish celebra el 11º año consecutivo de crecimiento récord" BigFishGames.com, 28 de enero de 2014, http://pressroom.bigfishgames.com/2014-01-28-Big-Fish celebra el 11º año consecutivo de crecimiento récord .

Observó que ... Paul Farrall, correspondencia personal con Gene Kim, enero de 2015.

Farrall definió dos ... Ibid., 2014.

Concluye: "El ... Ibid.

Ernest Mueller observó ... Ernest Mueller, correspondencia personal con Gene Kim, 2014.

Como Damon Edwards ... Edwards, "DevOps Kaizen".

Dianne Marsh, Directora ... "Dianne Marsh 'presentando el cambio mientras preserva Engineering Velocity ", video de YouTube, 17:37, publicado por Flowcon, 11 de noviembre de 2014, https://www.youtube.com/watch?v=eW3ZxY67fnc .

Jason Cox dijo ... Jason Cox, "Disney DevOps".

En Etsy, esto ... "devopsdays Minneapolis 2015 - Katherine Daniels - DevOps: The Missing Piezas ", video de YouTube, 33:26, publicado por DevOps Minneapolis, 13 de julio de 2015, https://www.youtube.com/watch?v=LNJkVw93yTU .

Como Ernest Mueller ... Ernest Mueller, correspondencia personal con Gene Kim, 2015.

Scrum es un ágil ... Hirotaka Takeuchi e Ikujiro Nonaka, "Desarrollo de nuevos productos Game ", *Harvard Business Review* (enero de 1986): 137-146.

CAPÍTULO 9

En su presentación ... Em Campbell-Pretty, "DOES14 - Em Campbell-Pretty - Cómo un Business Exec Led Agile, Lead, CI / CD ", video de YouTube, 29:47, publicado por DevOps Cumbre Empresarial, 20 de abril de 2014, https://www.youtube.com/watch?v=-4pIMMTbtwE .

Campbell-Pretty se convirtió en ... Ibid.

Crearon un ... Ibid.

Campbell-Pretty observó ... Ibid.

Campbell-Pretty describió ... Ibid.

La primera versión ... "Historial de control de versiones", PlasticSCM.com, consultado el 31 de mayo de 2016, https://www.plasticscm.com/version-control-history.html .

Un control de versiones ... Jennifer Davis y Katherine Daniels, DevOps efectivos: Construyendo un Cultura de colaboración, afinidad y herramientas a escala (Sebastopol, CA: O'Reilly Media, 2016), 37.

Bill Baker, un ... Simon Sharwood, "¿Son sus servidores MASCOTAS o GANADERAS?", *The Register* , 18 de marzo de 2013, http://www.theregister.co.uk/2013/03/18/servers_pets_or_cattle_cern/ .

En Netflix, el ... Jason Chan, "OWASP AppSecUSA 2012: Real Cloud Application en la nube Seguridad ", video de YouTube, 37:45, publicado por Christiaan008, 10 de diciembre de 2012, https://www.youtube.com/watch?v=daNA0jXDvYk .

El último patrón ... Chad Fowler, "Trash Your Servers and Burn Your Code: Immutable Infraestructura y componentes desechables ", ChadFowler.com, 23 de junio de 2013, http://chadfowler.com/2013/06/23/immutable-deployments.html .

La aplicación completa ... John Willis, "Docker y las tres formas de DevOps Parte 1: El First Way — Systems Thinking " , *Docker* , 26 de mayo de 2015, https://blog.docker.com/2015/05/docker-three-ways-devops/ .

CAPÍTULO 10

Gary Gruver, ex ... Gary Gruver, correspondencia personal con Gene Kim, 2014.

Tuvieron problemas ... "DOES15 - Mike Bland - El dolor ha terminado, si lo quieres" Slideshare.net, publicado por Gene Kim, 18 de noviembre de 2015, http://www.slideshare.net/ITRevolution/does15-mike-bland-pain-is-over-if-you-want-it-55236521 .

Bland describe cómo ... Ibid.

Bland describió que ... Ibid.

Como Bland describe ... Ibid.

Como señala Bland ... Ibid.

En el próximo ... Ibid.

Eran Messeri, un ... Eran Messeri, “Lo que sale mal cuando miles de ingenieros comparten ¿La misma construcción continua? ”, presentación en la Conferencia GOTO, Aarhus, Dinamarca, 2 de octubre de 2013.

Messeri explica: "Hay ... Ibid.

Todo su código ... Ibid.

Algunos de los ... Ibid.

En Desarrollo, continuo ... Jez Humble y David Farley, correspondencia personal con Gene Kim, 2012.

La tubería de despliegue ... Jez Humble y David Farley, *Entrega continua: confiable Lanzamientos de software a través de la automatización de compilación, prueba e implementación* (Upper Saddle River, NJ: Addison-Wesly, 2011), 3.

Humilde y Farley ... Ibid., 188.

Como Humble y ... Ibid., 258.

Martin Fowler observa ... Martin Fowler, "Integración continua", MartinFowler.com, 1 de mayo de 2006 http://www.martinfowler.com/articles/continuousIntegration.html .

Martin Fowler describió ... Martin Fowler, "TestPyramid", MartinFowler.com, 1 de mayo, 2012, http://martinfowler.com/bliki/TestPyramid.html .

Esta técnica fue ... Martin Fowler, "Test Driven Development", MartinFowler.com, 5 de marzo de 2005, http://martinfowler.com/bliki/TestDrivenDevelopment.html .

Nachi Nagappan, E. Michael ... Nachiappan Nagappan, E. Michael Maximilien,

Thirumalesh Bhat y Laurie Williams, “Realizando mejoras de calidad a través de pruebas desarrollo impulsado: resultados y experiencias de cuatro equipos industriales ", *Empir Software Ingeniería* , 13, (2008): 289-302, http://research.microsoft.com/en-us / groups / ese / nagappan_tdd.pdf .

En su 2013 ... Elisabeth Hendrickson, "Sobre el cuidado y la alimentación de los ciclos de retroalimentación" Slideshare.net, publicado por Elisabeth Hendrickson, 1 de noviembre de 2013, http://www.slideshare.net/ehendrickson/care-and-feeding-of-feedback-cycles .

Sin embargo, simplemente automatizando ... "Disminuyendo los falsos positivos en las pruebas automatizadas" Slideshare.net, publicado por Sauce Labs, 24 de marzo de 2015, http://www.slideshare.net/saucelabs/decreasing-false-positives-in-automated-testing .
Martin Fowler, "Erradicación del no determinismo en las pruebas", MartinFowler.com, 14 de abril, 2011, http://martinfowler.com/articles/nonDeterminism.html .

Como Gary Gruver ... Gary Gruver, "DOES14 - Gary Gruver - Macy's - Transformando Procesos tradicionales de desarrollo de software empresarial ", video de YouTube, 27:24, publicado por DevOps Enterprise Summit 2014, 29 de octubre de 2014, https://www.youtube.com/watch?v=-HSSGiYXA7U .

Randy Shoup, ex ... Randy Shoup, "El ciclo virtuoso de la velocidad: lo que aprendí Acerca de ir rápido en eBay y Google por Randy Shoup ", video de YouTube, 30:05, publicado por Flowcon, 26 de diciembre de 2013, https://www.youtube.com/watch?v=EwLBoRyXTOI .

Esto es a veces ... David West, "La caída del agua es una realidad_de_agile_para_la mayoría" Slideshare.net, publicado por harsoft, 22 de abril de 2013, http://www.slideshare.net/harsoft/water-scrumfall-isrealityofagileformost .

CAPÍTULO 11

La sorprendente amplitud ... Gene Kim, "La asombrosa transformación de DevOps de HP Equipo de firmware de LaserJet (Gary Gruver), "ITRevolution.com, 2013, http://itrevolution.com/the-amazing-devops-transformation-of-the-hp-laserjet-firmware-team-gary-gruver / .

Gruver describió esto ... Ibid.

Compilar banderas (#define ... Ibid.

Gruver admite que está basado en troncales ... Gary Gruver y Tommy Mouser, *Liderando el Transformación: aplicación de principios ágiles y DevOps a escala* (Portland, OR: IT Revolution Press), 60.

Gruver observó: "Sin ... Kim," La asombrosa transformación de DevOps " ITRevolution.com.

Jeff Atwood, fundador ... Jeff Atwood, "Ramificación de software y universos paralelos" CodingHorror.com, 2 de octubre de 2007, http://blog.codinghorror.com/software-branching-y-universos-paralelos / .

Así es como ... Ward Cunningham, "Ward explica la metáfora de la deuda", c2.com, 2011, http://c2.com/cgi/wiki?WardExplainsDebtMetaphor .

Ernest Mueller, quien ... Ernest Mueller, "2012: A Release Odyssey", Slideshare.net, publicó por Ernest Mueller, 12 de marzo de 2014, http://www.slideshare.net/mxyzplk/2012-a-release-Odyssey .

En ese momento ... "Bazaarvoice, Inc. anuncia sus resultados financieros para el Cuarto Fiscal Trimestre y año fiscal finalizado el 30 de abril de 2012 ", BasaarVoice.com, 6 de junio de 2012, http://investors.bazaarvoice.com/releasedetail.cfm?ReleaseID=680964 .

Mueller observó: "Es ... Ernest Mueller", DOES15 - Ernest Mueller - DevOps Transformaciones en National Instruments y ... ", video de YouTube, 34:14, publicado por DevOps Cumbre empresarial, 5 de noviembre de 2015, https://www.youtube.com/watch? v = 6Ry40h1UAyE .

"Al ejecutar estos ... Ibid.

Mueller describió más detalladamente ... Ibid.

Sin embargo, los datos ... Kersten, IT Revolution, y PwC, *2015 State of DevOps Report* .

CAPITULO 12

En 2012, Rossi ... Chuck Rossi, "Liberar ingeniería y empujar karma: Chuck Rossi", publica en Página de Facebook de Chuck Rossi, 5 de abril de 2012, https://www.facebook.com/notes/facebook-ingeniería / release-engineering-and-push-karma-chuck-rossi / 10150660826788920 ,

Justo antes de ... Ryan Paul, "Exclusivo: una mirada detrás de escena al lanzamiento de Facebook ingeniería ", *Ars Technica* , 5 de abril de 2012, http://arstechnica.com/business/2012/04/exclusive-a-behind-the-scenes-look-at-facebook-release-engineering / 1 / .

Rossi continuó: "Si ... Chuck Rossi," Libera la ingeniería y empuja el karma ".

El front-end de Facebook ... Paul, "Exclusivo: una mirada detrás de escena al lanzamiento de Facebook ingeniería ", *Ars Technica* .

Explicó que ... Chuck Rossi, "Envíe temprano y envíe el doble de veces", publique en Chuck Página de Facebook de Rossi, 3 de agosto de 2012, https://www.facebook.com/notes/facebook-ingeniería / envío temprano y envío dos veces más frecuente / 10150985860363920 ,

Kent Beck, el ... Kent Beck, "Despliegue lento provoca reuniones", publicación en Kent Beck Página de Facebook, 19 de noviembre de 2015), https://www.facebook.com/notes/kent-beck/slow-implementación-causas-reuniones / 10554273711567937 _rdr = p ,

Scott Prugh, su ... Prugh, "DOES14: Scott Prugh, CSG - DevOps y Lean in Legacy Ambientes ".

Prugh observó: "Es ... Ibid.

Prugh escribe: "Nosotros ... Ibid.

Prugh también observa: ... Ibid.

En sus experimentos ... Puppet Labs y IT Revolution Press, *2013 State of DevOps Report* (Portland, OR: Puppet Labs, 2013), http: //www.exin-

library.com/Player/eKnowledge/2013-state-of-devops-report.pdf .

Prugh informó que ... Scott Prugh y Erica Morrison, "DOES15 - Scott Prugh y Erica Morrison - Conway & Taylor Meet the Strangler (v2.0) ", video de YouTube, 29:39, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015, https://www.youtube.com/watch?v = tKdIHCL0DUg .

Considere lo siguiente ... Tim Tischler, conversación personal con Gene Kim, FlowCon 2013

En la práctica, los ... Puppet Labs y IT Revolution Press, *2013 State of DevOps Report.*

En Puppet Labs ' ... Velásquez, Kim, Kersten y Humble, *2014 State of DevOps Report* .

El proceso de implementación ... Chad Dickerson, "Optimización para la felicidad del desarrollador" CodeAsCraft.com, 6 de junio de 2011, https://codeascraft.com/2011/06/06/optimizing-for-desarrollador-felicidad / .

Como Noah Sussman ... Noah Sussman y Laura Beth Denker, "Divide and Conquer" CodeAsCraft.com, 20 de abril de 2011, https://codeascraft.com/2011/04/20/divide-and-concurrir / .

Sussman escribe: "A través de ... Ibid.

Si todas las pruebas ... Ibid.

Una vez que es un ... Erik Kastner, "Quantum of Deployment", CodeAsCraft.com, 20 de mayo de 2010, https://codeascraft.com/2010/05/20/quantum-of-deployment/ .

Esta técnica fue ... Timothy Fitz, "Despliegue continuo en IMVU: haciendo el imposible cincuenta veces al día ", TimothyFitz.com, 10 de febrero de 2009, http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-imposible cincuenta veces al día .

Este patrón es ... Fitz, "Despliegue continuo", TimothyFitz.com .; Michael Hrenko "DOES15 - Michael Hrenko - DevOps asegurados por Blue Shield of California", YouTube video, 42:24, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015, https://www.youtube.com/watch?v=NlgrOT24UDw .

Dan North y Dave ... Humble y Farley, *Entrega continua* , 265.

El grupo inmune ... Eric Ries, *The Lean Startup: cómo utilizan los emprendedores de hoy Innovación continua para crear negocios radicalmente exitosos* (Nueva York: Aleatorio House, 2011), Audiolibro.

Un ejemplo sofisticado ... Andrew 'Boz' Bosworth, "Construyendo y probando en Facebook" publicar en la página de Facebook de Boz, 8 de agosto de 2012, https://www.facebook.com/notes/facebook-ingeniería / construcción y prueba en facebook / 10151004157328920 ; "Característica de Etsy marcado API utilizado para rampas operativas y pruebas A / B ", GitHub.com,

https://github.com/etsy/feature ; "Biblioteca para la API de gestión de configuración" GitHub.com https://github.com/Netflix/archaius .

En 2009, cuando ... John Allspaw, "Gestión convincente de que la cooperación y la colaboración valió la pena ", KitchenSoap.com, 5 de enero de 2012, http://www.kitchensoap.com/2012/01/05/convincing-management-that-cooperation- y la colaboración valió la pena .

386 de 1189.

Del mismo modo, como Chuck ... Rossi, "Libera la ingeniería y empuja el karma".

Durante casi una década ... Emil Protalinski, "Facebook supera 1.55B usuarios activos mensuales y 1.01B usuarios activos diarios ", *Venture Beat* , 4 de noviembre de 2015, http://venturebeat.com/2015/11/04/facebook-passes-1-55b-monthly-active-users-and-1- 01 mil millones de usuarios activos diarios .

Para 2015, Facebook ... Ibid.

Eugene Letuchy, un ... Eugene Letuchy, "Facebook Chat", publicación en Eugene Letuchy Página de Facebook, 3 de mayo de 2008, http://www.facebook.com/note.php? note_id = 14218138919 & id = 944554719 .

Implementando esto computacionalmente intensivo ... Ibid.

Como Letuchy escribió ... Ibid.

Sin embargo, en 2015 ... Jez Humble, correspondencia personal con Gene Kim, 2014.

Sus definiciones actualizadas ... Ibid.

En Amazon y ... Ibid.

CAPITULO 13

Este es el ... Jez Humble, "¿Qué es la entrega continua", ContinuousDelivery.com, consultado el 28 de mayo de 2016, https://continuousdelivery.com/ .

Él observa que ... Kim, Gruver, Shoup y Phillips, "Explorando el territorio desconocido" de microservicios ".

Él reflexiona: "Mirando ... Ibid.

La arquitectura de eBay fue ... Shoup, "Del monolito a los microservicios".

Charles Betz, autor ... Charles Betz, *Arquitectura y patrones para el servicio de TI Gestión, planificación de recursos y gobernanza: hacer zapatos para el zapatero Niños* (Witham, MA: Morgan Kaufmann, 2011), 300.

Como Randy Shoup ... Randy Shoup, "Del monolito a los microservicios", Slideshare.net, publicado por Randy Shoup, 8 de octubre de 2014, http://www.slideshare.net/RandyShoup/goto- aarhus2014-enterprisearchitecturemicroservices .

Shoup señala: "Organizaciones ... Ibid.

Como observa Randy Shoup ... Ibid.

Uno de los más ... Werner Vogels, "Una conversación con Werner Vogels", *acmqueque* 4, no. 4 (2006): 14-22, http://queue.acm.org/detail.cfm?id=1142065 .

Vogels le dice a Gray ... Ibid.

Describiendo el pensamiento ... Ibid.

Vogels señala: "El ... Ibid.

En 2011, Amazon ... John Jenkins, "Velocity 2011: Jon Jenkins," Velocity Culture, "" Video de YouTube, 15:13, publicado por O'Reilly, 20 de junio de 2011, { https://www.youtube.com/watch?v=dxk8b9rSKOo .

---

387 de 1189.

Para 2015, ellos ... Ken Exner, "Transformando el desarrollo de software", video de YouTube, 40:57, publicado por Amazon Web Services, 10 de abril de 2015, https://www.youtube.com/watch?v = YCrhemssYuI & feature = youtu.be .

El término estrangulador ... Martin Fowler, "StranglerApplication", MartinFowler.com, 29 de junio, 2004, http://www.martinfowler.com/bliki/StranglerApplication.html .

Cuando implementamos ... Boris Lublinsky, "Versiones en SOA", *The Architecture Journal* , Abril de 2007 https://msdn.microsoft.com/en-us/library/bb491124.aspx .

La aplicación de estrangulador ... Paul Hammant, "Introducción de rama por abstracción" PaulHammant.com, 26 de abril de 2007, http://paulhammant.com/blog/branch_by_abstraction.html .

Una observación de ... Martin Fowler, "StranglerApplication", MartinFowler.com, junio 29, 2004, http://www.martinfowler.com/bliki/StranglerApplication.html .

Blackboard Inc., es ... Gregory T. Huang, "CEO de Blackboard Jay Bhatt sobre el futuro global de Edtech ", *Xconomy,* 2 de junio de 2014, http://www.xconomy.com/boston/2014/06/02/blackboard-ceo-jay-bhatt-on-the-global-futuro de edtech / .

Como David Ashman ... David Ashman, "DOES14 - David Ashman - Blackboard Learn - Keep Tu cabeza en las nubes ", video de YouTube, 30:43, publicado por DevOps Enterprise Summit 2014, 28 de octubre de 2014, https://www.youtube.com/watch?v=SSmixnMpsI4 .

En 2010, Ashman ... Ibid.

Cómo comenzó esto ... David Ashman, correspondencia personal con Gene Kim, 2014.

Ashman señaló: "Para ... Ibid.

"De hecho", Ashman ... Ibid.

Ashman concluyó: "Teniendo ... Ibid.

CAPITULO 14

En Operaciones, nosotros ... Kim, Behr y Spafford, *The Visible Ops Handbook: Implementing ITIL en 4 pasos prácticos y auditables* (Eugene, OR: IT Process Institute, 2004), Kindle edición, Introducción.

En contraste, el ... Ibid.

En otras palabras ... Ibid.

Para habilitar esto ... "Telemetría", *Wikipedia* , última modificación el 5 de mayo de 2016, https://en.wikipedia.org/wiki/Telemetry .

McDonnell describió cómo ... Michael Rembetsy y Patrick McDonnell, "Continuamente Desplegando Cultura: Escalando la Cultura en Etsy - Velocity Europe 2012 ", Slideshare.net, publicado por Patrick McDonnell, 4 de octubre de 2012, http://www.slideshare.net/mcdonnps/continuously-deploying-culture-scaling-culture-at-etsy-14588485 .

McDonnell explicó más ... Ibid.

Para 2011, Etsy ... John Allspaw, conversación personal con Gene Kim, 2014.

Como Ian Malpass ... Ian Malpass, "Mide cualquier cosa, mide todo" CodeAsCraft.com, 15 de febrero de 2011, http://codeascraft.com/2011/02/15/measure-nada-medir-todo / .

Uno de los hallazgos ... Kersten, IT Revolution y PwC, *2015 State of DevOps Report* .

Los dos primeros ... "2014 Estado de DevOps resultados! Conferencia de velocidad ", Slideshare.net, publicado por Gene Kim, 30 de junio de 2014, http://www.slideshare.net/realgenekim/2014-state-of-devops-results-velocity-conference .

En The Art ... James Turnbull, *The Art of Monitoring* (Seattle, WA: Amazon Digital Servicios, 2016), edición Kindle, Introducción.

La capacidad resultante ... "Monitorama - Por favor, no más minutos, milisegundos, Monolitos o herramientas de monitoreo ", Slideshare.net, publicado por Adrian Cockcroft, 5 de mayo de 2014, http://www.slideshare.net/adriancockcroft/monitorama-please-no-more .

Scott Prugh, Jefe ... Prugh, “DOES14: Scott Prugh, CSG - DevOps y Lean in Legacy Ambientes ".

Para apoyar estos ... Brice Figureau, "Los 10 mandamientos de la tala", Blog de Mastersen, 13 de enero de 2013, http://www.masterzen.fr/2013/01/13/the-10-commandments-of-registro / .

Elegir el derecho ... Dan North, correspondencia personal con Gene Kim, 2016.

Para ayudar a garantizar ... Anton Chuvakin, "LogLogic / Chuvakin Log Checklist", republicado con permiso, 2008, http://juliusdavies.ca/logging/llclc.html .

En 2004, Kim ... Kim, Behr y Spafford, *The Visible Ops Handbook* , Introducción.

Este fue el ... Dan North, "Operaciones y operabilidad", SpeakerDeck.com, 25 de febrero de 2016, https://speakerdeck.com/tastapod/ops-and-operability .

Como John Allspaw ... John Allspaw, correspondencia personal con Gene Kim, 2011.

Esto es a menudo ... "Radiadores de información", AgileAlliance.com, consultado el 31 de mayo de 2016, https://www.agilealliance.org/glossary/incremental-radiators/ .

Aunque puede haber ... Ernest Mueller, correspondencia personal con Gene Kim, 2014.

Prachi Gupta, Director ... Prachi Gupta, "Visualizando el rendimiento del sitio de LinkedIn" Blog de ingeniería de LinkedIn, 13 de junio de 2011, https://engineering.linkedin.com/25/visualizing-linkedins-site-performance .

Así comenzó Eric ... Eric Wong, "Eric el interno: el origen de InGraphs", LinkedIn, junio 30 de 2011 http://engineering.linkedin.com/32/eric-intern-origin-ingraphs .

Wong escribió: "Para ... Ibid.

En ese momento ... Ibid.

Al escribir sobre ... Gupta, "Visualizando el rendimiento del sitio de LinkedIn".

Ed Blankenship, Senior ... Ed Blankenship, correspondencia personal con Gene Kim, 2016.

Sin embargo, cada vez más estos ... Mike Burrows, "El servicio de bloqueo Chubby por vagamente-sistemas distribuidos acoplados, " *OSDI'06: Séptimo Simposio sobre Diseño de Sistemas Operativos*

389 de 1189.

e Implementación , noviembre de 2006, http://static.googleusercontent.com/media/research.google.com/en//archive/chubby-osdi06.pdf .

El cónsul puede ser ... Jeff Lindsay, "Descubrimiento del servicio de cónsul con Docker", Progrium.com, 20 de agosto de 2014, http://progrium.com/blog/2014/08/20/consul-service-discovery-with-cargador de muelle .

Como Jody Mulkey ... Jody Mulkey, "DOES15 - Jody Mulkey - DevOps en la empresa: A Transformation Journey ", video de YouTube, 28:22, publicado por DevOps Enterprise Summit, 5 de noviembre de 2015 https://www.youtube.com/watch?v=USYrDaPEFtM .

CAPITULO 15

En 2015, Netflix ... Carta de Netflix a los accionistas , 19 de enero de 2016,

http://files.shareholder.com/downloads/NFLX/2432188684x0x870685/C6213FF9-5498-4084-A0FF-74363CEE35A1 / Q4_15_Letter_to_Shareholders_-_COMBINED.pdf .

Roy Rapoport describe ... Roy Rapoport, correspondencia personal con Gene Kim, 2014.

Una de las estadísticas ... Victoria Hodge y Jim Austin, "Una encuesta de detección de valores atípicos Metodologías ", *Artificial Intelligence Review* 22, no. 2 (octubre de 2004): 85-126, http://www.geo.upm.es/postgrado/CarlosLopez/papers / Hodge + Austin_OutlierDetection_AIRE381.pdf .

Rapoport explica que ... Roy Rapoport, correspondencia personal con Gene Kim, 2014.

Rapoport continúa: "Nosotros ... Ibid.

Rapoport afirma que ... Ibid.

Como John Vincent ... Toufic Boubez, "Matemáticas simples para la detección de anomalías toufic boubez - software metafor - monitorama pdx 2014-05-05 ", Slideshare.net, publicado por tboubez, mayo 6, 2014, http://www.slideshare.net/tboubez/simple-math-for-anomaly-detection-toufic-boubez-metafor-software-monitorama-pdx-20140505 .

Tom Limoncelli, coautor ... Tom Limoncelli, "Deja de monitorear si tu service is up! ", EverythingSysAdmin.com, 27 de noviembre de 2013, http://everythingsysadmin.com/2013/11/stop-monitoring-if-service-is-up.html .

Como el Dr. Toufic ... Toufic Boubez, "Matemáticas simples para la detección de anomalías toufic boubez - metafor software - monitorama pdx 2014-05-05 ", Slideshare.net, publicado por tboubez, 6 de mayo de 2014, http://www.slideshare.net/tboubez/simple-math-for-anomaly-detection-toufic-boubez-metafor-software-monitorama-pdx-20140505 .

Dra. Nicole Forsgren ... Dra. Nicole Forsgren, correspondencia personal con Gene Kim, 2015.

Scryer trabaja por ... Daniel Jacobson, Danny Yuan y Neeraj Joshi, "Scryer: Netflix's Motor de escalado automático predictivo ", *The Netflix Tech Blog* , 5 de noviembre de 2013, http://techblog.netflix.com/2013/11/scryer-netflixs-predictive-auto-scaling.html .

Estas técnicas son ... Varun Chandola, Arindam Banerjee y Vipin Kumar, "Anomalía detección: una encuesta, " *ACM Computing Surveys* 41, no. 3 (julio de 2009): artículo no. 15, http://doi.acm.org/10.1145/1541880.1541882 .

Página 390

Tarun Reddy, VP ... Tarun Reddy, entrevista personal con Gene Kim, sede del Rally, Boulder, CO, 2014.

En Monitorama en 2014 ... "Prueba de Kolmogorov-Smirnov" , *Wikipedia* , última modificación el 19 de mayo, 2016, http://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test .

Incluso diciendo Kilmogorov-Smirnov ... "Matemáticas simples para la detección de anomalías toufic boubez - software metafor - monitorama pdx 2014-05-05 ", Slideshare.net, publicado por tboubez, mayo 6, 2014, http://www.slideshare.net/tboubez/simple-math-for-anomaly-detection -toufic-

boubez-metafor-software-monitorama-pdx-20140505.

CAPITULO 16

En 2006, Nick ... Mark Walsh, "Ad Firms Right Media, AdInterax vende a Yahoo" *MediaPost*, 18 de octubre de 2006, http://www.mediapost.com/publications/article/49779/ad-firmas-derecho-medios-adinterax-sell-to-yahoo.html? edition = .

Galbreath describió el ... Nick Galbreath, conversación personal con Gene, 2013.

Sin embargo, Galbreath observó ... Nick Galbreath, "Despliegue continuo - El nuevo # 1 Función de seguridad, de BSildesLA 2012 ", Slideshare.net, publicado por Nick Galbreath, 16 de agosto, 2012, http://www.slideshare.net/nickgsuperstar/continuous-deployment-the-new-1-Característica de seguridad .

Después de observar muchos ... Ibid.

Galbreath observa que ... Ibid.

Como Patrick Lightbody ... "Volocity 2011: Patrick Lightbody, 'Desde el inicio hasta la adquisición'" Video de YouTube, 15:28, publicado por O'Reilly, 17 de junio de 2011, https://www.youtube.com/watch? v = ShmPod8JecQ.

Como Arup Chakrabarti ... Arup Chakrabarti, "Errores comunes de operaciones", presentación en Heavy Bit Industries, 3 de junio de 2014, http: // www .heavybit.com / library / video / common-ops-errores /

Más recientemente, Jeff ... "Desde Design Thinking hasta DevOps y Back Again: Unifying Design & Operations ", video de Vimeo, 21:19, publicado por William Evans, 5 de junio de 2015, https://vimeo.com/129939230.

Como anónimo ... Anónimo, conversación personal con Gene Kim, 2005.

Guía de lanzamiento y ... Tom Limoncelli, "SRE @ Google: Miles de DevOps desde 2004 ", video de YouTube de USENIX Association Talk, NYC, publicado por USENIX, 45:57, publicado el 12 de enero de 2012, http://www.youtube.com/watch?v=iIuTnhdTzK .

Como Treynor Sloss tiene ... Ben Treynor, "Claves para SRE" (presentación, Usenix SREcon14, Santa Clara, CA, 30 de mayo de 2014), https://www.usenix.org/conference/srecon14/technical-sesiones / presentación / keys-sre .

Treynor Sloss ha resistido ... Ibid.

Incluso cuando es nuevo ... Limoncelli, "SRE @ Google".

Tom Limoncelli señaló ... Ibid.

Limoncelli señaló: "En ... Ibid.

Además, Limoncelli observó ... Tom Limoncelli, correspondencia personal con Gene

Kim, 2016.

Limoncelli explicó: "Ayudando ... Ibid., 2015.

CAPITULO 17

En general, Jez ... Humble, O'Reilly y Molesky, *Lean Enterprise* , Parte II.

En 2012, ellos ... Intuit, Inc., "Informe anual 2012: Formulario 10-K", 31 de julio de 2012, http://s1.q4cdn.com/018592547/files/doc_financials/2012 / INTU_2012_7_31_10K_r230_at_09_13_12_FINAL_and_Camera_Ready.pdf .

Cook explicó que ... Scott Cook, "Liderazgo en una era ágil: una entrevista con Scott Cook ", Intuit.com, 20 de abril de 2011, https://web.archive.org/web/20160205050418/http://network.intuit.com/2011/04/20/leadership-in-the-agile-age/

Él continuó: "Por ... Ibid.

En eras anteriores ... "Marketing directo", Wikipedia, última modificación el 28 de mayo de 2016, https://en.wikipedia.org/wiki/Direct_marketing .

Curiosamente, tiene ... Freakonomics, "Combatir la pobreza con evidencia real: completo Transcripción ", Freakonomics.com, 27 de noviembre de 2013, http://freakonomics.com/2013/11/27/fighting-poverty-with-actual-evidence-full-transcripción / .

Ronny Kohavi, distinguido ... Ron Kohavi, Thomas Crook y Roger Longbotham, "Experimentación en línea en Microsoft" (documento presentado en el Decimoquinto ACM SIGKDD Conferencia internacional sobre descubrimiento de conocimiento y minería de datos, París, Francia, 2009), http://www.exp-platform.com/documents/exp_dmcasestudies.pdf .

Kohavi continúa ... Ibid.

Jez Humble bromeó ... Jez Humble, correspondencia personal con Gene Kim, 2015.

En un 2014 ... Wang, Kendrick, "Cultura de experimentación continua de Etsy y A / B Testing Spurs Mobile Innovation ", Apptimize.com, 30 de enero de 2014, http://apptimize.com/blog/2014/01/etsy-continuous-innovation-ab-testing/ .

Barry O'Reilly, coautor ... Barry O'Reilly, "Cómo implementar una hipótesis basada en Desarrollo ", BarryOReilly.com, 21 de octubre de 2013, http://barryoreilly.com/2013/10/21/how-to-implement-hypothesis-driven-development/ .

En 2009, Jim ... Gene Kim, "Aprendizaje organizacional y competitividad: revisando el "Allspaw / Hammond 10 implementaciones por día en Flickr" Story ", ITRevolution.com, 2015, http://itrevolution.com/organizational-learning-and-competitiveness-a-different-view-of-the-allspawhammond-10-deploys-per-day-at-flickr-story / .

Stoneham observa que ... Ibid.

Él continúa: "Estos ... Ibid.

Sus asombrosos logros ... Ibid.

Stoneham concluyó: "Esto ... Ibid.

CAPITULO 18

Once a pull ... Scott Chacon, "Github Flow", ScottChacon.com, 31 de agosto de 2011, http://scottchacon.com/2011/08/31/github-flow.html .

Por ejemplo, en ... Jake Douglas, "Implementación en Github", GitHub.com, 29 de agosto de 2012, https://github.com/blog/1241-deploying-at-github .

A quince minutos ... John Allspaw, "Pensamiento contrafactual, Reglas y la Capital del Caballero Accidente ", KitchenSoap.com, 29 de octubre de 2013, http://www.kitchensoap.com/2013/10/29/counterfactuals-knight-capital/ .

Uno de los principales ... Bradley Staats y David M. Upton, "Lean Knowledge Work", *Harvard Business Review* , octubre de 2011,https://hbr.org/2011/10/lean-knowledge-work .

En el 2014 ... Velásquez, Kim, Kersten y Humble, *Informe del Estado de DevOps 2014* .

Como Randy Shoup ... Randy Shoup, entrevista personal con Gene Kim, 2015.

Como Giary Özil ... Giray Özil, publicación de Twitter, 27 de febrero de 2013, 10:42 am, https://twitter.com/girayozil/status/306836785739210752 .

Como se señaló anteriormente ... Eran Messeri, "Lo que sale mal cuando miles de ingenieros comparten the Same Continuous Build? "(2013), http://scribes.tweetscriber.com/realgenekim/206 .

En 2010, allí ... John Thomas y Ashish Kumar, "Bienvenidos a Google Engineering Blog de herramientas ", blog de Google Engineering Tools, publicado el 3 de mayo de 2011, http: // google-engtools.blogspot.com/2011/05/welcome-to-google-engineering-tools.html .

Esto requiere considerable ... Ashish Kumar, "Desarrollo a la velocidad y escala de Google "(presentación en QCon, San Francisco, CA, 2010), https://qconsf.com/sf2010/dl/qcon-sanfran-2010 / slides / AshishKumar_DevelopingProductsattheSpeedandScaleofGoogle.pdf .

Él dijo: "Yo ... Randy Shoup, correspondencia personal con Gene Kim, 2014.

Jeff Atwood, uno ... Jeff Atwood, "Programación de pares frente a revisiones de código" CodingHorror.com, 18 de noviembre de 2013, http://blog.codinghorror.com/pair-programación-vs-código-revisiones / .

Él continuó: "La mayoría ... Ibid.

La Dra. Laurie Williams realizó ... "Programación de pares", página ALICE Wiki, última modificación 4 de abril de 2014, http://euler.math.uga.edu/wiki/index.php?title=Pair_programming .

Ella argumenta que ... Elisabeth Hendrickson, "DOES15 - Elisabeth Hendrickson - Todo se trata Comentarios ", video de YouTube, 34:47, publicado por DevOps Enterprise Summit, 5 de noviembre, 2015, https://www.youtube.com/watch?v=r2BFTXBundQ .

En su 2015 ... Ibid.

El problema Hendrickson ... Ibid.

Peor, desarrolladores expertos ... Ibid.

Hendrickson lamentó que ... Ibid.

Ese fue un verdadero ... Ryan Tomayko y Shawn Davenport, entrevista personal con Gene Kim, 2013.

Página 393

Es muchos ... Ibid.

Leyendo el ... Ibid.

Adrian Cockcroft observó ... Adrian Cockcroft, entrevista de Michael Ducy y Ross Clanton, "Adrian Cockcroft de Battery Ventures - The Goat Farm - Episode 8", *The Goat Farm* , podcast audio, 31 de julio de 2015, http://goatcan.do/2015/07/31/adrian-cockcroft-of-batería-empresas-la-cabra-granja-episodio-8 / .

Del mismo modo, Dr. Tapabrata Pal ... Tapabrata Pal, "DOES15 - Tapabrata Pal - Banking on Innovación y DevOps ", video de YouTube, 32:57, publicado por DevOps Enterprise Summit, 4 de enero de 2016, https://www.youtube.com/watch?v=bbWFCKGhxOs .

Jason Cox, Senior ... Jason Cox, "Disney DevOps".

En Target en ... Ross Clanton y Heather Mickman, 'DOES14 - Ross Clanton y Heather Mickman - DevOps at Target ", video de YouTube, 29:20, publicado por DevOps Enterprise Cumbre 2014, 29 de octubre de 2014, https://www.youtube.com/watch?v=exrjV9V9vhY .

"A medida que avanzamos ... Ibid.

Ella agregó: "Yo ... Ibid.

Considere una historia ... John Allspaw y Jez Humble, correspondencia personal con Gene Kim, 2014.

CAPITULO 19

El resultado es ... Spear, *The High-Velocity Edge* , cap. 1)

"Para tal ... Ibid., Cap. 10)

Un ejemplo sorprendente ... Julianne Pepitone, "Amazon EC2 Outage Downs Reddit, Quora" *CNN Money* , 22 de abril de 2011, http://money.cnn.com/2011/04/21/technology/amazon_server_outage .

En enero de 2013 ... Timothy Prickett Morgan, "Un vistazo raro a la escala masiva de AWS ", *Enterprise Tech* , 14 de noviembre de 2014, http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/ .

Sin embargo, un Netflix ... Adrian Cockcroft, Cory Hicks y Greg Orzell, "Lecciones Netflix Aprendí de la interrupción de AWS ", *The Netflix Tech Blog* , 29 de abril de 2011, http://techblog.netflix.com/2011/04/lessons-netflix-learned-from-aws-outage.html .

Lo hicieron ... Ibid.

Dr. Sidney Dekker ... Sidney Dekker, *Just Culture: Equilibrando la seguridad y la rendición de cuentas*

(Universidad de Lund, Suecia: Ashgate Publishing Company, 2007), 152.

Afirma que ... "DevOpsDays Brisbane 2014 - Sidney Decker - Falla del sistema, humano Error: ¿Quién tiene la culpa? Video de Vimeo, 1:07:38, publicado por info@devopsdays.org, 2014, https://vimeo.com/102167635 .

Como John Allspaw ... Jenn Webb, entrevista con John Allspaw, "Post-Mortems, Sans Finger-Apuntando ", el *Postcast Radar O'Reilly* , podcast audio, 21 de agosto de 2014, http://radar.oreilly.com/2014/08/postmortems-sans-finger-pointing-the-oreilly-radar-podcast.html .

Página 394

Post-mortems sin culpa, un ... John Allspaw, "PostMortems sin culpa y una cultura justa" CodeAsCraft.com, 22 de mayo de 2012, http://codeascraft.com/2012/05/22/blameless-postmortems / .

Ian Malpass, un ... Ian Malpass, "DevOpsDays Minneapolis 2014 - Ian Malpass, Fallible humanos ", video de YouTube, 35:48, publicado por DevOps Minneapolis, 20 de julio de 2014, https://www.youtube.com/watch?v=5NY-SrQFrBU .

Dan Milstein, uno ... Dan Milstein, "Post-Mortems en HubSpot: lo que aprendí de 250 Por qué ", *HubSpot* , 1 de junio de 2011, http://product.hubspot.com/blog/bid/64771/Post-Mortems-at-HubSpot-What-I-Learned-From-250-Whys .

Randy Shoup, ex ... Randy Shoup, correspondencia personal con Gene Kim, 2014.

También podemos ... "Post-Mortem para la interrupción del 24 de febrero de 2010", el sitio web de Google App Engine, 4 de marzo de 2010, https://groups.google.com/forum/#!topic/google-appengine / p2QKJ0OSLc8 ; "Resumen de la interrupción del servicio de Amazon DynamoDB y Impactos relacionados en la región este de los Estados Unidos ", sitio web de Amazon Web Services, consultado el 28 de mayo, 2016, https://aws.amazon.com/message/5467D2/ .

Este deseo de ... Bethany Macri, "Morgue: ayudar a comprender mejor los eventos mediante la construcción de un Herramienta Post Mortem - Bethany Macri ", video de Vimeo, 33:34, publicado por info@devopsdays.org, 18 de octubre de 2013, http://vimeo.com/77206751 .

Por ejemplo, como ... Spear, *The High-Velocity Edge* , cap. 4)

Dra. Amy C. Edmondson ... Amy C. Edmondson, "Estrategias para aprender del fracaso" *Harvard Business Review* , abril de 2011, https://hbr.org/2011/04/strategies-for-learning-de fallo .

El Dr. Spear resume ... Ibid.

Ahora sabemos ... Ibid., Cap. 3)

Sin embargo, antes de ... Michael Roberto, Richard MJ Bohmer y Amy C. Edmondson, "Enfrentando amenazas ambiguas", *Harvard Business Review* , noviembre de 2006, https://hbr.org/2006/11/facing-ambiguous-threats/ar/1 .

Describen cómo ... Ibid.

Ellos observan: "Firmas ... Ibid.

Los autores concluyen ... Ibid.

Sobre los fracasos, Roy ... Roy Rapoport, correspondencia personal con Gene Kim, 2012.

Él continúa: "Yo ... Ibid.

Concluye: "DevOps ... Ibid.

Como Michael Nygard ... Michael T. Nygard, ¡suéltelo *!: Diseño e implementación de producción Software listo* (Biblioteca pragmática: Raleigh, NC, 2007), edición Kindle, Parte I.

Incluso más ... Jeff Barr, "Actualización de mantenimiento EC2" , *Blog de AWS* , 25 de septiembre de 2014, https://aws.amazon.com/blogs/aws/ec2-maintenance-update/ .

Como Christos Kalantzis ... Bruce Wong y Christos Kalantzis, "Un estado de Xen - Caos Monkey & Cassandra ", *The Netflix Tech Blog* , 2 de octubre de 2014, http://techblog.netflix.com/2014/10/a-state-of-xen-chaos-monkey-cassandra.html .

Página 395

Pero, Kalantzis continúa ... Ibid.

Como Kalantzis y ... Ibid.

Aún más sorprendente ... Roy Rapoport, correspondencia personal con Gene Kim, 2015.

Patrones arquitectónicos específicos ... Adrian Cockcroft, correspondencia personal con Gene Kim, 2012.

En esta sección ... Jesse Robbins, "GameDay: Creando resiliencia a través de la destrucción - LISA11 ", Slideshare.net, publicado por Jesse Robbins, 7 de diciembre de 2011, http://www.slideshare.net/jesserobbins/ameday-creating-resiliency-through-destruction .

Robbins define la resistencia ... Ibid.

Jesse Robbins observa ... Jesse Robbins, Kripa Krishnan, John Allspaw y Tom Limoncelli, "Ingeniería de resiliencia: Aprender a aceptar el fracaso", *amcqueue* 10, no. 9 9 (13 de septiembre de 2012): https://queue.acm.org/detail.cfm?id=2371297 .

Como bromea Robbins ... Ibid.

Como describe Robbins ... Ibid.

Robbins explica: "Tú ... Ibid.

Durante ese tiempo ... "Kripa Krishnan: 'Aprender continuamente de los fracasos' en Google" Video de YouTube, 21:35, publicado por Flowcon, 11 de noviembre de 2014, https://www.youtube.com/watch?v=KqqS3wgQum0 .

Krishnan escribió: "Un ... Kripa Krishnan," Envejeciendo lo inesperado ", *Comunicaciones de el ACM 55* , no. 11 (noviembre de 2012): 48-52,

http://cacm.acm.org/magazines/2012/11/156583-weathering-the-unexpected/abstract .
Algunos de los aprendizajes ... Ibid.

Como Peter Senge ... Ampliamente atribuido a Peter Senge.

CAPITULO 20

Como Jesse Newland ... Jesse Newland, "ChatOps en GitHub", SpeakerDeck.com, 7 de febrero, 2013, https://speakerdeck.com/jnewland/chatops-at-github .

Como Mark Imbriaco ... Mark Imbriaco, correspondencia personal con Gene Kim, 2015.

Permitieron Hubot ... Newland, "ChatOps en GitHub".

Hubot a menudo realizaba ... Ibid.

Newland observa que ... Ibid.

En lugar de poner ... Leon Osterweil, "Los procesos de software también son software", papel presentado en la Conferencia Internacional sobre Ingeniería de Software, Monterey, CA, 1987, http://www.cs.unibo.it/cianca/wwwpages/ids/letture/Osterweil.pdf .

Justin Arbuckle fue ... Justin Arbuckle, "¿Qué es ArchOps: Chef Executive Roundtable"? (2013)

Lo que resultó fue ... Ibid.

La conclusión de Arbuckle fue ... Ibid.

Página 396

Para 2015, Google ... Cade Metz, "Google tiene 2 mil millones de líneas de código, y todo está en uno Place ", *Wired* , 16 de septiembre de 2015, http://www.wired.com/2015/09/google-2-billion-código de líneas y un lugar / .

El Chrome y ... Ibid.

Rachel Potvin, una ... Ibid.

Además, como Eran ... Eran Messeri, "Lo que sale mal cuando miles de ingenieros ¿Compartir la misma construcción continua? (2013) http://scribes.tweetscriber.com/realgenekim/206 .

Como Randy Shoup ... Randy Shoup, correspondencia personal con Gene Kim, 2014.

Tom Limoncelli, coautor ... Tom Limoncelli, "Sí, realmente puedes trabajar desde HEAD" EverythingSysAdmin.com, 15 de marzo de 2014, http://everythingsysadmin.com/2014/03/yes-you-really-can-work-from-head.html .

Tom Limoncelli describe ... Tom Limoncelli, "Python es mejor que Perl6" EverythingSysAdmin.com, 10 de enero de 2011, http://everythingsysadmin.com/2011/01/python-is-better-than-perl6.html .

Google usó C ++ ... "¿Qué lenguajes de programación usa Google internamente?" Foro de Quora.com, accedido el 29 de mayo de 2016, https://www.quora.com/Which-programming-languages-does-Google-use-internally ; "¿Cuándo permitirá Google otros idiomas además de ¿Python, C ++, Java y Go se utilizarán para proyectos internos? ", Acceso al foro de Quora.com 29 de mayo de 2016, https://www.quora.com/When-will-Google-permit-languages-other-than-Python-C-Java-and-Go-to-be-be-be-be-used-for-internal-projects / answer / Neil-Kandalgaonkar .

En una presentación ... Ralph Loura, Olivier Jacques y Rafael García, "DOES15 - Ralph Loura, Olivier Jacques y Rafael García: rompiendo los paradigmas tradicionales de TI para ... " Video de YouTube, 31:07, publicado por DevOps Enterprise Summit, 16 de noviembre de 2015, https://www.youtube.com/watch?v=q9nNqqie_sM .

En muchas organizaciones ... Michael Rembetsy y Patrick McDonnell, "Continuamente Desplegando Cultura: Escalando la Cultura en Etsy - Velocity Europe 2012 ", Slideshare.net, publicado por Patrick McDonnell, 4 de octubre de 2012, http://www.slideshare.net/mcdonnps/continuously-deploying-culture-scaling-culture-at-etsy-14588485 .

En ese momento, Etsy ... Ibid.

En el próximo ... Ibid.

Del mismo modo, Dan McKinley ... Dan McKinley, "Por qué MongoDB nunca funcionó en Etsy" McFunley.com, 26 de diciembre de 2012, http://mcfunley.com/why-mongodb-never-worked-out-at-etsy .

CAPITULO 21

Uno de los ... "Kaizen", *Wikipedia* , modificado por última vez el 12 de mayo de 2016, https://en.wikipedia.org/wiki/Kaizen .

El Dr. Spear explica ... Spear, *The High-Velocity Edge* , cap. 8)

Página 397

Spear observa que ... Ibid.

Clanton describe, "Nosotros ... Mickman y Clanton," (Re) construyendo una Cultura de Ingeniería ".

Ravi Pandey, un ... Ravi Pandey, correspondencia personal con Gene Kim, 2015.

Clanton se expande sobre ... Mickman y Clanton, "(Re) construyendo una cultura de ingeniería".

Además de ... Hal Pomeranz, "Queue Inversion Week", *Righteous IT* , 12 de febrero de 2009, https://righteousit.wordpress.com/2009/02/12/queue-inversion-week/ .

Como Dr. Spear ... Spear, *The High-Velocity Edge* , cap. 3)

En una entrevista con Jessica ... Jessica Stillman, "Hack Days: Not Just for Facebookers" *Inc.* , 3 de febrero de 2012, http://www.inc.com/jessica-stillman/hack-days-not-just-for-

facebookers.html .

En 2008, Facebook ... AP, "Número de usuarios activos en Facebook a lo largo de los años", *Yahoo! Noticias* , 1 de mayo de 2013, https://www.yahoo.com/news/number-active-users-facebook-over-230449748.html? Ref = gs .

Durante un hack ... Haiping Zhao, "HipHop for PHP: Move Fast", publica en Haiping Zhao Página de Facebook, 2 de febrero de 2010, https://www.facebook.com/notes/facebook-ingeniería / hiphop-for-php-move-fast / 280583813919 .

En una entrevista con Cade ... Cade Metz, "Cómo tres hombres reconstruyeron la Fundación de Facebook ", *Wired* , 10 de junio de 2013, http://www.wired.com/wiredenterprise/2013/06/facebook-hhvm-saga/all/ .

Steve Farley, VP ... Steve Farley, correspondencia personal con Gene Kim, 5 de enero de 2016.

Karthik Gaekwad, quien ... "Agile 2013 Talk: Cómo DevOps lo cambia todo" Slideshare.net, publicado por Karthik Gaekwad, 7 de agosto de 2013, http://www.slideshare.net/karthequian/howdevops changeseverythingagile2013karthikgaekwad / .

Como Glenn O'Donnell ... Glenn O'Donnell, "DOES14 - Glenn O'Donnell - Forrester - Modern Los servicios exigen una cultura DevOps más allá de las aplicaciones ", video de YouTube, 12:20, publicado por DevOps Enterprise Summit 2014, 5 de noviembre de 2014, https://www.youtube.com/watch? v = pvPWKuO4_48 .

A partir de 2014 ... Nationwide, 2014 Annual Report, https://www.nationwide.com/about-us / nationwide-annual-report-2014.jsp .

Steve Farley, vicepresidente ... Steve Farley, correspondencia personal con Gene Kim, 2016.

Capital One, one ... "DOES15 - Tapabrata Pal - Banca en Innovación y DevOps", YouTube video, 32:57, publicado por DevOps Enterprise Summit, 4 de enero de 2016, https://www.youtube.com/watch?v=bbWFCKGhxOs .

Dr. Tapabrata Pal ... Tapabrata Pal, correspondencia personal con Gene Kim, 2015.

Target es la ... "Hoja informativa corporativa", sitio web de la compañía Target, accedido el 9 de junio de 2016, https://corporate.target.com/press/corporate .

Por cierto, el primero ... Evelijn Van Leeuwen y Kris Buytaert, "DOES15 - Evelijn Van Leeuwen y Kris Buytaert - Dando la vuelta a la portacontenedores ", video de YouTube, 30:28,

Página 398

publicado por DevOps Enterprise Summit, 21 de diciembre de 2015, https://www.youtube.com/watch?v=0GId4AMKvPc .

Clanton describe, "2015 ... Mickman y Clanton," (Re) construyendo una cultura de ingeniería ".

En Capital One ... "DOES15 - Tapabrata Pal - Banca en Innovación y DevOps", YouTube

video, 32:57, publicado por DevOps Enterprise Summit, 4 de enero de 2016, https://www.youtube.com/watch?v=bbWFCKGhxOs .

Bland explica que ... Bland, "DOES15 - Mike Bland - El dolor ha terminado, si lo deseas".

Aunque ellos ... Ibid.

Usaron varios ... Ibid.

Bland describió: "El ... Ibid.

Bland continúa: "Uno ... Ibid.

Como Bland describe ... Ibid.

Bland continúa: "Es ... Ibid.

Él continúa: "El ... Ibid.

Bland describe Fixits ... Mike Bland, "Fixits, o I Am the Walrus", Mike-Bland.com, 4 de octubre de 2011, https://mike-bland.com/2011/10/04/fixits.html .

Estos Fixits, como ... Ibid.

CAPITULO 22

Uno de los mejores ... James Wickett, "Atacando tuberías: la seguridad se encuentra con el continuo Entrega ", Slideshare.net, publicado por James Wickett, 11 de junio de 2014, http://www.slideshare.net/wickett/attacking-pipelinessecurity-meets-continuous-delivery .

James Wickett, uno ... Ibid.

Ideas similares fueron ... Tapabrata Pal, "DOES15 - Tapabrata Pal - Banca en Innovación y DevOps ", video de YouTube, 32:57, publicado por DevOps Enterprise Summit, 4 de enero de 2016, https://www.youtube.com/watch?v=bbWFCKGhxOs .

Justin Arbuckle, ex ... Justin Arbuckle, entrevista personal con Gene Kim, 2015.

Él continúa: "Por ... Ibid.

Esto ayudó a ... Snehal Antani, "IBM Innovate DevOps Keynote", video de YouTube, 47:57, publicado por IBM DevOps, 12 de junio de 2014, https://www.youtube.com/watch?v=s0M1P05-6Io .

En una presentación ... Nick Galbreath, "DevOpsSec: Aplicación de los principios de DevOps a la seguridad, DevOpsDays Austin 2012 ", Slideshare, publicado por Nick Galbreath, 12 de abril de 2012, http://www.slideshare.net/nickgsuperstar/devopssec-apply-devops-principles-to-security .

Además, afirma ... Ibid.

Además, deberíamos ... "Serie de hojas de trucos de OWASP", OWASP.org, última modificación en marzo 2, 2016, https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series .

La escala de ... Justin Collins, Alex Smolen y Neil Matatall, "Poniendo a tus Robots a Work V1.1 ", Slideshare.net, publicado por Neil Matatall, 24 de abril de 2012, http://www.slideshare.net/xplodersuv/sf-2013-robots/ .

A principios de 2009 ... "¿Qué pasa con las empresas que son pirateadas? Casos de la FTC, "Bomba gigante foro, publicado por SuicidalSnowman, julio de 2012, http://www.giantbomb.com/forums/off-topic-31 / what-going-to-companies-that-get-hack-ftc-case-540466 / .

En su anterior ... Collins, Smolen y Matatall, "Poniendo a tus robots a trabajar V1.1".

El primer gran ... Ingeniería de Twitter, "Hack Week @ Twitter", blog de Twitter, 25 de enero, 2012, https://blog.twitter.com/2012/hack-week-twitter .

Josh Corman observó ... Josh Corman y John Willis, "Inmutable Awesomeness - Josh Corman y John Willis en DevOps Enterprise Summit 2015 ", video de YouTube, 34:25, publicado por Sonatype, 21 de octubre de 2015, https://www.youtube.com/watch?v=-S8-lrm3iV4 .

En el 2014 ... Verizon, ″ Informe de investigaciones de violación de datos de 2014" (Verizon Enterprise Soluciones, 2014), https://dti.delaware.gov/pdfs/rp_Verizon-DBIR-2014_en_xg.pdf .

En 2015, esto ... "Informe del estado de la cadena de suministro de software 2015: obstáculos de velocidad ocultos en the Way to 'Continuous' ”(Fulton, MD: Sonatype, Inc, 2015), http://cdn2.hubspot.net/hubfs/1958393/White_Papers/2015_State_ of_the_Software_Supply_Chain_Report-.pdf? t = 1466775053631 .

La última estadística ... Dan Geer y Joshua Corman, "Casi demasiado grande para fracasar"; *login :: El Revista Usenix* , 39, no. 4 (agosto de 2014): 66-68, https://www.usenix.org/system/files/login/articles/15_geer_0.pdf .

Gobierno Federal de EE. UU. ... Wyatt Kash, "Nuevos detalles publicados sobre la propuesta de TI 2016 gasto ", *FedScoop*, 4 de febrero de 2015, http://fedscoop.com/what-top-agencies-would-gastos en proyectos en 2016 .

Como Mike Bland ... Bland, "DOES15 - Mike Bland - El dolor ha terminado, si lo deseas".

Además, Cloud.gov ... Mossadeq Zia, Gabriel Ramírez, Noah Kunin, “Cumplimiento Albañilería: Bulding una plataforma de gestión de riesgos, ladrillo por ladrillo ” , *18F* , 15 de abril de 2016, https://18f.gsa.gov/2016/04/15/compliance-masonry-buildling-a-risk-management-plataforma / .

Marcus Sachs, uno ... Marcus Sachs, correspondencia personal con Gene Kim, 2010.

Necesitamos ... "Mejores prácticas de configuración de VPC", blog Flux7, 23 de enero de 2014, http://blog.flux7.com/blogs/aws/vpc-best-configuration-practices .

En 2010, Nick ... Nick Galbreath, "Fraude Engineering, de Merchant Risk Council Annual Meeting 2012 ", Slideshare.net, publicado por Nick Galbreath, 3 de mayo de 2012, http://www.slideshare.net/nickgsuperstar/fraud-engineering .

De particular preocupación ... Nick Galbreath, "DevOpsSec: Aplicación de los principios de DevOps para Seguridad, DevOpsDays Austin 2012 ", Slideshare.net, publicado por Nick Galbreath, 12 de abril, 2013, http://www.slideshare.net/nickgsuperstar/devopssec-apply-devops-principles-to-seguridad .

Siempre fuimos ... Ibid.

Este fue un ridículo ... Ibid.

Como observó Galbreath ... Ibid.

Galbreath observó: "Uno ... Ibid.

Como Jonathan Claudius ... Jonathan Claudius, "Atacar servicios en la nube con código fuente" Speakerdeck.com, publicado por Jonathan Claudius, 16 de abril de 2013, https://speakerdeck.com/claudijd/attacking-cloud-services-with-source-code .

CAPITULO 23

ITIL define la utilidad ... *Axelos, ITIL Service Transition* (ITIL Lifecycle Suite) (Belfast, Irlanda: TSO, 2011), 48.

Se fundó Salesforce ... Reena Matthew y Dave Mangot, "DOES14 - Reena Mathew y Dave Mangot - Salesforce ", Slideshare.net, publicado por ITRevolution, 29 de octubre de 2014, http://www.slideshare.net/ITRevolution/does14-reena-matthew-and-dave-mangot-la fuerza de ventas .

Para 2007, el ... Dave Mangot y Karthik Rajan, "Agile.2013.effecting.a.dev ops.transformation.at.salesforce ", Slideshare.net, publicado por Dave Mangot, 12 de agosto de 2013, http://www.slideshare.net/dmangot/agile2013effectingadev-opstransformationatsalesforce .

Karthik Rajan, entonces ... Ibid.

En el 2014 ... Matthew y Mangot, "DOES14 - Salesforce".

Para Mangot y ... Ibid.

Además, notaron ... Ibid.

Bill Massie es ... Bill Massie, correspondencia personal con Gene Kim, 2014.

Debido al alcance ... "Glosario", sitio web del Consejo de Normas de Seguridad de PCI, accedido el 30 de mayo, 2016, https://www.pcisecuritystandards.org/pci_security/glossary .

Son revisión de código ... Consejo de Normas de Seguridad *PCI, Datos de la Industria de Tarjetas de Pago (PCI) Soportes de seguridad: requisitos y procedimientos de evaluación de seguridad, versión 3.1* (PCI Consejo de Normas de Seguridad, 2015), Sección 6.3.2. https://webcache.googleusercontent.com/search? q = caché: hpRe2COzzdAJ: https: //www.cisecuritystandards.org/documents/PCI_DSS_v3-1_SAQ_D_Merchant_rev1-1.docx + & cd = 2 & hl = es & ct = clnk & gl = us .

Para cumplir con esto ... Bill Massie, correspondencia personal con Gene Kim, 2014.

Massie observa que ... Ibid.

Como resultado ... Ibid.

Como Bill Shinn ... Bill Shinn, "DOES15 - Bill Shinn - ¡Pruébalo! La última milla para DevOps en Organizaciones reguladas ", Slideshare.net, publicado por ITRevolution, 20 de noviembre de 2015,

http: //www.slideshare.net/ITRevolution/does15-bill-shinn-prove-it-the-last-mile-for-organizaciones devops-in-reguladas.

Ayudando a grandes empresas ... Ibid.

Shinn señala: "Uno ... Ibid.

Página 401

"Eso estuvo bien ... Ibid.

Él explica: "En ... Ibid.

Shinn afirma que ... Ibid.

Shinn continúa: "Con ... Ibid.

Eso requiere derivar ... Ibid.

Shinn continúa: "Cómo ... Ibid.

Shinn da un ... Ibid.

Para ayudar a resolver ... James DeLuccia, Jeff Gallimore, Gene Kim y Byron Miller, *DevOps Kit de herramientas de defensa de auditoría* (Portland, OR: IT Revolution, 2015), http://itrevolution.com/devops-and-auditors-the-devops-audit-defense-toolkit .

Hizo el ... Mary Smith (un seudónimo), correspondencia personal con Gene Kim, 2013

Ella observó: ... Ibid., 2014.

CONCLUSIÓN

Como Jesse Robbins ... "Hacking Culture at VelocityConf", Slideshare.net, publicado por Jesse Robbins, 28 de junio de 2012, http://www.slideshare.net/jesserobbins/hacking-culture-at-velocidadconf .

APÉNDICE

El movimiento Lean comenzó ... Ries, *The Lean Startup.*

Un director clave ... Kent Beck et al., "Doce principios del software ágil" AgileManifesto.org, 2001, http://agilemanifesto.org/principles.html .

Sobre la base de ... Humilde y Farley, *Entrega continua* .

Esta idea era ... Fitz, "Despliegue continuo en IMVU".

Toyota Kata describe ... Rother, *Toyota Kata,* Introducción.

Su conclusión fue ... Ibid ...

En 2011, Eric ... Ries, *The Lean Startup.*

En The Phoenix ... Kim, Behr y Spafford, *The Phoenix Project,* 365.
Mito 1: "Humanos ... Denis Besnard y Erik Hollnagel, *algunos mitos sobre la industria Seguridad* (París, Centre De Recherche Sur Les Risques Et Les Crises Mines, 2012), 3, http://gswong.com/?wpfb_dl=31 .

Mito 2: "Sistemas ... Ibid., 4.

Mito 3: "Seguridad ... Ibid., 6.

Mito 4: "Accidente ... Ibid., 8.

Mito 5: "Accidente ... Ibid., 9.

Mito 6: Seguridad ... Ibid., 11.

Página 402

Más bien, cuando el ... John Shook, "Cinco piezas faltantes en su trabajo estandarizado (Parte 3 de 3) ", Lean.org, 27 de octubre de 2009, http://www.lean.org/shook/DisplayObject.cfm?o=1321 .

Es hora de resolver ... "Retrospectiva posterior al evento - Parte 1", Rally Blogs, consultado el 31 de mayo de 2016, https://www.rallydev.com/blog/engineering/post-event-retrospective-part-i .

Bethany Macri, de ... "Morgue: Ayudando a comprender mejor los eventos mediante la construcción de una publicación Mortem Tool - Bethany Macri ", video de Vimeo, 33:34, publicado por info@devopsdays.org, 18 de octubre de 2013, http://vimeo.com/77206751 .

Estas discusiones tienen ... Cockcroft, Hicks y Orzell, "Lecciones aprendidas por Netflix".

Desde entonces, Caos ... Ibid.

Lenny Rachitsky escribió ... Lenny Rachitsky, "7 claves para una salud pública exitosa Dashboard ", *Transparent Uptime* , 1 de diciembre de 2008, http://www.transparentuptime.com/2008/11/rules-for-successful-public-health.html .

Índice

Sín
E

Nota: Las cifras se indican con *f* ; las notas al pie se indican con *n*

SÍMBOLOS

% C / A, 11 , 65

NÚMEROS

2PT. *Ver equipo de* dos pizzas

Equipo 18F, 325–326

Informe del estado de DevOps 2013, 159-160

UN

AAS *Ver* Amazon Auto Scaling
Adams, Keith, 302

Ágil

Infraestructura y movimiento de velocidad, 5 5

Movimiento de Infraestructura, 354

Manifiesto, 4–5

Movimiento, 354

principios, xxii – xxiii

Aisin Seiki Global, 43-44

Alcoa 41–42 , 279

Algra, Ingrid, 305 *n*

Allspaw, John

Infraestructura ágil y movimiento de velocidad, 5

lanzamientos oscuros, 173-174

fallas de implementación, 251–252

cultura de aprendizaje, 273–274

métricas de producción, 204

Movimiento de velocidad, 354

Allstate 66

Página 404

Amazonas

entrega continua, 176

despliegues por día, xxxiv *n*

arquitectura evolutiva, 184-185

organización orientada al mercado, 81

Arquitectura orientada a Servicios, 90-91

Amazon Auto Scaling, 221–222

Amazon AWS

cumplimiento en entornos regulados, 342–344

Resiliencia, 271–273 ,281–282

interrupción del servicio, 271 *n*

Andon cord
consecuencias de no tirar, 140
ilustrado, 361 *f*
pululando, 31 , 32
virtual, 138-140
y paros laborales, 360–361
Antani, Snehal, 314
APIs
habilitación, 91–93
API de funciones, 245
interacciones de servicio utilizando, 89
versionado, 185
registro de aplicaciones, 201-203, 201 *n*
patrones de lanzamiento basados en aplicaciones, 171–175
Arbuckle, Justin, 290, 314
arquitectura evolutiva
Estudio de caso de Amazon, 184-185
arquetipos arquitectónicos, 183*F*
Estudio de caso de Blackboard Learn, 186–189 , 187*f* ,188 *f*
repositorio de código, 187 *f* , 188 *f*
funcionalidad de desacoplamiento, 186
descripción de, 179-180
servicios inmutables, 185
arquitectura vagamente acoplada, 181–182

Página 405

monolitos vs microservicios, 182-185
Segunda ley de termodinámica arquitectónica, 180-181
Arquitectura orientada a Servicios, 182
patrón de aplicación de estrangulador, 180, 185-189
arquitectura estrechamente acoplada, 180-181, 185
API versionadas, 185

servicios versionados, 185
arquitectura, ligeramente acoplada, 89–93 , 181–182 , 254–255

arquitectura, monitoreo, 198-199

arquitectura, orientada a servicios, 89 , 90–91 , 182

Ashman, David, 187

ATDD *Ver* desarrollo, aceptación basada en pruebas

Atwood, Jeff, 147, 259–260

Austin, Jim 215-216

proceso de construcción de entorno automatizado

activos para registrar en el repositorio de control de versiones, 116

sistemas de configuración automatizada, 118

beneficios de la automatización, 114–115

mecanismos de construcción comunes, 113

papel crítico del control de versiones, 117

consistencia ambiental, 118

Desarrollo del medio ambiente bajo demanda, 113–115

reconstrucción del entorno versus reparación, 118

entornos almacenados en el control de versiones, 115–116

infraestructura inmutable, 119

metadatos 115

nueva definición de desarrollo terminado, 119-121

desarrollo rápido del entorno, 112

repositorio de control de versiones compartido, 115–118

sprints 119-120

Estandarización, 114

pruebas, 113

entornos de prueba, 113

usos de la automatización, 114

control de versiones como predictor del desempeño organizacional, 117

sistemas de control de versiones, 115–118

conjunto de pruebas de validación automatizada
desarrollo impulsado por pruebas de aceptación, 134–135
pruebas de aceptación, 131 , 132
herramientas de análisis, 138
automatización de pruebas manuales, 135-136
herramientas de gestión de configuración de código, 138
validación del entorno, 137–138
detección de errores, 132-133
prueba rápida, 132, 133-134
comentarios, 130
construcciones verdes, 129–130
prueba ideal vs no ideal, 133 *f*
pruebas de integración, 131 ,132
pruebas de requisitos no funcionales, 137–138
pruebas de rendimiento, 136-137
tipos de prueba, 130–131
desarrollo basado en pruebas, 134-135
prueba en paralelo, 133-134, 134*F*
pruebas unitarias, 130–131 , 132–133
prueba no confiable, 135
automatización. *Ver* proceso de creación de entorno automatizado; automatización de procesos de despliegue; pruebas automatizadas
si
Panadero, bill, 118
Barnes & Noble, 51
tamaños de lote
despliegue continuo, 20
gestión de errores, 19
grande, 19–20
flujo de una pieza, 19, 20
pequeño, 18-20
estrategia de lotes pequeños, 19
pequeño vs grande, 20*F*
Bazaarvoice 97 *n* ,149-151

Beck, Kent, 134 , 154

Beedle, Mike, 102*norte*

Behr, Kevin, 195, 203 *n*

Besnard, Denis 359–360

Betz, Charles, 180-181, 313 *n*

Big Fish Games, 95–97

bimodal IT, 56

Blackboard Learn

caso de estudio, 187*f* ,188*F*

Perl, 187

patrón de aplicación de estrangulador, 186–189

autopsias sin culpa

contramedidas, 276

objetivos de 274–275

resultados de, 275–276

publicidad, 277–278

agenda de muestra, 362–364

partes interesadas presentes, 275

tiempo de actividad transparente, 277 *n*

Bland, Mike, 123–126 , 124 *n* , 306–307 ,325

En blanco, Steve, 355

Blankenship, Ed, 210

bombardeo aéreo

metas, 301

mejora, 299

kaizen, 299

Blockbuster, 51

patrón de despliegue azul-verde

despliegue, 166-169 , 166 *f* , 167*norte*

Farley, David, 168-169

liberaciones de bajo riesgo, 166-169 , 166 *f*

Ruby on Rails, 167*norte*

BMW, 67

Bohmer, Richard MJ, 279

Booch, Grady, 6 *n*

Página 408

Fronteras, 51

Boubez, Toufic, 219, 224–226

Bouwman, Jan-Joost, 305 *n*

Guardafrenos, 322 , 322*F*

servicios brownfield. *Ver* servicios, brownfield

Bloques de construcción, 188-189

construir-medir-aprender ciclo, 355

organizaciones burocráticas, 39

Burgess, Mark, 6 *n*

lógica de negocios

  cambios a, 78, 79

  coordinar cambios a, 79

  pasar a la capa de aplicación, 79

gerente de relaciones comerciales, 96

Buytaert, Kris, 305 *n*

C

C ++

  eBay 179*n* ,182

  Facebook, 153*n* , 175 ,302

  Google 296*norte*

  Servidor web de Google, 123

Cagan, Marty, 70

Campbell-Pretty, Em, 111-112

Canahuati, Pedro, 85

patrón de liberación canaria, 153 *n* , 169-171 , 170 *f* , 170*norte*

pruebas canarias, 153

Capitol One 304-306

estudios de caso

  Amazonas, 184-185

Amazon AWS, 271–273, 344–345

técnicas de detección de anomalías, 224–226

Sistemas de cajeros automáticos, 344–345

Bazaarvoice 149-151

Big Fish Games, 95-97

Blackboard Learn, 186–189, 187*f* ,188*F*

Página 409

Capitol One, 304-306

CSG International, 157–159

Dixons Retail, 168-169

Etsy, 77–80, 162-164, 297–298 ,328–330 , 339–341

Facebook, 153-155, 174-175

Gobierno federal, 325–326

Google 237–239 , 257–258

Servidor web de Google, 123–126

HP, 144–146

Intuit 241–248

LinkedIn, 71–73 ,207-208

Seguro a nivel nacional, 304-306

Netflix 215-216, 221–222 ,271–273

Nordstrom 51–55 ,61–62

Pivotal Labs, 260–261

Right Media, 227–229

Salesforce.com, 337–338

Objetivo, 91–93, 299–300, 304-306

Twitter, 320–323

Yahoo! Respuestas246–248

Chacon, Scott, 249

Chakrabarti, Arup, 232

cambiar los procesos de aprobación. *Ver también* revisiones de código

caso de estudio, 249–251

cambiar las juntas asesoras, 253

cambiar las fallas de control, 252–253

cambio congela, 258-259

revisión de tiempos de entrega, 258*F*

revisiones de código, 255–258

coordinación y programación de cambios, 254–255

pensamiento contrafactual, 251, 251*norte*

cortar los procesos burocráticos, 263–264

peligros de 251–252

transferencia de correo electrónico, 257

roles de ingeniero, 259

GitHub Flow, 250

Estudio de caso de Google, 257–258

pautas para revisiones de código, 256

en una arquitectura débilmente acoplada, 254–255

prueba manual, 258–259

sobre el hombro, 256

programación en pareja, 256, 259–263

revisiones por pares, 249–251, 253, 254*f* ,255–258

Estudio de caso de Pivotal Labs, 260–261

solicitudes de extracción, 250, 250*f* , 261–263

revisar los pasos, 250-251

lotes pequeños, 255–256

desarrollo basado en pruebas, 259–260

asistido por herramienta, 257

controles de cambio tradicionales, 252–254

tipos de revisiones de código, 256–257

Mono del caos, 272–273, 281–282, 364

salas de chat, 74

ChatOps, 287–289

Chuvakin, Anton A., 202

CI. *Ver* integración continua

Clanton, Ross, 263, 299–300 , 305

Claudio, Jonathan, 330

Clemm, Josh 71 , 72

Cloud.gov, 325–326

sistema inmune en racimo, 171*norte*

patrón de liberación del sistema inmune en racimo, 169 , 170-171

entrenando kata, 45

Cockcoft, Adrian, 82*n* ,200 , 263

código

comete, 148

herramientas de gestión de configuración, 138

despliegue, 22 ,160-162

automatización del proceso de despliegue, 160

cambios en el proceso de implementación, 154

Página 411

infraestructura como, 6 *n*

fusionándose, 143–144

migración, 117*norte*

embalaje, 128

repositorios, 187 *f* ,188*f* ,290–292 , 315–317

reutilizar, 289–290

firma, 319–320

integridad del código fuente, 319–320

confirmaciones de código, 148

Revisiones de código. *Consulte también los* procesos de aprobación de cambios

revisión de tiempos de entrega, 258*F*

transferencia de correo electrónico, 257

Estudio de caso de Google, 257–258

pautas para, 256

sobre el hombro, 256
programación en pareja, 256
lotes pequeños, 255–256
asistido por herramienta, 257
tipos de 256-257
conocimiento colectivo, 42–43
conformidad
documentación y prueba de auditoría y cumplimiento, 341–345
objetivos de cumplimiento normativo, 235–236
seguridad y cumplimiento y procesos de aprobación de cambios, 333–335
herramientas de gestión de configuración, 116*norte*
restricciones
cuellos de botella, 22
despliegue de código, 22
creación del entorno, 22
arquitectura demasiado apretada, 23
prueba de configuración y ejecución, 22–23
continua experimentación y aprendizaje, 37-46
entrega continua
tubería de despliegue, 127-129 ,127 *f*
Google 176

Página 412

liberaciones de bajo riesgo, 175-177
Movimiento de entrega continua, 5–6, 354
despliegue continuo, 6 6, 20, 175-177
integración continua
caso de estudio, 149-151
fusión de código, 143–144
descripción de, 144-146
Dev vs DevOps, 126*norte*
confirmaciones de código frecuentes, 148

commits cerrados, 148
problemas de integración, 143
desarrollo de lotes grandes, 147–148
y prácticas de desarrollo basadas en troncales, 148-151
y control de versiones, 148-149
Convergencia de DevOps, 353–356
Conway, Melvin, 77
Ley de Conway 77–78, 88
Cocinero, Scott, 242
conflicto crónico central, xxiv – xxvi, xxv *n*
nube de conflicto central, 356–357 , 356 *f*
Corman, Josh, 313 , 323
costo de demora, 213*norte*
Software COTS, 361
Cox, Jason, 87, 99–100 , 263
CSG International
servicios brownfield, 56
entrenamiento cruzado, 86–87
implementaciones diarias, 157-159
Cunningham, Ward, 148
re
lanzamientos oscuros, 173–175
tableros de instrumentos, 207*norte*
conjuntos de datos *Ver* telemetría
Debois, Patrick, 5
ingeniero de lanzamiento dedicado, 96

Página 413

DeGrandis, Dominica, 18
Dekker, Sidney
solo cultura, 273
cultura de seguridad, 28 , 38

escaneo de dependencia
Java, 319
Ruby on Rails, 319
Desplegador, 163-164 ,163 *f*
despliegue
autoservicio automatizado, 159-160
patrón azul verdoso, 166–169 ,166*f* ,167*norte*
cambio, 124
código, 22, 154 ,160-162
consistencia, 156
continuo, 20 , 175-177
diario, 157-159
desacoplamiento de lanzamientos, 164–175
definido, 164
bajo demanda, 165
rápido, 161, 161*F*
fluir, 156
cuestiones, 78
tiempo de espera, 8-11, 9 *f* ,165
haciendo más seguro, 229–230
superposición de actividades de despliegue de producción, 213
paso, 154
requisitos de tubería, 156-157
automatización de procesos, 155-164, 159 *f*
desarrollador de autoservicio, 162–164
velocidad y éxito, 79
herramienta, 163-164, 163*F*
plazo de ejecución
diseño y desarrollo, 8
tiempo de entrega vs tiempo de procesamiento, 9-10, 9 *f*
Lean Manufacturing, 9

Desarrollo de productos ajustados, 8

largo, 10, 10*f* ,165

corto, 10-11, 11*F*

pruebas y operaciones, 9 9

flujo de trabajo, 9 9

tubería de despliegue

Descompostura, 138-140

contenedores en, 128 *n*

entrega continua, 127-129, 127*F*

y seguridad de la información, 330–331

protección de tubería de despliegue

Estudio de caso de Amazon AWS, 342–344

documentación y prueba de auditoría y cumplimiento, 341–345

categorías de cambios, 334–335, 334*norte*

cumplimiento en entornos regulados, 341–345

pruebas destructivas, 338

Estudio de caso de Etsy, 339–341

cambios normales 334 , 336–338

telemetría de producción para sistemas ATM, 344–345

Caso de estudio de Salesforce, 337–338

seguridad y cumplimiento y procesos de aprobación de cambios, 333–335

separación de funciones, 338–341

cambios estándar, 334 ,335–336

cambios urgentes, 334–335

automatización del proceso de despliegue

implementaciones automáticas de autoservicio, 159–160

Automatización de pasos manuales, 155–156

implementación de código como parte de la canalización de implementación, 160-162

procesos de promoción de código, 160

Estudio de caso de CSG International, 157–159

consistencia de implementación, 156

flujo de despliegue, 156

requisitos de la tubería de despliegue, 156–157

consistencia ambiental, 157, 158

Estudio de caso de Etsy, 162-164

implementaciones rápidas, 161 ,161 *f*

reducción del tiempo de entrega, 156

MTTR 158, 159*f* ,161*F*

documentación de proceso, 155

disminución del incidente de producción, 158, 159 *f*

implementación de desarrollador de autoservicio, 162-164

Equipo de operaciones compartidas, 157

prueba de humo, 156, 163

implementación vs lanzamientos, 164–175

Desarrollador en entornos de producción

activos para registrar en el repositorio de control de versiones, 116

sistemas de configuración automatizada, 118

beneficios de la automatización, 114–115

mecanismos de construcción comunes, 113

papel crítico del control de versiones, 117

consistencia ambiental, 118

Desarrollo del medio ambiente bajo demanda, 113–115

reconstrucción del entorno versus reparación, 118

entornos almacenados en el control de versiones, 115–116

infraestructura inmutable, 119

metadatos 115

nueva definición de desarrollo terminado, 119-121

desarrollo rápido del entorno, 112

repositorio de control de versiones compartido, 115–118

sprints 119-120

Estandarización, 114

pruebas, 113

entornos de prueba, 113

usos de la automatización, 114

control de versiones como predictor del desempeño organizacional, 117

sistemas de control de versiones, 115–118

Desarrollo. *Ver* entradas en Dev

desarrollo, aceptación basada en pruebas, 134-135

desarrollo, probado

pruebas automatizadas, 134–135, 293

Página 416

manejo de densidad de defectos, 135 *n*

y emisiones de bajo riesgo, 154

y programación de pares, 259–260

prueba antes de escribir el código, 9 9*norte*

desarrollo, basado en troncales, 143-151

desarrollo, cascada, 5

DevOps

Infraestructura ágil y movimiento de velocidad, 5

Manifiesto ágil, 4–5

valor comercial de, xxxii – xxxiii

Movimiento de entrega continua, 5–6

definida, 4

DevOpsDays, 5

espiral descendente adentro xxx – xxxii

ética de, xxix – xxxv

historia de, 3–6

Movimiento Lean, 4

revolución, xxii

compromiso de equipo, 101 *n*

Las tres formas 11-12

Movimiento Toyota Kata, 6 6

Mitos de DevOps

LAMPARAS, xvi

MySQL, xvi

PHP, xvi

Transformación de DevOps

bimodal IT, 56

salas de chat, 74

expandiendo DevOps, 58-59

servicios greenfield vs brownfield, 54–56

aprovechando a los innovadores, 57-58

Estudio de caso de LinkedIn, 71–73

haciendo visible el trabajo, 73

gestión de deuda técnica, 69–71

fases de iniciativas, 59

Página 417

ambiente de comunicación rápida, 74

reforzando el comportamiento deseado, 73–74

herramientas compartidas, 73–74

cola de trabajo compartido, 73–74

sistemas de compromiso, 56–57

sistemas de registro, 56

deuda técnica, 69-71 ,70 *f*

curva de adopción de tecnología, 58 *f*

equipo de transformación, 66-73

DevOpsDays, 5 5, 305*norte*

Dignan, Larry, 91

Disney 87, 99–100

Dixons Retail, 168-169

documentación

pruebas automatizadas como, 293

proceso, 155

espiral descendente, xxvi – xxviii, xxx – xxxii, 357 ,357*F*

Drucker, Peter, 60 60

Dweck, Carol, 87

mi

eBay, 70 ,179-180, 179 *n* ,182

sitios de comercio electrónico

técnicas de detección de anomalías, 224–226

seguridad de la aplicación, 318

fuentes de métricas, 210

Nordstrom 51-55

Objetivo, 91–93

Edmondson, Amy C. 278, 279

Edwards, Damon, 64, 97

Puntuación neta del promotor del empleado, xxxiii *n*

consistencia ambiental, 157 , 158

seguridad ambiental, 324–326, 324 *n*

ambientes

consistencia, 157, 158

restricciones de creación, 22

Página 418

definición de, 113*norte*

ambiente de comunicación rápida, 74

validación, 137–138

entornos, construcción automatizada

activos para registrar en el repositorio de control de versiones, 116

sistemas de configuración automatizada, 118

beneficios de la automatización, 114–115

mecanismos de construcción comunes, 113

papel crítico del control de versiones, 117

consistencia ambiental, 118

Desarrollo del medio ambiente bajo demanda, 113–115

reconstrucción del entorno versus reparación, 118

entornos almacenados en el control de versiones, 115–116

infraestructura inmutable, 119

metadatos 115

nueva definición de desarrollo terminado, 119-121

desarrollo rápido del entorno, 112

repositorio de control de versiones compartido, 115–118

sprints 119-120

Estandarización, 114

pruebas, 113

entornos de prueba, 113

usos de la automatización, 114

control de versiones como predictor del desempeño organizacional, 117

sistemas de control de versiones, 115–118

ambientes, tipo producción

activos para registrar en el repositorio de control de versiones, 116

sistemas de configuración automatizada, 118

beneficios de la automatización, 114–115

mecanismos de construcción comunes, 113

papel crítico del control de versiones, 117

desarrollo bajo demanda, 113-115

consistencia ambiental, 118

Desarrollo del medio ambiente bajo demanda, 113–115

reconstrucción del entorno versus reparación, 118

entornos almacenados en el control de versiones, 115–116

infraestructura inmutable, 119

metadatos 115

nueva definición de desarrollo terminado, 119-121

desarrollo rápido del entorno, 112

repositorio de control de versiones compartido, 115–118

sprints 119-120

Estandarización, 114

pruebas, 113

entornos de prueba, 113

usos de la automatización, 114

control de versiones como predictor del desempeño organizacional, 117

sistemas de control de versiones, 115–118
errores
detección, 28 , 114 *n* ,132-133
gestión, 19
Etsy
servicios brownfield, 56
caso de estudio, 77–80, 162-164
despliegue de código, 162 *n*
Ops designadas, 100-101
Transformación DevOps, 51
organizaciones orientadas a la funcionalidad, 84
LAMP stack en, 196
biblioteca de métricas, 204-206
Morgue, 277–278
MySQL en, 196, 297–298
Aprendizaje organizacional, 40
PHP, 196, 297 , 297 *n*
monitoreo de producción, 196–198
lenguajes de programación utilizados, 297*norte*
publicidad de autopsias, 277
Pitón, 297, 297*norte*
telemetría de seguridad, 328–330
separación de funciones, 339–341

Page 420

Sprouter 78-80, 88
estandarización de la pila tecnológica, 297–298
Evans, Eric J., 89
Evans, Jason, 302
Programación extrema, 134 , 154
F
Facebook

C ++, 153*n* ,175 ,302
patrón de liberación canario, 170
caso de estudio, 153-155
lanzamientos oscuros, 174–175
base de código front-end, 153 *n*
Portero, 172*norte*
JavaScript, 175
PHP, 153 *n* , 154, 175 , 302
dolor compartido, 85
deuda técnica, 302
Farley, David
pruebas automatizadas, 126–127
patrón de despliegue azul-verde, 168-169
entrega continua, 175-176, 354
Movimiento de entrega continua, 5–6
integración continua, 126*norte*
infraestructura como código, 6*norte*
Farley, Steve 303, 305
Farrall, Paul, 95-97
Experimentación de ciclo de liberación rápida
caso de estudio, 246–248
API de funciones, 245
historia de, 243
integración en la planificación de funciones, 245–248
integración en pruebas de características, 244–245
integrando en lanzamiento, 245
resultados de, 244
investigación de usuarios, 244–245

Yahoo! Respuestas246–248
prueba rápida 132, 133-134

alternar funciones, 171-173 , 172 *n* , 175, 229–230
caracteristicas
consecuencias de nuevo, 57
extra, 24
planificación, 245–248
pruebas, 244–245
Agencias del gobierno federal, 325–326
realimentación
procesos automatizados de construcción, integración y prueba, 30
prueba automatizada, 126 , 130
detección de errores, 28
controles de calidad ineficaces, 32–34
mecanismos de telemetría de producción, 229
Operaciones y resultados orientados al mercado, 103
optimización para centros de trabajo posteriores , 34–35
prevención de problemas, 30
visibilidad del problema, 29-30
telemetría de producción, 229
Automatización de control de calidad, 33–34
seguridad en sistemas complejos, 27-29
pululando, 30-32
bucles de avance, 29, 30
Fernández, Roberto, 59 , 80
primeras historias, 360 *f*
Fitz, Tim
entrega continua, 354
despliegue continuo, 6, 175-177
patrón expandir / contraer, 168*norte*
arreglar adelante, 230
mentalidad fija, 87
Flickr, 173-174
Forsgren, Nicole, 220
Fowler, Martin, 132, 185 ,186

sol

Gaekwad, Karthik, 303

Galbreath, Nick, 227–229 ,328–330

Game Days, 282–284

Ganglia, 196

García, Rafael, 297

Portero, 172*n* ,175

Gauntlt, 313 ,317

Planta de General Motors Fremont, 29, 31 ,37

organizaciones generativas, 39-40

GitHub

  organizaciones orientadas a la funcionalidad, 84

  GitHub Flow, 250

  conocimiento organizacional, 287–289

  revisiones por pares, 249–251

GitHub Flow, 250

establecimiento de objetivos, 68

Google

  prueba automatizada, 125 *n*

  C ++, 296*norte*

  revisiones de código, 257–258

  entrega continua, 176

  Programa de recuperación de desastres (DiRT), 284

  organizaciones orientadas a la funcionalidad, 84

  síndrome impostor, 124 *n*

  Java, 296 *n*

  JavaScript, 296 *n*

  revisiones de preparación para lanzamiento y entrega, 237–239

  servicio de producción, 234

  lenguajes de programación utilizados, 297*norte*

  publicidad de autopsias, 277

  Pitón, 296

  Arquitectura orientada a Servicios, 90

  repositorio de código fuente, 291–292

Motor de aplicaciones de Google, 139

Google Cloud Datastore, 181 *f* , 182

Servidor web de Google

Cordón Andon, 138

pruebas automatizadas, 123-126

C ++, 123

caso de estudio, 123-126

Fixit Grouplet, 307

Prueba de Grouplet, 124–126 , 306–307

Govindarajan, Vijay, 66

Grafana 204 , 224

Grafito, 196, 204, 224

Gray, Jim, 184

servicios greenfield. *Ver* servicios, greenfield

mentalidad de crecimiento, 87

Gruver, Gary

pruebas automatizadas, 123 , 136

integración continua, 144-146

controles de calidad ineficaces, 34

Gupta, Prachi, 207-208

GWS *Ver* servidor web de Google

H

Hammant, Paul, 186*norte*

Hammond, Paul, 5, 354

Revisión de preparación para la transferencia , 237–239 , 239 *f*

traspasos

peligros de 358–359

pérdida de conocimiento, 21

reduciendo el tamaño del lote, 21

gestión del flujo de trabajo, 21

fase de endurecimiento, 53 *n*

Hendrickson, Elisabeth, 30, 135 , 260–261

heroicos, 10 , 25 , 25 *n*

cultura de alta confianza, 37-38

Compilador HipHop, 302

Hodge, Victoria J., 215–216

Hollnagel, Erik, 359–360

HP, 144-146

HRR. *Ver* Revisión de preparación para la transferencia

Humilde, Jez

  pruebas automatizadas, 126–127

  entrega continua, 175-176

  Movimiento de entrega continua, 5–6

  integración continua, 126*norte*

  peligros de los procesos de aprobación de cambios, 252

  arquitectura evolutiva, 179

  infraestructura como código, 6*norte*

  investigación de usuarios, 244

desarrollo impulsado por hipótesis, 241–248

yo

ICHT, 339–341

Imbriaco, Mark, 288

síndrome del impostor, 124 *n*

bombardeo de mejora

  aprendizaje organizacional y mejoramiento, 299

  Lanza Steven 299

  Sistema de producción de Toyota, 299

ejemplos de objetivos de mejora, 68

kata de mejora, 6 6, 355

seguridad industrial, 359–360

radiador de información, 206–208

seguridad de información

Equipo 18F, 325–326

seguridad de la aplicación, 318–323

pruebas de seguridad automatizadas, 318*F*

malos caminos, 318

Guardafrenos, 322, 322 *f*

construir imágenes, 317

Cloud.gov, 325–326

firma de código, 319–320

creando telemetría de seguridad, 327–330

Página 425

violaciones de datos, 323–324

y seguimiento de defectos y autopsias, 315

escaneo de dependencia, 319

y la tubería de despliegue, 317–318 , 330–331

análisis dinámico, 319

seguridad ambiental, 324–326

Estudio de caso de Etsy, 328–330

Estudio de caso del gobierno federal, 325–326

Gauntlt, 313 , 317

Grafito, 329*F*

caminos felices 318

integración en la producción de telemetría, 326–327

invitando a InfoSec a demostraciones de productos, 314

Java, 324

Metasploit, 325

Nmap, 325

controles de seguridad preventivos, 315–317

Ruby on Rails, 322

DevOps robusto, 313

caminos tristes, 318

bibliotecas de seguridad, 316

repositorios y servicios de código compartido, 315–317

seguridad de la cadena de suministro de software, 323–324

integridad del código fuente, 319–320

Intentos de inyección SQL, 329, 329*F*

análisis estático, 319 ,320–323

Estudio de caso de Twitter, 320–323

flujo de valor, 63

Infosec. *Ver* seguridad de la información

infraestructura como código, 6 6*norte*

métricas de infraestructura, 213 *n*

InGraphs, 208

entorno de desarrollo integrado, 128*norte*

integración, 120 *n*

Intuit, 241–248

Página 426

longitud de iteración, 68

ITIL, 116 *n* , 231*n* , 253, 333, 334 *n*

ITIL CMDB, 212 , 212 *n*

J

Jacob, Adán, 6 *n*

Jacques, Olivier, 297

Java

automatización, 127

Bazaarvoice 149

escaneo de dependencia, 319

eBay 179*norte*

Google 296*norte*

seguridad de la información, 324

LinkedIn, 71

infraestructura de registro, 201*norte*

ORM, 79*norte*

métricas de producción, 204

enhebrar bibliotecas, 292

JavaScript

Facebook, 175

Google 296*norte*

telemetría de producción, 209

Jones, Daniel T. 19

solo cultura, 38

K

Kalantzis, Christos, 281–282

tableros kanban

y el movimiento Lean, 4 4

compartir entre Ops y Dev, 104

gestión del flujo de trabajo, 16–17, 16 *f*

Kanies, Luke, 6 6*norte*

Kastner, Erik, 163-164

Kim, Gene 195, 203 *n* , 233, 252, 313

Kissler, Courtney, 51–54 ,61–62

Knight Capital, 251–252

Página 427

el intercambio de conocimientos, 42–43

Kohavi, Ronny, 244

Krishnan, Kripa, 284

L

Pila LAMP

Mitos de DevOps, xvi

en Etsy 196

Lauderbach, John, 88*norte*

Revisión de preparación de lanzamiento, 238–239

tiempo de espera

revisión de cambio, 258 *f*

despliegue, 8-11 , 9*f* ,165

y el movimiento Lean, 353

reducción, 156

Lean Manufacturing

plazo de implementación, 8–11

organizaciones orientadas a la funcionalidad, 84

flujo de valor de fabricación, 7–8

flujo de valor tecnológico, 8-11

Las tres formas 11-12

Movimiento Lean, 4, 353

Principios lean, xxii

Desarrollo de productos lean, 9 9

Movimiento Lean Startup, 355

Movimiento Lean UX, 355

cultura de aprendizaje

Estudio de caso de Amazon AWS, 271–273

amplificación de señales de falla débil, 279–280

mala teoría de la manzana, 273

autopsias sin culpa, 274–276

Mono del caos, 272–273, 281–282

Días de juego, 282–284

inyectando fallas en el entorno de producción, 281–282 , 282 *n*

cultura justa, 273–274

líderes y, 44-46

Página 428

Morgue, 277–278

Estudio de caso de Netflix, 271–273

publicidad de autopsias, 277–278

redefiniendo el fracaso, 280–281

ensayos fallidos, 282–284

Resiliencia, 281–282
ingeniería de resiliencia, 282–283
La tercera vía, 44–46
y Toyota Kata, 45
y el sistema de producción de Toyota, 45
Lesiecki, Nick, 306–307
Letuchy, Eugene, 174–175
Lightbody, Patrick, 231
Limoncelli, Tom, 218 ,238–239, 292, 296–297
LinkedIn
caso de estudio, 71–73
Java, 71
Operación Inversión, 72
Oráculo, 71
métricas de autoservicio, 207-208
Pequeño Christopher xxvii ,73
Inicio sesión, 201-203, 201*norte*
infraestructura de registro
Java, 201*norte*
Ruby on Rails, 201 *n*
Loura, Ralph, 297
Amor, Paul, 313.
lanzamientos de bajo riesgo
patrones de lanzamiento basados en aplicaciones, 165-166 ,171-175
implementaciones automáticas de autoservicio, 159–160
Automatización de pasos manuales, 155–156
patrón de despliegue azul-verde, 166-169, 166*F*
patrón de liberación canaria, 169-171 , 170 *f*
pruebas canarias, 153
patrón de liberación del sistema inmunitario del grupo, 169 ,170-171

implementación de código como parte de la canalización de implementación, 160-162
cambios en el proceso de implementación del código, 154

procesos de promoción de código, 160

entrega continua, 175-177

despliegue continuo, 175–177

Estudio de caso de CSG International, 157–159

lanzamientos oscuros, 173–175

cambios en la base de datos, 167-168, 168 *n*

desacoplamiento de despliegue de versiones, 164–175

consistencia de implementación, 156

despliegue definido, 164

flujo de despliegue, 156

plazo de ejecución de implementación, 165

despliegue bajo demanda, 165

ritmo de despliegue, 154

requisitos de la tubería de despliegue, 156–157

automatización del proceso de despliegue, 155–164

herramienta de despliegue, 163-164 ,163 *f*

Estudio de caso de Dixons Retail, 168-169

consistencia ambiental, 157, 158

patrones de liberación basados en el entorno, 165 , 166–171

Estudio de caso de Etsy, 162-164

arquitectura evolutiva, 179-189

Estudio de caso de Facebook, 174–175

implementaciones rápidas, 161 ,161 *f*

función alterna, 171-173 ,175

reducción del tiempo de entrega, 156

MTTR 158, 159*f* ,161*F*

degradación del rendimiento, 172

sistemas de punto de venta, 168-169

documentación de proceso, 155

disminución del incidente de producción, 158, 159 *f*

frecuencia de liberación, 153 , 154*F*

riesgo de liberación, 165

lanzamientos definidos, 164-165

Resiliencia, 172

retroceder, 172

implementación de desarrollador de autoservicio, 162-164

Equipo de operaciones compartidas, 157

prueba de humo, 156, 163

LRR. *Ver* Revisión de preparación de lanzamiento

METRO

Macri, Bethany, 363

Macys.com 136

haciendo visible el trabajo, 15-17, 73 , 104

Malpass, Ian, 197 ,276

Mangot, Dave, 337–338

plazo de entrega de fabricación, 4 4

flujo de valor de fabricación

definido, 7

descripción de, 7–8

problemas de retroalimentación, 29

aprendizaje integrador, 37

ambiente de baja confianza, 37

flujo de trabajo, 7–8

Marsh, Dianne, 98

Martin, Karen, 7 7

Massie, Bill, 339, 341

Mateo, Reena, 337–338

Maximilien, E. Michael, 135*norte*

McDonnell, Patrick, 196

McKinley, Dan, 298

Mediratta, Bharat, 124 ,306–307

Messeri, Eran, 125 ,291

metadatos, 115

Metasploit, 325

métrica

negocio procesable, 211*F*

aplicaciones y negocios, 210–212

infraestructura, 212–213

Página 431

bibliotecas 204-205

producción, 204-206

autoservicio, 207-208

fuentes, 209

para mejorar la telemetría, 65-66

Metz, Cade, 302

Mickman, Heather, 91–93 , 263, 305

Microsoft, 118

Microsoft Operations Framework, 195

Milstein, Dan, 276

producto mínimo viable, 355

MOF *Ver* Microsoft Operations Framework

Moore, Geoffrey A., 57

Morgue, 277–278

MTTR

automatización del proceso de despliegue, 158 , 159 *f*

grabación, 277

telemetría 197 , 197*F*

Mueller, Ernst, 97 *n* ,149-151 , 207

Mulkey, Jody, 84–85 , 212

MySQL

Mitos de DevOps, xvi

en Etsy 196, 297–298

norte

Nagappan, Nachi, 135*norte*

Nagios 198

NASA, 279–280

Instrumentos Nacionales, 54–55

Seguro nacional, 304-306

Netflix

capacidad de autoescalado, 221–222

organización orientada al mercado, 81

Netflix AWS, 118 *n*

valores organizacionales, 90 *n*

redefiniendo el fracaso, 280–281

Página 432

Resiliencia, 271–273 ,281–282

plataformas de autoservicio, 98

análisis de telemetría, 215–216

Newland, Jesse, 287–289

Nike, 159

Nmap, 325

requisitos no funcionales, 294

Nordstrom 51–55 ,61–62

Norte, Dan, 168–169, 202

Programa NR, 42–43

Nygard, Michael, 281

O

Obidos, 184

Capa de mapeo relacional de objetos, 79 , 79*norte*

O'Donnell, Glenn, 304

Ohno, Taiichi, 18 *n*

O'Neill, Paul, 41, 279

Proyecto de seguridad de aplicaciones web abiertas, 319, 319*n* ,324*norte*

Operación Escudo del Desierto, 165*norte*

Operación Inversión, 72

Operaciones *Ver* entradas en Ops

Operaciones y resultados orientados al mercado

creando servicios compartidos, 97-99

ingenieros integrados en equipos de servicio, 99–100

comentarios, 103

integración en rituales de desarrollo, 101-104

equipos internos de servicios compartidos, 97-99

enlaces asignados a equipos de servicio, 100-101

haciendo visible el trabajo, 104

Enlaces de operaciones, 96

participación en retrospectivas de desarrollo, 102-103

participación en Dev standups, 102

plataformas de autoservicio, 97-98

silos, 102

estandarización de herramientas, 98-99

Página 433

optimización para centros de trabajo posteriores

tipos de clientes, 34

Diseñando para principios de fabricación, 34

Oráculo

ajustes de configuración de la aplicación, 368

Software COTS, 361

eBay 179*norte*

Migración de código ERP, 117*norte*

LinkedIn, 71

culturas organizacionales, 39, 39 *f*

conocimiento organizacional

pruebas automatizadas como documentación, 293

automatizar procesos estandarizados, 289–290

ChatOps, 287-289

reutilización de código, 289–290

requisitos no funcionales codificados, 294

comunidades de práctica, 293

Estudio de caso de Etsy, 297–298

Hubot, 287-289
integrando la automatización en salas de chat, 287–289
historias de usuario de Ops de gran valor, 295
repositorio de código fuente, 290–292
opciones tecnológicas para lograr los objetivos de la organización, 295–298
estandarización de la pila tecnológica, 297–298
desarrollo basado en pruebas, 293
usando salas de chat y bots de chat, 287–289
aprendizaje organizacional y mejora
asistir a conferencias externas, 304–306
objetivos de bombardeo, 301
Estudio de caso del Capitol One, 304–306
permitir el aprendizaje y la enseñanza, 303–304
bombardeo de mejora, 299
consultoría interna y coaching, 306–307
kaizen blitz, 299
Estudio de caso de Nationwide Insurance, 304–306

Página 434

pagar deuda técnica, 300–303
Estudio de caso objetivo, 299–300 , 304–306
La tercera vía, 38–40
organizaciones
orientado a funciones, 80 , 84
orientado al mercado, 80–81
orientado a matriz, 80
ORM *Ver* capa de mapeo relacional de objetos
Osterling, Mike, 7 7
Özil, Giray, 256
PAG
programación en pareja
y revisiones de código, 256

descripción de, 259–263

par de reversión programada, 139 *n*

horas de apareamiento, 260 *n*

Estudio de caso de Pivotal Labs, 260–261

Pal, Tapabrata, 263 ,305 ,313

Pandey, Ravi, 300

Paroski, Drew, 302

organizaciones patológicas, 39

PayPal, 58*norte*

Perl

Blackboard Learn, 187

eBay 179*norte*

métricas de producción, 204

Perrow, Charles 28

PHP

Ley de Conway, 78

Mitos de DevOps, xvi

Etsy, 196 , 297, 297 *n*

Facebook, 153*n* , 154 , 175, 302

ORM, 79

telemetría de producción, 205, 230, 230*F*

Pivotal Labs, 260–261

Página 435

sistemas de punto de venta, 168–169

Poppendieck, Mary, 24

Poppendieck, Tom, 24

Después de muerte. *Ver* post mortems sin culpa

visibilidad del problema, 29-30

problemas

resolución de problemas basada en hechos, 203–204

integración, 143

líderes y resolución de problemas, 44

prevención de, 30

visibilidad del problema, 29-30

enjambre de pequeños, 32

métricas de producción

Java, 204

Perl, 204

Pitón, 204 204

Ruby on Rails, 204 204

monitoreo de producción. *Ver* telemetría

Telemetría de producción. *Ver también* telemetría; análisis de telemetría

investigación contextual, 232-233

función alterna, 229–230

mecanismos de retroalimentación, 229

arreglar adelante, 230, 230*F*

equipos orientados a funciones, 231–232

Estudio de caso de Google, 237–239

Revisión de preparación para la transferencia , 238–239

mejora del flujo, 232*norte*

JavaScript, 209

revisiones de preparación para lanzamiento y entrega, 237–239, 239*F*

guía de lanzamiento, 234–235

Revisión de preparación para el lanzamiento, 238–239

haciendo más seguras las implementaciones, 229–230

equipos orientados al mercado, 231

deberes de rotación de buscapersonas, 230–232

PHP 205 , 230, 230 *f*

Página 436

servicio de producción, 234–239

objetivos de cumplimiento normativo, 235–236

retroceder, 230

mecanismo de devolución de servicio, 236*f* ,237

ingenieros de confiabilidad del sitio, 237–239

Observación UX, 233 ,233 *n*

Prugh, Scott

y bimodal IT, 57

entrenamiento cruzado, 86–87

implementaciones diarias, 157-158

telemetría 201

Laboratorios de marionetas, 159-160

Pitón

Etsy, 297 , 297 *n*

Google 296

ORM, 79*norte*

métricas de producción, 204

Q

controles de calidad, 32-34

tiempo de cola, 358–359 , 358*F*

colas

largo, 81

tiempo de cola, 358–359 ,358 *f*

trabajo compartido, 73–74

tamaño, 18 , 18 *n*

R

Rachitsky, Lenny, 365

Rajan, Karthik, 337

Rapoport, Roy, 215-216, 280–281

Proceso unificado racional, 5 ,354

Raymond, Eric A., 77

Red Hat, 114

Reddy, Tarun, 222–223

reforzando el comportamiento deseado, 73-74

patrones de liberación

basado en la aplicación, 165–166 , 171–175

canario, 169-171, 170*F*

sistema inmunitario en racimo, 169 , 170-171

basado en el medio ambiente, 165 ,166-171

lanzamientos, 164-165

Rembetsy, Michael 51, 196 ,298

repositorios

código 187*f* , 188*f* ,290–292 , 315–317

control de versiones, 115–118

resiliencia, 43–44 , 172 ,271–273 , 281–282

retrabajo, 11

Richardson, Vernon, xxviii *n*

Ries, Eric, 20, 171*n* , 355

Medios de comunicación correctos, 227–229

Robbins, Jesse, 283 ,354

Roberto, Michael 279

Retroceder, 139 *n* ,230

Rossi, Chuck, 153-155, 174 *n*

Rother, Mike

Kata como entrenador, 45

organizaciones orientadas a la funcionalidad, 84

kata de mejora, 40

Movimiento Toyota Kata, 6 6

Ruby on Rails

automatización, 127

despliegue azul-verde, 167 *n*

escaneo de dependencia, 319

seguridad de la información, 322

infraestructura de registro, 201*norte*

ORM, 79

métricas de producción, 204

Movimiento informático robusto, 355–356

S

Sachs, Marcus, 326

cultura de seguridad, 28 ,38-40

seguridad en sistemas complejos, 27-29

seguridad en el trabajo, 41–42

Salesforce.com, 337–338

Schafer, Andrew, 5 , 354

Schwaber, Ken, 102*norte*

Scott, Kevin, 72-73

Metodología Scrum, 102 *n* , 119–120

Scryer, 221–222

segundas historias 360*F*

plataformas de autoservicio, 97-98 ,206-208

Senge, Peter

organizaciones de aprendizaje, 40

visibilidad del problema, 29

mecanismo de devolución de servicio, 236*f* , 237, 237*norte*

servicios, brownfield

CSG International, 56

definido, 55

Transformación DevOps, 54–56

Etsy, 56

mejorando la velocidad y la calidad, 57

con el mayor beneficio comercial potencial, 55 *n*

transformaciones de 55–56

flujos de valor, 54-56

servicios, greenfield

consecuencias de nuevas características, 57

definido, 54

tipos de proyectos, 54-56

servicios inmutables, 185

servicios compartidos, 97–99

servicios versionados, 185

Shingo, Shigeo, 23
Shinn, Bill, 342–344

Shoup, Randy

Desglose de la tubería de despliegue, 139

arquitectura evolutiva, 179, 182

arquitectura vagamente acoplada, 90

revisiones por pares de cambios en el código, 255–256

publicidad de autopsias, 277

repositorio de código fuente, 291–292

silos

Operaciones y resultados orientados al mercado, 102

organización del equipo, 85

Ejército simio, 364–365

flujo de una pieza, 19, 20

prueba de humo, 156 ,163

suavizado 223, 223 *f* , 223*norte*

Así como. *Ver* arquitectura, orientada al servicio

Souders, Steve, 354

Spafford, George, 195, 203 *n* , 313

Lanza, Steven

condiciones de seguridad, 28

bombardeo de mejora, 299

Fallas de TI, xxvii

Aprendizaje organizacional, 40

pagar deuda técnica, 302

Resiliencia, 271

lugar de trabajo seguro, 41–42

tableros de planificación de sprint, 16-17

sprints, 119-120

Sprouter 78-80 ,88

fase de estabilización, 53*norte*

ingenieros de pila, 86

EstadísticasD, 204–205

Stillman, Jessica, 302

Stoneham, él, 246–248

patrón de aplicación de estrangulador, 180 ,185-189, 186 *n*

Sussman, Noah, 162-163

Sussna, Jeff, 233*norte*

enjambre

  Andon cord, 31, 32

Página 440

  y práctica común de gestión, 31

  objetivo de, 30

  razones para, 31

  de problemas menores, 32

sistemas de compromiso

  definido, 56–57

  y sistemas de registro brownfield relacionados, 57

sistemas de registro, 56

T

Cuadro, 223

Objetivo

  Habilitación API, 91–93

  caso de estudio, 91–93

  cortar los procesos burocráticos, 263–264

  DevOps Dojo, 299–300

  primeros DevOpsDays, 305 *n*

  conferencias tecnológicas internas, 304–306

TDD. *Ver* desarrollo, probado

organización del equipo

  Activación de API en Target, 91–93

  contextos delimitados, 89

cambios en la lógica de negocios, 78, 79

gerente de relaciones comerciales, 96

colaboración, 88

Ley de Conway, 77–78 ,88

equipos multifuncionales e independientes, 82

entrenamiento cruzado, 85–87

cambios en los procedimientos almacenados de la base de datos, 78

transferencias decrecientes, 79

ingeniero de lanzamiento dedicado, 96

problemas de implementación, 78

velocidad de despliegue y éxito, 79

incrustar las habilidades necesarias, 82-83

mentalidad fija, 87

organizaciones de orientación funcional, 80

Página 441

financiación de servicios y productos, 87–88

mentalidad de crecimiento, 87

integrando Ops en equipos de desarrollo, 95-105

equipos internos de servicios compartidos, 97-99

colas largas, 81

arquitectura vagamente acoplada, 89–93

hacer que la orientación funcional funcione, 83–84, 83*F*

orientaciones de mercado, 80–81, 82–83

orientaciones matriciales, 80

optimizando el costo, 81-82

optimizando la velocidad, 82-83

calidad como objetivo compartido, 84-85

interacciones de servicio a través de API, 89

Arquitectura orientada a Servicios, 89

dolor compartido, 85

silos, 85

especialistas versus generalistas, 85–87, 86*F*
ingenieros de pila, 86
sincronización, 78
límites del equipo, 88
tamaño del equipo, 90-91
pruebas y operaciones, 89
equipo de dos pizzas, 90-91
tamaño del equipo, 90-91
equipos
Equipo 18F, 325–326
desarrollo, 8
orientado a funciones, 231–232
orientado al mercado, 231
servicio, 83 *n* ,97-101
operaciones compartidas, 157 ,158 *n*
formación de equipo, 67
tamaño del equipo, 90-91
flujo de valor tecnológico, 8
pruebas, 124-126

Página 442

transformación, 66-74
equipo de dos pizzas, 90-91
deuda técnica
descripción de, 148
gerencia, 69–71
pagando, 144 ,300-303
reduciendo, 69-71 ,70*F*
curva de adopción de tecnología, 58*F*
flujo de valor tecnológico
ausencia de retroalimentación rápida, 29-30
creando una cultura de alta confianza, 37-38

definido, 8

plazo de implementación, 8–11

entradas, 8

aprendizaje integrador, 37–38

respuestas a incidentes y accidentes, 38–39

telemetría *Ver también* telemetría de producción; análisis de telemetría

métricas empresariales accionables, 211 *f*

registro de aplicaciones, 201–203

aplicaciones y métricas empresariales, 210–212

infraestructura centralizada, 198–200

cultura de causalidad, 195

embudo de adquisición de clientes, 210

recopilación de datos, 199

Nivel de DEPURACIÓN, 201

definido, 196

Nivel de error, 202

enrutador de eventos, 199

resolución de problemas basada en hechos, 203–204

Nivel fatal, 202

gráficos y tableros, 204-205

identificar brechas, 209–213

tiempo de resolución del incidente, 197*F*

Nivel INFO, 202

radiador de información, 206-208

Página 443

seguridad de la información en telemetría de productos, 326–327

métricas de infraestructura, 212–213

ITIL CMDB, 212

Estudio de caso de LinkedIn, 207–208

centralización de registros, 199

generación de entrada de registro, 202–203

niveles de registro, 201-202

haciendo más seguras las implementaciones, 229–230

métricas para mejorar, 65–66

bibliotecas de métricas, 204-205

biblioteca de métricas, 204-206

fuentes de métricas, 209

arquitectura de monitoreo, 198–199

marco de monitoreo, 200*F*

MTTR 197 ,197*F*

superposición de actividades de despliegue de producción, 213

métricas de producción, 204–206

y la segunda vía, 30

telemetría de seguridad, 327–330

métricas de autoservicio, 207-208

plataformas de autoservicio, 206-208

StatsD, 204-205

herramientas, 205*norte*

Nivel WARN, 202

Análisis de telemetría. *Ver también* telemetría de producción; telemetría

3 regla de desviación estándar, 219, 219*f* ,225 , 225 *f*

alertas de resultados no deseados, 218

herramientas de análisis, 224 *n*

técnicas de detección de anomalías, 222–226

automatizado, 222*F*

capacidad de autoescalado, 221–222 , 221 *f*

caso de estudio, 215-216

técnicas de filtrado, 224

Distribución gaussiana, 217, 217*F*

Prueba de Kolmogorov-Smirnov, 224, 225 , 226 *f*

Página 444

medio, 216–217

Estudio de caso de Netflix, 221–222
distribución no gaussiana, 219–222, 220 *f*
técnicas no paramétricas, 225
detección de valores atípicos, 215-216
precursores de incidentes de producción, 218
Detección de valores atípicos del servidor, 216
suavizado 223, 223 *f* ,223 *n*
desviaciones estandar, 216–217 ,217*F*
técnicas estadísticas 216–217 , 223
entornos de prueba, 113*norte*
pruebas
automatizado, 123-127, 125*n* ,130 ,134-135 , 136 ,293
conjunto de pruebas de validación automatizada, 132, 133-134 ,133 *f* , 134*f* , 136-138
pruebas destructivas, 338
prueba rápida, 132, 133-134
prueba ideal vs no ideal, 133 *f*
prueba manual, 258–259
pruebas de requisitos no funcionales, 137–138
pruebas de rendimiento, 136-137
prueba de humo, 156, 163
entornos de prueba, 113
prueba en paralelo, 133-134, 134*F*
prueba, A / B
caso de estudio, 246–248
API de funciones, 245
historia de, 243
integración en la planificación de funciones, 245–248
integración en pruebas de características, 244–245
integrando en lanzamiento, 245
resultados de, 244
investigación de usuarios, 243*n* ,244–245
Yahoo! Respuestas246–248
pruebas automatizadas
desarrollo impulsado por pruebas de aceptación, 134–135

pruebas de aceptación, 131 , 132, 139

herramientas de análisis, 138

procesos automatizados de construcción y prueba, 127

suites de pruebas automatizadas, 126

conjunto de pruebas de validación automatizada, 129–138

automatización de pruebas manuales, 135-136

cambiar despliegue, 124

herramientas de gestión de configuración de código, 138

embalaje de código, 128

tubería de despliegue, 127-129

validación del entorno, 137–138

detección de errores, 132-133

indicadores de falla, 139

prueba rápida, 132, 133-134

comentarios, 126 , 130

construcciones verdes, 129–130

manejo de entrada desde puntos de integración externos, 131 *n*

prueba ideal vs no ideal, 133 *f*

pruebas de integración, 131 ,132

pruebas de requisitos no funcionales, 137–138

pruebas de rendimiento, 136-137

aumentos de producción, 124

tipos de prueba, 130–131

desarrollo basado en pruebas, 134-135

prueba en paralelo, 133-134, 134*F*

equipos de prueba, 124-126

y desarrollo basado en troncales, 145–146

pruebas unitarias, 130–131 , 132–133 , 139

prueba no confiable, 135

control de versiones, 128

La primera forma

comparación de tamaño de lote, 20*F*

cuellos de botella, 22

identificación de restricciones, 21-23
continuo, 20

Página 446

tamaño de cola de control, 18
descripción de, 11
gestión de errores, 19
reducción de traspaso, 21
flujo de trabajo creciente, 15
tableros kanban, 16-17
lotes grandes, 19-20
Limitar el trabajo en proceso, 17-18
pérdida de conocimiento, 21
haciendo visible el trabajo, 15-17
multitarea, 17-18
reduciendo el tamaño del lote, 18-20
flujo de una pieza, 19, 20
lotes pequeños, 18-20
estrategia de lotes pequeños, 19
tableros de planificación de sprint, 16-17
transferencia de trabajo, 15-16
eliminación de residuos, 23-25
interrupciones de trabajo, 17
gestión del flujo de trabajo, 16, 21
visualizaciones de flujo de trabajo, 16–17
La segunda manera
Andon cord, 31, 32
condiciones de seguridad, 28–29
descripción de, 12
detección de errores, 28
fracaso, 28
circuitos de retroalimentacion, 29, 30

bucles de avance, 29 , 30
aumento del flujo de información, 29

controles de calidad ineficaces, 32–34

optimización para centros de trabajo posteriores , 34–35

revisiones por pares, 33–34

visibilidad del problema, 29-30

Automatización de control de calidad, 33–34

Página 447

seguridad en sistemas complejos, 27-29

pululando, 30-32

telemetría 30

La tercera vía

autopsias sin culpa, 40

conocimiento colectivo, 42–43

creando una cultura de alta confianza, 37-38

descripción de, 12-13

mejora del trabajo diario, 40-42

aprendizaje integrador, 37–38

el intercambio de conocimientos, 42–43

liderazgo, 44

cultura de aprendizaje, 44–46

culturas organizacionales, 39 , 39*F*

Aprendizaje organizacional, 38-40

Resiliencia, 43-44

cultura de seguridad, 38-40

soluciones, 40

lugar de trabajo seguro, 41–42

Las tres formas

La primera forma, 11

ilustrado, 12*F*

flujo de trabajo creciente, 12

El Proyecto Phoenix 11

La segunda manera, 12

La tercera vía, 12-13

Teoría de las restricciones, 356–357

Three Mile Island, 28

herramientas de alerta basadas en umbrales, 199*norte*

Ticketmaster, 84–85

Timberland, 67

Tischler, Tim, 159

Tomayko, Ryan, 261–262

Mantenimiento Productivo Total, 4 , 353

Toyota Kata

Página 448

descripción de, 6 6

organizaciones orientadas a la funcionalidad, 84

y la mejora del trabajo diario, 40

y cultura de aprendizaje, 45

movimiento, 355

Sistema de producción de Toyota

Cordón Andon, 138

cambiar los procesos de aprobación, 253

bombardeo de mejora, 299

radiador de información, 206

y el movimiento Lean, 4 ,353

y cultura de aprendizaje, 45

y sistemas seguros, 28

Movimiento Toyota Kata, 6 6

tiempo de actividad transparente, 365

Treynor, Ben, 237–238

Trimble, Chris, 66

Turnbull, James, 198

Gorjeo, 320–323

equipo de dos pizzas, 90-91

U

Nosotros marina de guerra, 42

investigación de usuarios, 244–245

historias de usuarios, 8, 295 ,335, 336

V

flujo de valor

Desarrollo, 63

Infosec, 63

fabricación, 7–8

Operaciones, 63

dueño del producto, 63

gerentes de liberación, 63

miembros de apoyo, 63

tecnología, 8–11

ejecutivos de tecnología, 63

Página 449

prueba, 63

gerente de flujo de valor, 63

mapeo de flujo de valor, 4 4, 61-62, 353

mapeo de flujo de valor

%CALIFORNIA, sesenta y cinco

áreas de enfoque, 64

creando, 63–66

ejemplo de, 65 *f*

primer pase, 65

mapa de flujo de valor futuro, 66

y el movimiento Lean, 4 ,353

métricas para mejorar, 65–66

mejoras en la cadena de valor, 61–62

flujos de valor

expandiendo DevOps, 58-59

servicios greenfield vs brownfield, 54–56

aprovechando a los innovadores, 57-58

seleccionar una secuencia para la transformación DevOps, 51–60

sistemas de compromiso, 56–57

sistemas de registro, 56

curva de adopción de tecnología, 58 *f*

Van Leeuwen, Evelijn, 305 *n*

Vance, Ashlee, 72

Movimiento de velocidad, 354

control de versiones

activos para registrar en el repositorio de control de versiones, 116

pruebas automatizadas, 128

derivación, 143*norte*

e integración continua, 148-149

papel crítico del control de versiones, 117

entornos almacenados en el control de versiones, 115–116

metadatos 115

repositorio de control de versiones compartido, 115–118

control de versiones como predictor del desempeño organizacional, 117

sistemas de control de versiones, 115–118

Vincent, John, 216

Vogels, Werner, 91, 184

W

Wall Street Journal, 66

desperdicio y dificultades

defectos, 25

características adicionales, 24

procesos adicionales, 24

heroicos, 25 , 25 *n*

movimiento, 24-25
trabajo no estándar o manual, 25
trabajo parcialmente realizado, 24
cambiar de tarea, 24
esperando, 24
eliminación de residuos, 23-25
antipatrón de caída de agua Scrum, 140*norte*
Westrum, Ron, 39-40
Wickett, James, 313
Williams, Laurie, 135 *n* , 260
Willis, John
Infraestructura ágil, 5
convergencia de DevOps, 3
WIP *Ver* trabajo en proceso
Wolaberg, Kirsten, 58 *n*
Womack, James P.
tamaños de lote, 19
líderes y resolución de problemas, 44
Wong, Eric, 208
trabajo en proceso
tamaño de cola de control, 18
interrupciones, 17
multitarea, 17-18
visibilidad de trabajo, 15-17, 73 ,104
flujo de trabajo
creciente, 12 , 15

Página 451

gestión, 16
visualizaciones, 16-17
Y
Yahoo! Respuestas246–248

Z

Zenoss, 198, 208

Zhao, Haiping, 302

Zuckerberg, Mark, 302

Expresiones de gratitud

Jez Humble
Crear este libro ha sido una labor de amor para Gene en
especial. Es un inmenso privilegio y placer
he trabajado con Gene y mis otros coautores, John
y Pat, junto con Todd, Anna, Robyn y el editorial
y equipo de producción en IT Revolution preparando esto
trabajo, gracias. También quiero agradecer a Nicole Forsgren
cuyo trabajo con Gene, Alanna Brown, Nigel Kersten y
I en el Informe PuppetLabs / DORA *Estado de DevOps* sobre
Los últimos tres años han sido fundamentales para el desarrollo,
probando y refinando muchas de las ideas en este libro. Mi
esposa, Rani, y mis dos hijas, Amrita y Reshmi,
me han brindado amor y apoyo ilimitados durante mi
trabajo en este libro, como en cada parte de mi vida. Gracias.
te quiero. Finalmente, me siento increíblemente afortunado de ser parte de
Comunidad DevOps, que casi sin excepción
camina la charla de practicar la empatía y hacer crecer un
cultura de respeto y aprendizaje. Gracias a todos y cada uno
uno de ustedes.

John Willis
Ante todo, necesito reconocer a mi santo de un
esposa por aguantar mi loca carrera. Tomaría
otro libro para expresar cuánto aprendí de mi compañero

autores Patrick, Gene y Jez. Otro muy importante
Los influenciadores y asesores en mi viaje son Mark Hinkle,
Mark Burgess, Andrew Clay Shafer y Michael Cote. yo
También quiero agradecer a Adam Jacob por la contratación
yo en Chef y dándome la libertad de explorar, en el
primeros días, a esto le llamamos Devops. Por último pero definitivamente
no menos importante es mi compañero en el crimen, mi *co*- anfitrión de *Devops Cafe* ,
Damon Edwards

Patrick Debois
Me gustaría agradecer a los que estuvieron en este viaje, mucho
agradecimiento a todos ustedes.

Gene Kim
No puedo agradecer a Margueritte, mi amada esposa de casi
once años increíbles, suficientes para aguantarme
estar en modo de plazo por más de cinco años, así como mi
hijos, Reid, Parker y Grant. Y por supuesto, mi
padres, Ben y Gail Kim, por ayudarme a ser un
empollón temprano en la vida. También quiero agradecer a mis compañeros co-
autores de todo lo que aprendí de ellos, también
como Anna Noak, Aly Hoffman, Robyn Crummer-Olsen,
Todd Sattersten y el resto del equipo de IT Revolution
por guiar este libro hasta su finalización.

Estoy muy agradecido por todas las personas que me enseñaron
muchas cosas, que forman la base de este libro:
John Allspaw (Etsy), Alanna Brown (Marioneta), Adrian
Cockcroft (Battery Ventures), Justin Collins (guardafrenos

Pro), Josh Corman (Consejo Atlántico), Jason Cox (El Walt Disney Company), Dominica DeGrandis (LeanKit),

Damon Edwards (DTO Solutions), Dra. Nicole Forsgren (Chef), Gary Gruver, Sam Guckenheimer (Microsoft), Elisabeth Hendrickson (software fundamental), Nick Galbreath (Ciencias de la señal), Tom Limoncelli (Pila Intercambio), Chris Little, Ryan Martens, Ernest Mueller (AlienVault), Mike Orzen, Scott Prugh (CSG Internacional), Roy Rapoport (Netflix), Tarun Reddy (CA / Rally), Jesse Robbins (Orion Labs), Ben Rockwood (Chef), Andrew Shafer (Pivotal), Randy Shoup (Stitch Fix), James Turnbull (Kickstarter) y James Wickett (Ciencias de la señal).

También quiero agradecer a las muchas personas cuya increíble Viajes de DevOps que estudiamos, incluidos Justin Arbuckle, David Ashman, Charlie Betz, Mike Bland, Dr. Toufic Boubez, Em Campbell-Pretty, Jason Chan, Pete Cheslock, Ross Clanton, Jonathan Claudius, Shawn Davenport, James DeLuccia, Rob England, John Esser, James Fryman, Paul Farrall, Nathen Harvey, Mirco Hering, Adam Jacob, Luke Kanies, Kaimar Karu, Nigel Kersten, Courtney Kissler, Bethany Macri, Simon Morris, Ian Malpass, Dianne Marsh, Norman Marks, Bill Massie, Neil Matatall, Michael Nygard, Patrick McDonnell, Eran Messeri, Heather Mickman, Jody Mulkey, Paul Muller,

Jesse Newland, Dan North, Dr. Tapabrata Pal, Michael Rembetsy, Mike Rother, Paul Stack, Gareth Rushgrove, Mark Schwartz, Nathan Shimek, Bill Shinn, JP Schneider, Dr. Steven Spear, Laurence Sweeney, Jim Stoneham y Ryan Tomayko.

Y estoy profundamente agradecido por los muchos críticos quien nos dio comentarios fantásticos que dieron forma a este libro:

Will Albenzi, JT Armstrong, Paul Auclair, Ed Bellis, Daniel Blander, Matt Brender, Alanna Brown, Branden Burton, Ross Clanton, Adrian Cockcroft, Jennifer Davis, Jessica DeVita, Stephen Feldman, Martin Fisher, Stephen Fishman, Jeff Gallimore, Becky Hartman, Matt Hatch, William Hertling, Rob Hirschfeld, Tim Hunter, Stein Inge Morisbak, Mark Klein, Alan Kraft, Bridget Kromhaut, Chris Leavory, Chris Leavoy, Jenny Madorsky, Dave Mangot, Chris McDevitt, Chris McEniry, Mike McGarr, Thomas McGonagle, Sam McLeod, Byron Miller, David Mortman, Chivas Nambiar, Charles Nelles, John Osborne, Matt O'Keefe, Manuel País, Gary Pedretti, Dan Piessens, Brian Prince, Dennis Ravenelle, Pete Reid, Markos Rendell, Trevor Roberts, Jr., Frederick Scholl, Matthew Selheimer, David Severski, Samir Shah, Paul Stack, Scott Stockton, Dave Tempero, Todd Varland, Jeremy Voorhis y Branden Williams.

Y varias personas me dieron una visión increíble de lo que El futuro de la creación con cadenas de herramientas modernas

como Andrew Odewahn (O'Reilly Media) quien
usemos la fantástica plataforma de revisión Chimera,
James Turnbull (Kickstarter) por su ayuda creando mi
primera cadena de herramientas de renderizado de publicación, y Scott Chacon
(GitHub) por su trabajo en GitHub Flow para autores.

Biografías de autor

GENE Kim

Gene Kim es un galardonado con múltiples premios. CTO, investigador y autor de *The Phoenix Proyecto: una novela sobre TI, DevOps y Ayudando a que su negocio gane* y lo *visible Manual de operaciones* . Es fundador de TI Revolution y presenta la DevOps Enteprise Conferencias cumbre.

Jez Humble es coautor de *Lean Enterprise* y el galardonado Jolt

JEZ HUMILDE	enseña en UC Berkeley, y es CTO y co-fundador de DevOps Research y Valoración, LLC.
PATRICIO DEBOIS	**Patrick Debois** es un IT independiente consultor que está cerrando la brecha entre proyectos y operaciones utilizando Agile técnicas, en desarrollo, proyecto gestión y administración del sistema.
JOHN WILLIS	**John Willis** ha trabajado en TI industria de gestión por más de treinta-cinco años. Es autor de seis IBM. Redbooks y fue el fundador y jefe arquitecto en Chain Bridge Systems. Actualmente él es evangelista en Docker, Inc.