Parte 4: La Segunda Forma - FEEDBACK

CertiProf®

Professional Knowledge

www.certiprof.com

4.0 Las Prácticas Técnicas de la Retroalimentación

¿Por qué la Segunda Forma?

En el ámbito de la tecnología, nuestro trabajo se desarrolla casi por completo en sistemas complejos con un alto riesgo de consecuencias catastróficas. Como en la fabricación, a menudo descubrimos problemas sólo cuando se producen grandes fallos, como una interrupción masiva de la producción o una de producción o un fallo de seguridad que provoque el robo de datos de los clientes.

Hacemos que nuestro sistema de trabajo sea más seguro creando un flujo de información rápido, frecuente y de alta calidad de alta calidad a través de nuestro flujo de valor y nuestra organización, que que incluye bucles de retroalimentación y alimentación. Esto nos permite:

- Detectar y remediar problemas mientras aún son pequeños, más baratos y más fáciles de arreglar
- Evitar los problemas antes de que causen una catástrofe, y
- Crear un aprendizaje organizativo que que integramos en el trabajo futuro
- Tratar a los fallos y accidentes, cómo oportunidades de aprendizaje, en lugar de una causa de castigo y culpa

Puntos a cubrir

Mientras que la Primera Vía describe los principios que permiten el rápido flujo de traba de izquierda a derecha, la Segunda Vía describe los principios que permiten la de la derecha a la izquierda en todas las etapas del flujo de valor. Nuestro objetivo es crear un sistema de trabajo cada vez más seguro y resiliente.

En esta sección explicaremos:

1. Cómo crear telemetría para poder ver y resolver problemas
2. Utilizar nuestra telemetría para anticipar mejor los problemas y alcanzar los objetivos
3. Integrar la investigación y el feedback de los usuarios en el trabajo de los equipos de producto
4. Permitir la retroalimentación para que los departamentos de desarrollo y operaciones puedan realizar los despliegues de forma segura
5. Permitir la retroalimentación para aumentar la calidad de nuestro trabajo a través de revisiones por pares y la programación por pares

Agenda

- 4.1 Cómo crear telemetría para poder ver y resolver problemas.
- 4.2 Integrar la investigación y el feedback de los usuarios en el trabajo de los equipos de producto
- 4.3 Permitir la retroalimentación para que los departamentos de desarrollo y operaciones puedan realizar los despliegues de forma segura
- 4.4 Permitir la retroalimentación para aumentar la calidad de nuestro trabajo a través de revisiones por pares y la programación por pares

4.1 Telemetría: ¿Por qué?

Crear telemetría para permitir ver y resolver problemas

Un hecho de la vida en Operaciones, o cuando trabajamos con sistemas complejos, es que las cosas van a salir mal: pequeños cambios pueden llevar a resultados inesperados, incluyendo cortes y fallos globales que afectan a todos nuestros clientes. Esta es la realidad de la gestión de sistemas complejos: ninguna persona puede ver todo el sistema y entender cómo encajan todas las piezas.

Durante una interrupción al servicio, es posible que no se pueda determinar si el problema se produce debido a una falla:

- En nuestra aplicación (por ejemplo, defecto en el código)
- En nuestro entorno (por ejemplo, un problema de red, problema de configuración del servidor)
- Algo totalmente externo a nosotros (por ejemplo, un ataque masivo de denegación de servicio)

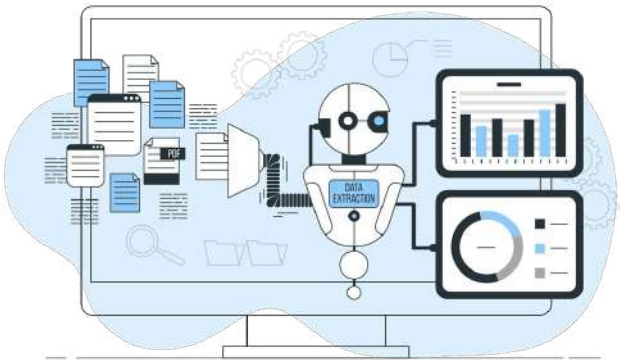
Ante esta situación, debemos usar un enfoque disciplinado para resolver problemas, utilizando la telemetría de la producción para entender los posibles factores que están contribuyendo al evento y así poder enfocarse en la solución de problemas, a diferencia de simplemente "reiniciar los servidores" a cada rato.

La telemetría cómo habilitador en el diagnóstico

Para habilitar este comportamiento disciplinado de resolución de problemas, tenemos que diseñar nuestros sistemas para crear continuamente telemetría.

Telemetría se define como "un proceso de comunicación automatizado por el que se recogen mediciones y otros datos en puntos remotos y se transmiten a equipos receptores para su control".

Nuestro objetivo es crear telemetría dentro de nuestras aplicaciones y entornos, tanto en nuestra pre-producción como en los ambientes productivos, incluyendo nuestra cadena de despliegue.



Cultura de Causalidad: las organizaciones de mejor desempeño resultaron ser mucho más efectivas tanto en el diagnóstico, cómo en la corrección de los incidentes de servicio que sus pares.

¿Por qué la telemetría?

En la Conferencia Velocity de 2012, McDonnell describió cuánto riesgo eso creaba:

"Estábamos cambiando algunas de nuestras infraestructuras más críticas, que, idealmente, los clientes nunca notarían. Sin embargo, ellos definitivamente notarán si nosotros estropeemos algo. Necesitábamos más métricas para darnos la confianza de que no estábamos realmente rompiendo las cosas mientras hacíamos estos grandes cambios, tanto para nuestros equipos de ingeniería y para miembros del equipo en áreas no técnicas, como marketing".

"Comenzamos a recoger toda la información de nuestro servidor en una herramienta de nombre Ganglia, mostrando toda la información en Graphite, una herramienta de código abierto en la que invertimos pesadamente. Al hacer esto, podríamos ver más rápidamente cualquier efecto colateral de el despliegue no intencional. Incluso empezamos a poner pantallas de televisión en toda la oficina para que todos pudieran ver el rendimiento de nuestros servicios".

Mejora la capacidad de resolver incidentes

Las organizaciones de alta performance resuelven incidentes de producción 168 veces más rápido que sus pares, con un promedio de alto rendimiento que tiene un MTTR medido en minutos, mientras que la mediana de bajo rendimiento tuvo un MTTR medido en días. Las dos principales prácticas técnicas que permitieron el MTTR rápido fueron:

- El uso del control de versiones por Operaciones
- La utilización de telemetría y monitoreo proactivo en el entorno de producción

El objetivo es garantizar que siempre tengamos telemetría suficiente para poder confirmar que nuestros servicios están funcionando correctamente en todos nuestros ambientes.

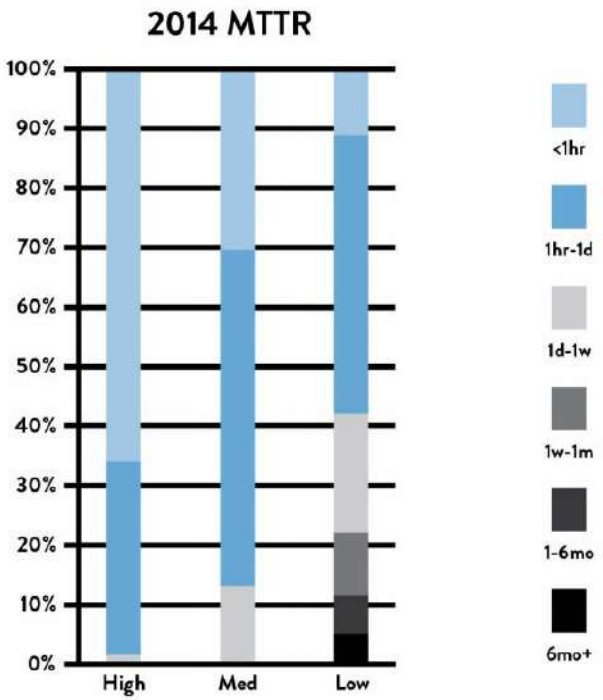


Figure 25: *Incident resolution time for high, medium, and low performers (Source: Puppet Labs, 2014 State of DevOps Report.)*

Fuente: DevOps Handbook

Resumen

El objetivo es asegurar que siempre tengamos suficiente telemetría para que podamos confirmar que nuestros servicios están funcionando correctamente en producción.

Y cuando se produzcan problemas, hacer que sea posible determinar rápidamente lo que va mal y tomar decisiones informadas sobre la mejor manera de solucionarlo, idealmente mucho antes de que los clientes se vean afectados.

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.

4.1 Telemetría: Arq. de Monitoreo Moderna

Infraestructura de Telemetría Centralizada

Durante décadas hemos acabado con silos de información, donde Desarrollo sólo crea eventos de registro que son útiles para los desarrolladores, y Operaciones sólo supervisa si los entornos están activos o inactivos.

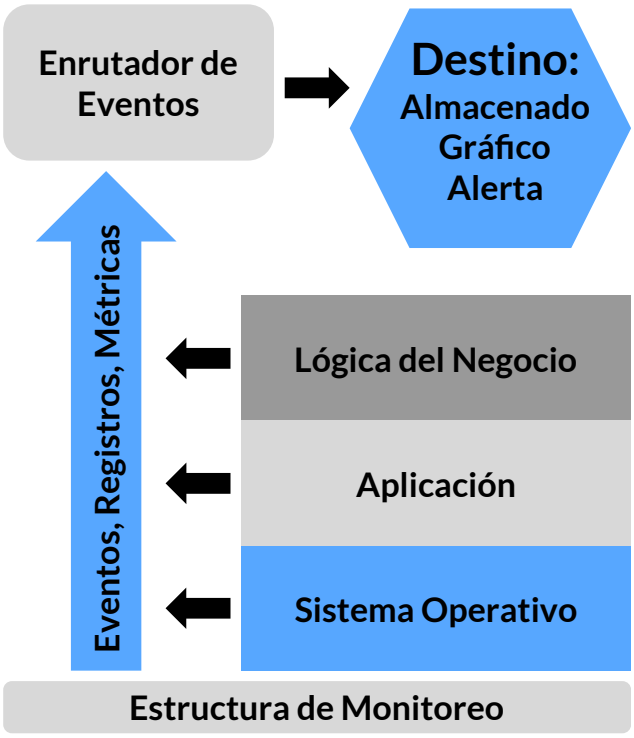
El resultado es que cuando ocurren eventos inoportunos, nadie puede determinar por qué todo el sistema no está funcionando como se diseñó o qué componente específico está fallando, impidiendo nuestra capacidad de devolver el sistema a su estado de funcionamiento.

Debemos diseñar y desarrollar nuestras aplicaciones y entornos para que generen telemetría suficiente, permitiéndonos entender cómo nuestro sistema se está comportando como un todo.

Infraestructura de Telemetría Centralizada y Moderna

Cuando todos los niveles de nuestro ecosistema de aplicaciones tienen monitoreo y registro, habilitamos otros recursos importantes, como gráficos y visualización de nuestras métricas, detección de anomalías, alerta proactiva y escalonamiento, etc. Para ello se sugiere:

- Recolección de datos en la capa de lógica de negocios, aplicaciones y entornos: para crear telemetría en forma de eventos, logs y métricas
- Un enrutador de eventos responsable de almacenar nuestros eventos y métricas: este recurso permite la visualización, la tendencia, la alerta, la detección de anomalías, etc



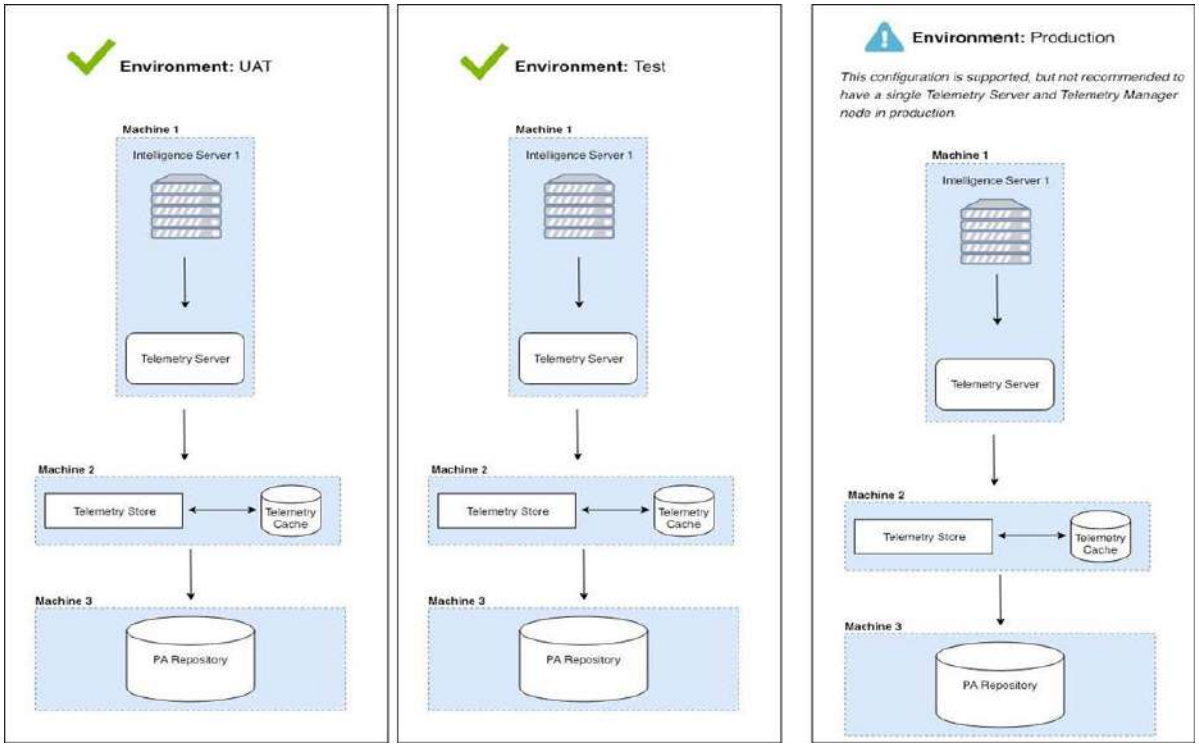
Arquitectura de monitoreo moderna

- Al centralizar los registros:
  - Podemos convertirlos en métricas que podemos contabilizar en el enrutador de eventos
- Al transformar los logs en métricas:
  - Es posible realizar operaciones estadísticas, como usar la detección de anomalías para encontrar valores discrepantes y variaciones más rápidamente

Adicionalmente, podemos recoger telemetría en el pipeline de despliegue en eventos importantes, ej.: las pruebas automatizadas que pasan o fallan o en los despliegues que se están realizando en cualquiera de los entornos.

También puedo recoger telemetría de la duración o el tiempo que se está tomando el ejecutar nuestros Builds o corridas de pruebas de regresión.

Al hacer esto, podemos detectar condiciones que podrían indicar problemas, como por ejemplo si la prueba de rendimiento o nuestra compilación tarda el doble de lo normal, lo que nos permite encontrar y corregir los errores antes de que pasen a producción.



https://www2.microstrategy.com/producthelp/Current/PlatformAnalytics/en-us/Content/pa_architecture_examples.htm

4.1 Telemetría: Logging de Aplicaciones

Crear Telemetría De Registro De Aplicaciones Que Ayuda A La Producción

Todos los miembros del pipeline utilizarán la telemetría de varias maneras:

- El equipo DEV pueden crear temporalmente más telemetría en su aplicación para diagnosticar mejor los **problemas en su local**
- El equipo OPS puede usar la telemetría para diagnosticar un **problema de producción**
- *Infosec* y los auditores pueden revisar la telemetría para confirmar la eficacia de un control necesario, rastrear **los resultados de negocios, el uso de recursos o las tasas de conversión**

Crear registros de telemetría

Para soportar estos varios modelos de uso, tenemos diferentes niveles de registro, algunos de los cuales también pueden accionar alertas, como los siguientes:

- **Nivel de DEBUG**
- **Nivel INFO**
- **Nivel de WARN (alerta)**
- **Nivel de ERROR (excepción)**
- **Nivel FATAL**

Crear registros de telemetría

Todos los eventos potencialmente significativos de la aplicación deben generar entradas de registro, incluidas las que figuran en esta lista elaborada por Anton A. Chuvakin, vicepresidente de investigación del grupo GTP Security and Risk Management de Gartner:

- Decisiones de autenticación/autorización (incluido el cierre de sesión)
- Acceso al sistema y a los datos
- Cambios en el sistema y las aplicaciones (especialmente los cambios de privilegios)
- Cambios en los datos, como añadir, editar o eliminar datos
- Entradas no válidas (posibles inyecciones maliciosas, amenazas, etc.)
- Recursos (RAM, disco, CPU, ancho de banda o cualquier otro recurso que tenga límites)
- Salud y disponibilidad del servicio
- Arranques y paradas
- Fallos y errores
- Disparos de circuit breakers
- Retrasos / Delays
- Éxito/fallo de las copias de seguridad

Logging para mejorar el análisis de los incidentes

La telemetría nos permite utilizar el método científico para formular hipótesis sobre la causa de un problema concreto y lo que se necesita para resolverlo. Ejemplos de preguntas que podemos responder durante la resolución del problema incluyen:

- ¿Qué pruebas tenemos de nuestra monitorización de que un problema se está produciendo realmente?
- ¿Cuáles son los eventos y cambios relevantes en nuestras aplicaciones y entornos que podrían haber contribuido al problema?
- ¿Qué hipótesis podemos formular para confirmar la relación entre las causas y los efectos propuestos?
- ¿Cómo podemos demostrar cuáles de estas hipótesis son correctas y solucionar el problema con éxito?

El valor de la resolución de problemas basada en hechos no sólo radica en un mejor MTTR (y mejores resultados para el cliente), sino también en el refuerzo de la percepción de una relación de beneficio mutuo entre Desarrollo y Operaciones.

4.1 Telemetría: Radiadores de Información

Crear Autoservicios para Acceso a Radiadores De Información y Telemetría

Queremos que nuestra telemetría de producción sea muy visible, lo que significa ponerla en áreas centrales donde trabajen Desarrollo y Operaciones, permitiendo así que todos los que estén interesados vean cómo están funcionando nuestros servicios (Desarrollo, Operaciones, Gestión de Productos e Infosec. entre otros)

Se promueve la responsabilidad entre los miembros del equipo, demostrando activamente valores:

- El equipo no tiene nada que esconder de sus visitantes (clientes, partes interesadas, etc.)
- El equipo no tiene nada que esconder de sí mismo: reconoce y enfrenta problemas

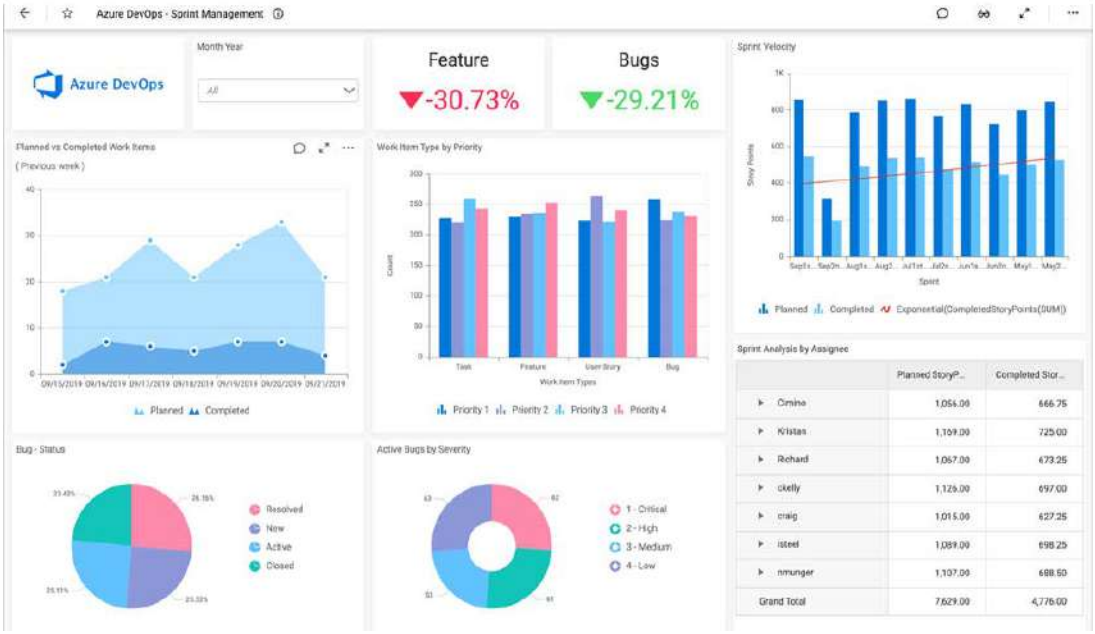
4.1 Telemetría: Brechas en la Telemetría

Encontrar y rellenar las lagunas en la Telemetría

Para conseguirlo es necesario crear suficiente telemetría en todas las caps de las aplicaciones y para todos nuestros entornos, así como para los pipelines de despliegue que los soportan. Necesitamos métricas de los siguientes niveles:

- Nivel de negocio
- Nivel de aplicación
- Nivel de infraestructura
- Nivel de software del cliente
- Nivel de pipelines de despliegue

Al tener cobertura de telemetría en todas estas áreas, podremos ver la salud de todo aquello en lo que se basa nuestro servicio, utilizando datos y hechos en lugar de rumores, acusaciones, culpas, etc.



https://www.boldbi.com/integrations/azure-devops

Encontrar y rellenar las lagunas en la Telemetría

- **Nivel de negocio:** Los ejemplos incluyen el número de transacciones de ventas, los ingresos de las transacciones de ventas, las inscripciones de usuarios, la tasa de abandono, los resultados de las pruebas A/B, etc
- **Nivel de aplicación:** Los ejemplos incluyen los tiempos de transacción, los tiempos de respuesta de los usuarios, fallos de la aplicación, etc
- **Nivel de infraestructura** (por ejemplo, base de datos, sistema operativo, red 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, aplicación móvil): Los ejemplos incluyen errores y fallos de la aplicación, tiempos de transacción medidos por el usuario, etc.
- **Nivel de Pipeline de despliegue:** Los ejemplos incluyen el estado de la cadena de construcción (por ejemplo, rojo rojo o verde para nuestros diversos conjuntos de pruebas automatizados), tiempos de despliegue de cambios de despliegue, frecuencias de despliegue, promociones de entornos de prueba y estado del entorno

4.1 Telemetría: Análisis Predictivos Telemetría

Analizar la telemetría para anticiparse mejor a los problemas y alcanzar los objetivos

Como vimos en la sección anterior necesitamos suficiente telemetría de producción en nuestras aplicaciones e infraestructura para ver y resolver los problemas a medida que se producen. Es importante que creemos herramientas que nos permitan descubrir variaciones y señales de fallo cada vez más débiles ocultas en nuestra telemetría de producción para poder evitar fallos catastróficos. Algunas técnicas que podemos usar son:

- Utilizar las medias y las desviaciones estándar para detectar posibles problemas
- Instrumentar y alertar sobre resultados no deseados
- Utilizar técnicas de detección de anomalías

Estas técnicas estadísticas que pueden utilizarse para analizar nuestra telemetría de producción, de modo que podamos encontrar y solucionar los problemas antes que ocurran, a menudo cuando todavía son pequeños y mucho antes de que causen resultados catastróficos. Esto nos permite encontrar señales de fallo cada vez más pequeñas sobre las que podemos actuar, creando un sistema de trabajo cada vez más seguro.

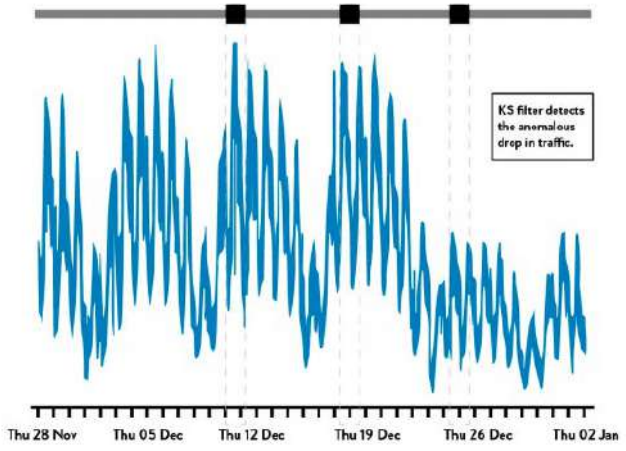


Figure 36: *Transaction volume: using Kolmogorov-Smirnov test to alert on*

Fuente: DevOps Handbook

4.2 Feedback

Permitir la retroalimentación para que el desarrollo y las operaciones puedan desplegar el código con seguridad

En esta sección hablaremos de los mecanismos de retroalimentación que nos permiten mejorar nuestro servicio en cada etapa de nuestro trabajo diario, ya sea desplegando cambios en producción, arreglando el código cuando las cosas van mal y se llama a los ingenieros, haciendo que los desarrolladores sigan su trabajo aguas abajo, creando requisitos no funcionales que ayuden a los equipos de desarrollo a escribir un código más listo para la producción, o incluso devolviendo servicios problemáticos para que sean autogestionados por Desarrollo.

Ampliar los ciclos de feedback

Al crear estos bucles de retroalimentación, hacemos que los despliegues de producción sean más seguros, aumentamos la preparación para la producción del código creado por Desarrollo y ayudamos a crear una mejor relación de trabajo entre Desarrollo y Operaciones al reforzar los objetivos, las responsabilidades y la empatía compartidos.

Algunas acciones para mejorar el feedback:

- Usar la telemetría para hacer el proceso de despliegue más seguro
- Rotación del Standby entre Operaciones y Desarrollo
- Hacer que los desarrolladores acompañan los procesos aguas abajo (downstream)
- Hacer que los desarrolladores autogestionen al principio su servicio en producción
  - Launch Guidance
  - Launch Readiness Review
  - Hand Off Readiness Review

Aumente la telemetría para liberaciones más seguras

Monitorear activamente las métricas asociadas a las funcionalidades durante el despliegue, garantiza que no violamos nuestro servicio inadvertidamente o, peor, que violamos otro servicio.

Si el cambio se rompe o perjudica cualquier funcionalidad, trabajamos rápidamente para restaurar el servicio, trayendo a quien más necesite para diagnosticar y corregir el problema.

Se puede optar por:

- **Desactivar los recursos rotos** con la "Feature Toggles"
- **Corregir**
- **Dar Marcha atrás**

Comparta deberes entre Dev y Ops (Stand-bys, Pager Rotation, Prod Issue Resolution, etc)

En cualquier servicio complejo todavía tendremos problemas inesperados, como incidentes e interrupciones que ocurren en momentos inoportunos y que se presentan de manera consistente (todas las noches a las 2 de la mañana). Estos, si no se corrigen, pueden causar problemas recurrentes con daño a las Operaciones e impacto a los usuarios.

Para evitar que esto ocurra, **todos los participantes** del flujo de valor comparten las responsabilidades de manejar los incidentes operativos. De este modo, el departamento de operaciones no se enfrenta, aislado y solo, a los problemas de producción relacionados con el código, sino que todo el mundo ayuda a encontrar el equilibrio adecuado entre la corrección de los defectos de producción y el desarrollo de nuevas funcionalidades, independientemente del lugar en el que nos encontremos en el flujo de valor.

Dev Team acompaña aguas abajo su desarrollo

Una de las técnicas más poderosas en interacción y diseño de experiencia del usuario (UX) es la **investigación contextual**, que es cuando el equipo del producto observa a un cliente utilizar la aplicación en su entorno natural, generalmente trabajando en su escritorio.

Nuestro objetivo es utilizar esa misma técnica para observar cómo nuestro trabajo afecta a nuestros clientes internos. Los desarrolladores deben seguir su trabajo en sentido descendente, de modo que puedan ver cómo los centros de trabajo descendentes deben interactuar con su producto para llevarlo a producción.

Al hacer esto, creamos una retroalimentación sobre los aspectos no funcionales de nuestro código -todos los elementos que no están relacionados con la función de cara al cliente, e identificamos formas de mejorar la capacidad de despliegue, gestión y funcionamiento, etc.

Dev teams auto gestionando sus servicios en producción al inicio

Cuando los **desarrolladores** han desplegando y están ya ejecutando su código en entornos de **producción** diariamente pueden surgir **errores graves.**

Aun equipos experimentados de Operaciones pueden experimentar lanzamientos de producción desastrosos porque es la primera vez que realmente vemos cómo nuestro código se comporta durante una liberación y bajo condiciones reales de producción.

Una contramedida potencial es tener **grupos de desarrollo auto gestionando sus servicios en producción** antes de ser elegibles para ser entregados a un grupo de operaciones administrado centralizado.

Al hacer que el **equipo de desarrollo** gestione inicialmente sus propias aplicaciones y servicios, el proceso de transición de nuevos servicios a la producción se hace mucho más **fácil y previsible.**

Ops puede retornar un módulo a desarrollo nuevamente

Para servicios **ya en producción**, necesitamos un mecanismo diferente para garantizar que Ops nunca quede atrapada en un servicio no soportable en producción.

Podemos crear un **mecanismo de devolución (handback)**, cuando un servicio de producción se vuelve muy frágil, Ops tiene la capacidad de devolver la responsabilidad del soporte a la producción **de vuelta al Dev.**

Cuando un servicio retorna a un estado **administrado por el desarrollador**, la función de **Operaciones** pasa del soporte a la **producción para la consulta,** ayudando al equipo a preparar el servicio para la producción.

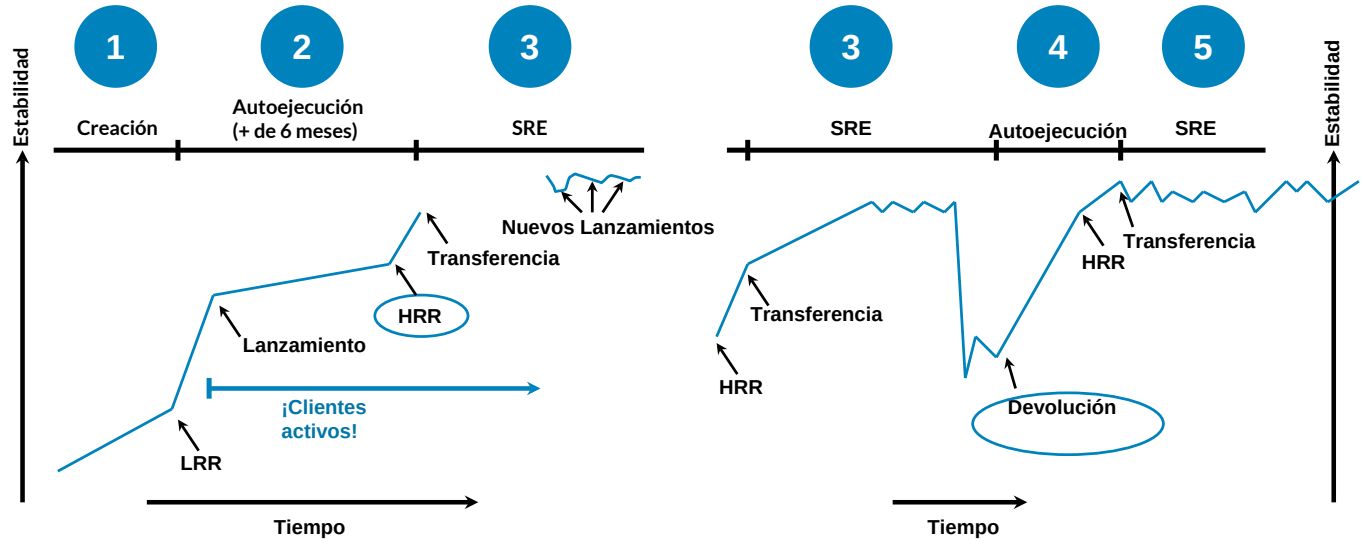
Guia de Lanzamiento (Launch Guidance)

Al crear una guía de lanzamiento, ayudamos a garantizar que cada equipo de producto se beneficie de la experiencia acumulada y colectiva de toda la organización, especialmente de Operaciones. Las orientaciones y los requisitos de lanzamiento incluirán probablemente lo siguiente:

- **Recuento y gravedad de los defectos:** ¿La aplicación funciona realmente como se ha diseñado?
- **Tipo/frecuencia de las alertas de buscapersonas:** ¿Genera la aplicación un número insoportable de alertas en producción?
- **Cobertura de la supervisión**: ¿La cobertura de la supervisión es suficiente para restablecer el servicio cuando las cosas van mal?
- **Arquitectura del sistema**: ¿Está el servicio lo suficientemente desacoplado como para soportar un alto índice de cambios y despliegues en producción?
- **Proceso de despliegue**: ¿Existe un proceso predecible, determinista y suficientemente automatizado para desplegar el código en producción?
- **Higiene de producción**: ¿Existe evidencia de suficientes buenos hábitos de producción que permitan que el soporte de producción sea gestionado por cualquier otra persona?

Superficialmente, estos requisitos pueden parecer similares a las listas de control de producción tradicionales que hemos utilizado en el pasado. Sin embargo, las diferencias clave son que requerimos una supervisión eficaz, que los despliegues sean fiables y deterministas, y una arquitectura que soporte despliegues rápidos y frecuentes.

Launch Readiness Review y Hand-Off Readiness Review



SRE: Ingeniero de Confiabilidad de Sitio
LRR: Revisión de Preparación de Lanzamiento
HRR: Revisión de Preparación para la Entrega

Fuente: DevOps Handbook

4.2 Feedback

Launch Readiness Review y Hand-Off Readiness Review

Google ha creado dos conjuntos de comprobaciones de seguridad para dos etapas críticas del lanzamiento de nuevos servicios, denominadas **Launch Readiness Review** y **Hand-Off Readiness Review** (LRR y HRR, respectivamente).

La **LRR** debe realizarse y aprobarse antes de que cualquier nuevo servicio de Google se ponga a disposición del público y reciba tráfico de producción en directo, mientras que la **HRR** se realiza cuando el servicio pasa a un estado gestionado por operaciones, normalmente meses después de la LRR.

Las listas de comprobación de la LRR y la HRR son similares, pero la HRR es mucho más estricta y tiene normas de aceptación más estrictas, mientras que la LRR es autoinformada por los equipos de producto.

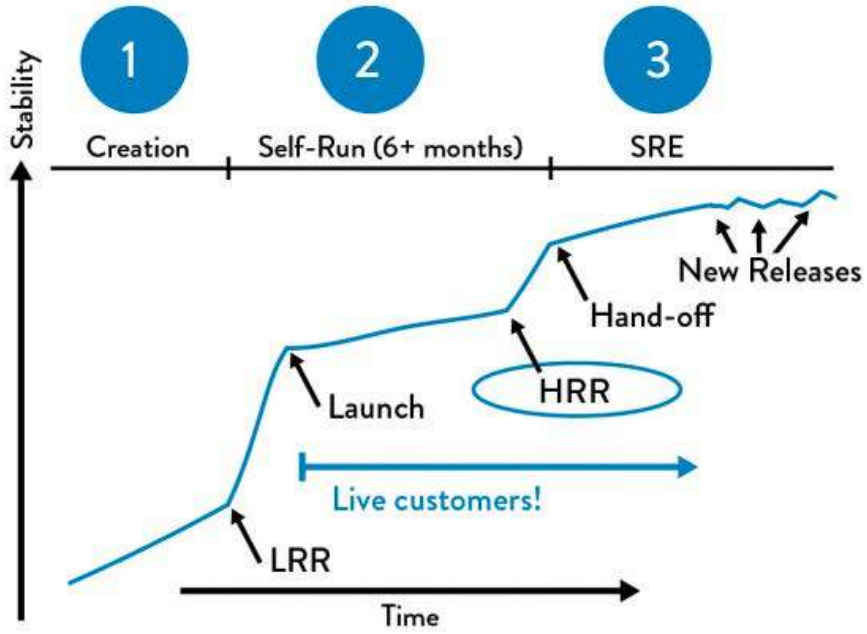


Figure 39: *The "Launch readiness review and hand-offs readiness review" at Google (Source: "SRE@Google: Thousands of DevOps Since 2004," YouTube video, 45:57, posted by USENIX, January 12, 2012, https://www.youtube.com/watch?v=iIuTnhdTzK0.)*

Fuente: DevOps Handbook

4.3 Desarrollo Basado en Hipótesis

Integrar el desarrollo basado en hipótesis y las pruebas A / B en nuestro trabajo diario

Con demasiada frecuencia, en los proyectos de software, los desarrolladores trabajan en las funciones durante meses o años, abarcando varias versiones, sin confirmar nunca si se están cumpliendo los resultados empresariales deseados, como por ejemplo si una función concreta está logrando los resultados deseados o incluso si se está utilizando.

"La forma más ineficiente de probar un modelo de negocio o una idea de producto es construir el producto completo para ver si la demanda prevista realmente existe" - Jez Humble.

Antes de construir una funcionalidad, debemos preguntarnos:

- ¿Debemos construirlo? y
- ¿Por qué?

Luego, deberíamos realizar los experimentos más baratos y rápidos posibles para validar a través de la investigación del usuario si la función deseada realmente logrará los resultados deseados. Podemos utilizar técnicas cómo:

- Desarrollo impulsado por hipótesis
- Los embudos de adquisición de clientes
- Las pruebas A / B

*"La manera más **ineficiente** de probar un modelo de negocio o una idea de producto es construir el **producto completo** para ver si la demanda prevista realmente existe".*

4.3 Desarrollo Basado en Hipótesis – Construcción de Funcionalidades

Integración de Prueba A/B en la validación de funcionalidades

Una técnica de búsqueda de usuarios es la definición del pipeline de adquisición de clientes y la realización de pruebas A / B.

La técnica A/B más comúnmente usada en la moderna práctica de UX implica un sitio en el que los visitantes se seleccionan aleatoriamente para exhibir una de las dos versiones de una página, un control (el "A") o un tratamiento (el "B").

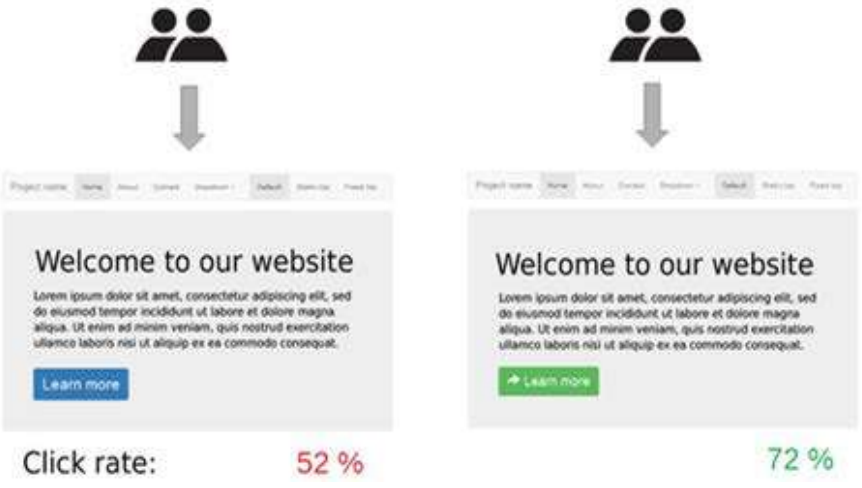
4.3 Desarrollo Basado en Hipótesis – Liberaciones

Integración de Prueba A/B en la liberación

Las pruebas A/B rápidas e iterativas son posibles gracias a la capacidad de realizar despliegues de producción de forma rápida y fácil bajo demanda.

- Esto requiere una telemetría de producción útil en todos los niveles del conjunto de aplicaciones

Al entrar en nuestro recurso o funcionalidad, se puede controlar qué porcentaje de los usuarios ve la versión de tratamiento de un experimento.



4.3 Desarrollo Basado en Hipótesis – Planificación

Integración de Prueba A/B en la planificación de funcionalidades

Una vez que tenemos la infraestructura para apoyar el lanzamiento y las pruebas de características A/B, debemos hacer que los Product Owners piensen en cada característica como una hipótesis y utilicen nuestros despliegues de producción como experimentos con usuarios reales para probar o refutar esa hipótesis.

La construcción de experimentos debe diseñarse en el contexto del general de adquisición de clientes. Barry O'Reilly, coautor de "Lean Enterprise: How High Performance Organizations Innovate at Scale" , describió cómo podemos enmarcar las hipótesis en el desarrollo de características.

Podemos enmarcar las hipótesis en el desarrollo de características de la siguiente forma:

- Creemos que aumentar el tamaño de las imágenes del hotel en la página de reservas dará lugar a una mejora del compromiso y la conversión de los clientes
- Tendremos confianza para proceder cuando veamos un aumento del 5% en clientes que revisan las imágenes del hotel y que luego proceden a reservar en cuarenta y ocho horas

Adoptar un enfoque experimental para el desarrollo de productos requiere no sólo dividir el trabajo en pequeñas unidades (historias o requisitos), sino también validar si cada unidad de trabajo genera los resultados esperados. Si no lo hace, modificamos nuestra hoja de ruta de trabajo con caminos alternativos que realmente alcancemos los resultados esperados.

4.4 Revisión de Pares

Crear procesos de revisión y coordinación para aumentar la calidad de nuestro trabajo actual

Nuestro objetivo es permitir que Desarrollo y Operaciones trabajen colaborativamente para reducir el riesgo que representa introducir cambios en la producción antes de que se realicen los despliegues.

Tradicionalmente, cuando revisamos los cambios para la implementación, tendemos a depender en gran medida de las revisiones, inspecciones y aprobaciones justo antes de la implementación. Con frecuencia, esas aprobaciones son otorgadas por equipos externos que a menudo están demasiado alejados 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 cambio.

En esta sección vamos a compartir unas prácticas técnicas más eficientes para reducir el riesgo de los despliegues en produccion.

4.4 Revisión y Coordinación

Crear procesos de revisión y coordinación para aumentar la calidad del trabajo

El proceso de revisión por pares en GitHub es un ejemplo impresionante de cómo la inspección puede aumentar la calidad, hacer que los despliegues sean más seguras y hacerlo integrándose al flujo de trabajo diario de todos.

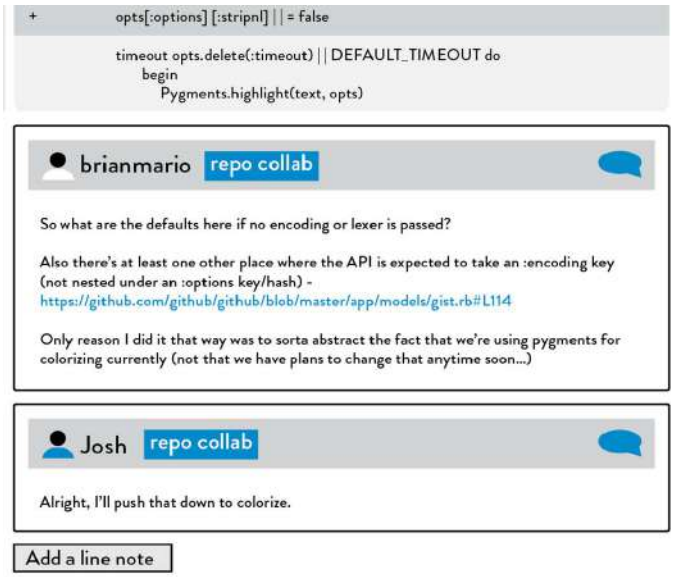
Ellos fueron pioneros en el proceso llamado pull request, una de las formas más populares de revisión por pares que abarca Dev y Ops.

El objetivo ahora es permitir que el Desarrollo y las Operaciones reduzcan el riesgo de cambios en la producción antes de que se realicen.

Crear procesos de revisión y coordinación

El Flujo de GitHub está compuesto de cinco etapas:

1. Para trabajar en algo nuevo, el desarrollador **crea una branch** local desde Master (por ejemplo, "new-auth2-scopes")
2. El desarrollador **efectúa sus cambios en ese branch localmente**, enviando regularmente su trabajo al mismo branch nombrado en el servidor
3. Cuando necesiten feedback o ayuda, o cuando crean que el branch está listo para ser "merged" crean un pull request
4. Cuando se hayan resuelto los comentarios al código y se obtengan las aprobaciones necesarias del recurso, el desarrollador podrá mezclarlo al maestro
5. Cuando los cambios de código se mezclan y se envían a master, el desarrollador las despliega en producción



Pull Request

Un pull request de mala calidad es aquel que no tiene contexto suficiente para el lector, con poca o ninguna documentación de lo que el cambio pretende hacer.

Por ejemplo, un pull request que simplemente contiene el siguiente texto: "Corregir el problema 3616 y 3841".

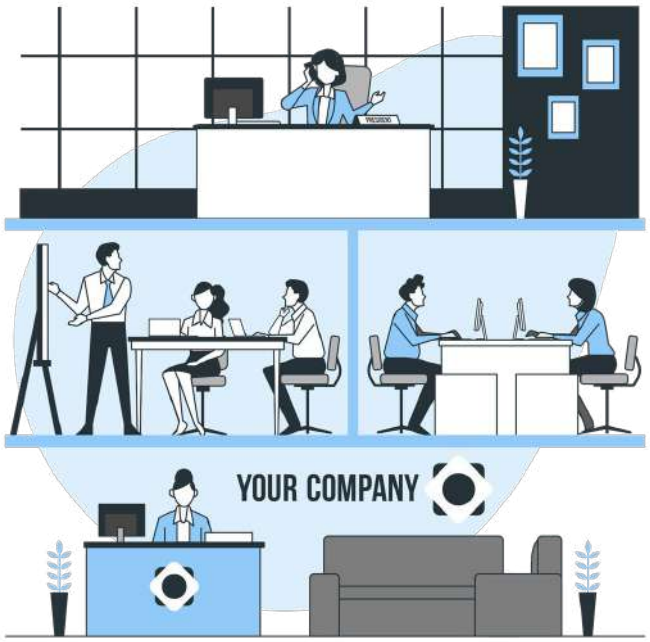
Descripción recomendada en un pull request:
- Debe haber suficientes detalles sobre por qué se está haciendo el cambio
- Como el cambio fue hecho
- Cualquier riesgo identificado
- Contramedidas resultantes

Información mejor de la solicitud tirada:
- Riesgos adicionales señalados
- Ideas sobre las mejores formas de desplegar el cambio deseado
- Ideas sobre cómo mejor atenuar el riesgo

Peligros potenciales del "control excesivo"

Los mecanismo para la gestión de cambio (Change Management) tradicionales pueden llevar a resultados negativos no intencionales, cómo lo son agregar retrasos adicionales a la entrega y una reducción la inmediatez del feedback necesario,y por lo tantos de su impacto, en el proceso de despliegue.

"Las personas más cercanas a un problema generalmente saben más sobre él" – STP.



Algunos de los controles que muchas veces implementamos cuando ocurren fallas en el proceso de control de cambios:

- Agregar más preguntas al formulario de solicitud de cambio
- Exigir más autorizaciones, como más un nivel de aprobación de gestión (por ejemplo, el vicepresidente, el CIO)
- Exigir más tiempo de preparación para aprobaciones de cambios, para que se evalúen adecuadamente

Estos controles generalmente agregan más fricción al proceso de despliegue, multiplicando el número de etapas y aprobaciones, y aumentando el tamaño de los lotes y los plazos de implementación.

Una de las creencias fundamentales del Sistema de Producción Toyota es que "las personas más cercanas a un problema suelen ser las que más saben sobre él". Esto se acentúa a medida que el trabajo que se realiza y el sistema en el que se produce el trabajo se vuelven más complejo y dinámico, como es típico en los flujos de valor de DevOps.

Como se ha demostrado repetidas veces, cuanto mayor es la distancia entre la persona que realiza el trabajo (es decir, el implementador del cambio) y la persona que decide sobre hacerlo (es decir, el autorizador de cambio), peor el resultado.

“Las personas más cercanas a un problema generalmente saben más sobre él” – STP.

Coordinación y programación del cambio

Siempre que tengamos varios grupos trabajando en sistemas que comparten dependencias es probable que haya que coordinar nuestros cambios para garantizar que no interfieran entre sí (por ejemplo, la organización, la agrupación y la secuenciación de los cambios).

En informática y diseño de sistemas, un **sistema poco acoplado** es aquel en el que cada uno de sus componentes hacen uso, o tienen poca o ninguna interacción con las funciones de otros componentes separados. Las subáreas incluyen el acoplamiento de clases, interfaces, datos y servicios. Sistemas **altamente acoplados** son los sistemas que están altamente interconectados.

En general:

- Cuanto más desacoplada sea nuestra arquitectura, menos tendremos comunicación y coordinación con otros equipos de componentes. Si arquitectura está realmente orientada a los servicios, los equipos pueden realizar cambios con un alto grado de autonomía, y es poco probable que los cambios locales provoquen interrupciones globales
- Sin embargo, incluso en una arquitectura poco acoplada, cuando muchos equipos realizan cientos de despliegues independientes al día, puede existir el riesgo de que los cambios interfieran unos con otros (por ejemplo, pruebas A/B simultáneas). Para mitigar estos riesgos, podemos utilizar salas de chat para anunciar los cambios y encontrar proactivamente los conflictos que puedan existir

Para mitigar riesgos, podemos utilizar salas de chat (tipo Slack) para anunciar cambios y localizar de manera proactiva posibles conflictos.

- En el caso de organizaciones más complejas y con arquitecturas más acopladas es posible que tengamos que programar deliberadamente nuestros cambios, y hacer que representantes de los equipos se reúnan, no para autorizar cambios, sino para programar y secuenciar sus cambios para minimizar los accidentes
- Sin embargo, en algunas áreas, como los cambios en la infraestructura global (por ejemplo, los cambios en los nodos la red, cambios en los conmutadores de la red central) siempre tendrán un mayor riesgo asociado. Estos cambios siempre requerirán contramedidas técnicas, como la redundancia la conmutación por error, la realización de pruebas exhaustivas y (en el mejor de los casos) la simulación

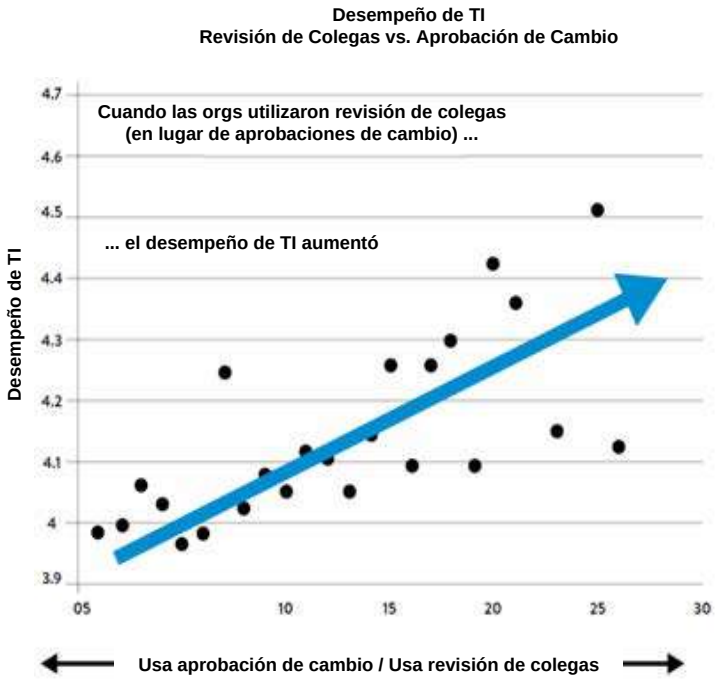
Revisión de Pares del Código

En lugar de exigir la aprobación de un órgano externo antes del despliegue, podemos exigir que los ingenieros hagan revisiones de sus cambios.

El objetivo:

- Encontrar errores, con los ingenieros cercanos al trabajo, examinando nuestros cambios
- Mejorar la calidad de nuestros cambios
- Crear los beneficios de la capacitación cruzada, el aprendizaje entre pares y la mejora de las habilidades

Pero para cambios en la base de datos o componentes esenciales para los negocios con baja cobertura de prueba automatizada, podemos exigir una revisión adicional de un especialista en el tema (por ejemplo, ingeniero de seguridad de la información, ingeniero de base de datos) o varias revisiones (por ejemplo, "+2" en lugar de sólo "+1").



Fuente: DevOps Handbook

El principio de tamaños pequeños de lotes también se aplica a revisiones de código.

- Cuanto mayor sea el tamaño de la alteración que necesita ser revisada, más tiempo lleva a entender y mayor es la carga sobre el ingeniero revisor

Existe una relación no lineal entre el tamaño del cambio y el riesgo potencial de integrar ese cambio - cuando usted pasa de un cambio de código de diez líneas a un código de cien líneas, el riesgo de algo ir mal es más de diez veces mayor, y así sucesivamente.

- La capacidad de criticar significativamente los cambios de código disminuye a medida que el tamaño del cambio aumenta. "Pida a un programador para revisar diez líneas de código, él encontrará diez ediciones. Pídale a él para hacer quinientas palabras, y él dirá que parece bueno".

Directrices generales para las revisiones de código:

- Todos deben tener a alguien que revise sus cambios antes de integrarlos a la rama principal
- Todos deben supervisar el flujo de commits de sus compañeros de equipo para que se puedan identificar y revisar los posibles conflictos
- Definir qué cambios se consideran de alto riesgo y pueden requerir la revisión de un experto (por ejemplo, cambios en la base de datos, módulos sensibles a la seguridad, como la autenticación, etc.)
- Si alguien envía un cambio que es demasiado grande para entender con facilidad, es decir, no se puede entender su impacto después de leerlo un par de veces, o hay que pedirle al remitente que lo aclare, este cambio debería dividirse en varios cambios más pequeños que puedan entenderse de un vistazo

Programación del Código en Pares

Las revisiones de código vienen en varias formas:

- Programación en pares
- Revisión sobre los hombros
- Envío de correo electrónico
- Revisión de código asistida por herramientas

Resumen

- Crear las condiciones que permitan a los ejecutores del cambio apropiarse plenamente de la calidad de sus cambios es una parte esencial de la cultura generativa de alta confianza que estamos tratando de construir. Además, estas condiciones nos permiten crear un sistema de trabajo cada vez más seguro, en el que todos nos ayudamos mutuamente a alcanzar nuestros objetivos, traspasando los límites necesarios para conseguirlo.