Engineer

revisión de código

~~ɓə~~ examinar un cambio ajeno antes de incorporarlo al código

Qué es

Una revisión de código es el paso en el que alguien distinto de quien escribió un cambio lo lee antes de que pase a formar parte de la base de código. La guía de prácticas de ingeniería de Google la define como el examen del código por cualquier persona distinta de quienes lo escribieron, y el capítulo sobre el tema de *Software Engineering at Google* añade que suele ocurrir antes de incorporar el código a la base de código. La mayoría de los equipos habla de esta práctica en plural, revisiones de código; los nombres cambian según la herramienta. En GitHub es una revisión de una pull request, en Google una revisión precommit y en el kernel de Linux se realiza por correo electrónico, donde quien lo revisa ofrece una etiqueta Reviewed-by si está conforme con el parche.

La guía de Google enumera lo que se examina: diseño, funcionalidad, complejidad, pruebas, nombres, comentarios, estilo y documentación. Detrás de cada palabra hay una pregunta. ¿Hace el código lo que pretendía su autor y resulta útil para quienes lo usan? ¿Podría ser más sencillo? ¿Hay pruebas automáticas y son correctas y están bien diseñadas? ¿Se ha actualizado la documentación que lo describe?

La mecánica es muy parecida en todas partes. Se envía un diff con una descripción, las personas que revisan comentan líneas concretas, quien lo escribió responde o sube una versión nueva y el ciclo se repite hasta que alguien acepta el cambio. En Google, esa aceptación se llama LGTM, abreviatura de *looks good to me* («me parece bien»), y el libro la describe como un bit de permisos: de forma predeterminada basta un LGTM, que, junto con la aprobación de un responsable del código y una comprobación de legibilidad, permite registrar el cambio en un commit. GitHub termina cada revisión con una de tres decisiones. Comment aporta observaciones sin aprobar ni pedir cambios; Approve indica que el cambio puede fusionarse tal como está; y Request changes señala puntos que deben atenderse antes. Cualquier persona con acceso de lectura al repositorio

puede revisar y comentar, y quienes lo administran pueden exigir una aprobación para fusionar. La guía de Google llama CL a cada cambio, de ahí que una de sus páginas se titule Small CLs.

Utilice el concepto en una decisión

Aplique «¿Qué es una revisión de código?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Un equipo que lleva una idea del boceto a un producto al que sus clientes pueden acceder, sin pasarla entre tres proveedores.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Ingresos multiplicados por 10 con mapas GIS — 10x crecimiento de ingresos impulsado desde el producto cartográfico

Ingresos multiplicados por 10 con mapas GIS Póngase en contacto →

Por qué importa

El libro enumera seis beneficios de una revisión bien llevada. Comprueba que el código sea correcto, asegura que otros ingenieros puedan entender el cambio, impone coherencia en toda la base de código, fomenta la responsabilidad compartida del equipo, difunde conocimientos y deja un registro histórico. Que el código sea correcto es el beneficio evidente, y los autores citan un estudio de IBM, publicado en 1986, que observó que los defectos detectados antes requerían menos tiempo para corregirse. El libro añade la condición decisiva: el tiempo invertido en revisar solo compensaba mientras el proceso se mantuviera ligero. El análisis estático y las pruebas automáticas asumen hoy buena parte de la comprobación de que el código es correcto, lo que, según el libro, ha reducido la necesidad de confiar esa tarea a personas.

La comprensión es el beneficio que ninguna herramienta sustituye. Por competente que sea una persona, no puede leer su propio trabajo como lo haría alguien ajeno; quien revisa sí puede. A menudo es la primera vez que alguien distinto del autor comprueba que un cambio tiene sentido. Esto importa porque el código se lee muchas más veces de las que se escribe. Los beneficios culturales son menos visibles. La revisión recuerda que el código no es propiedad individual, sino parte de un proyecto compartido; permite a los especialistas transmitir lo que saben sobre un área; y, como cada cambio conserva su revisión, cualquiera puede averiguar después cuándo apareció un patrón y leer la revisión que lo introdujo.

Revisar también consume tiempo, y revisar despacio cuesta más. La guía de Google sobre la rapidez de las revisiones indica que las revisiones lentas ralentizan a todo el equipo, los desarrolladores empiezan a sentir rechazo hacia el proceso y la calidad del código empeora, porque se presiona para dejar pasar cambios de menor calidad y se desincentivan las tareas de limpieza. Añade que la mayoría de las quejas sobre una revisión demasiado estricta desaparece cuando la misma persona simplemente responde antes. Su criterio de aprobación es pragmático: cuando un cambio deja claramente el código del sistema en mejor estado que antes, conviene inclinarse por aprobarlo, aunque no sea perfecto. La guía lo considera el principio de mayor rango entre sus reglas y resume el motivo en una frase: «no existe el código “perfecto”; solo existe código mejor».

Cómo se aplica en la práctica

Piense en una tienda cuyos precios se almacenan en peniques enteros. Se añade una campaña de primavera y se envía este cambio a revisión:

```
--- a/shop/pricing.py
+++ b/shop/pricing.py
@@ -8,2 +8,4 @@ def order_total(basket):
     subtotal = sum(line.price * line.quantity for line in basket.lines)
+    if basket.voucher == "SPRING":
+        subtotal = subtotal * 0.9
     return subtotal + shipping_for(basket)
```

Se dejan cuatro comentarios:

```
shop/pricing.py, línea 9
  Los demás cupones y sus tasas de descuento están en VOUCHERS,
  en shop/vouchers.py. ¿Podría ir ahí también SPRING? Así, la próxima
  campaña requeriría cambiar los datos, no esta función.

shop/pricing.py, línea 10
  En el resto de este módulo, los precios son peniques enteros, y
  multiplicarlos por 0.9 puede dejar una fracción de penique. Los otros
  descuentos pasan por round_money() en shop/money.py.

shop/pricing.py, línea 11
  Nit: el envío queda fuera del descuento. Si es intencionado,
  un breve comentario que lo indique ayudaría a quien lo lea después.

General
```

```
No hay ninguna prueba con un cupón en la cesta. ¿Podría añadir
una para SPRING y otra que muestre que un código desconocido
no cambia nada?
```

Cada comentario trata del código y no de la persona, y cada uno explica por qué. Los dos primeros y el último no llevan etiqueta, así que se espera que se atiendan; el tercero empieza por Nit:, que la guía de Google utiliza para un detalle menor que quien escribió el código debería corregir, aunque tenga poca repercusión. Sus otras etiquetas son Optional o Consider, para una buena idea que no es obligatoria, y FYI, para una observación sobre la que no se espera actuar en ese cambio. Sin etiquetas, advierte, quien escribió el código puede interpretar todos los comentarios como obligatorios. En GitHub, esta ronda termina con Request changes. Se traslada SPRING a la tabla, se hace pasar el descuento por la función auxiliar de redondeo, se añade el comentario y ambas pruebas y se solicita otra revisión; la segunda ronda termina con Approve o, en Google, con LGTM. La guía pide responder a una solicitud de revisión en un día laborable como máximo, sin interrumpir una tarea que requiere concentración, sino esperando a una pausa natural.

Ninguno de los comentarios trata del formato, el orden de las importaciones o si las pruebas pasan, porque una máquina debería haberlo indicado antes de que se abriera el diff. El libro describe comprobaciones presubmit que se ejecutan al enviar por primera vez un cambio a revisión y pueden rechazarlo antes de que nadie lo lea; el kernel pide ejecutar su comprobador de estilo, `scripts/checkpatch.pl`, antes de enviar un parche. El motor de comprobaciones obligatorias creado para un repositorio, creado en el marco de un puesto desempeñado desde 2025, lleva la idea muy lejos: 268 comprobaciones de commit registradas, 277 reglas de lint y 15 reglas de ESLint propias, además de 146 pruebas de las propias comprobaciones. Comprueban, por ejemplo, que una migración pueda revertirse y que una clave de traducción exista en ambas configuraciones regionales, y cada commit marcado como corrección debe incluir una prueba que falle sin la corrección. Las comprobaciones codificaban reglas mecánicas, de modo que quienes revisaban el código podían dedicar más tiempo al diseño. También declara el coste: crear commits es lento y resulta exasperante tener que sortear una comprobación mal escrita.

Dónde surgen los problemas

El cambio es demasiado grande para leerlo. La guía de Google considera que 100 líneas suelen ser un tamaño razonable y que 1.000 suelen ser demasiadas, aunque deja la decisión a quien revisa; señala también que 200 líneas en un

archivo pueden estar bien mientras que las mismas 200 repartidas entre 50 archivos quizá no. Los cambios pequeños se revisan más deprisa y a fondo y tienen menos probabilidades de introducir errores; el libro añade que se puede rechazar justificadamente un cambio por ser demasiado grande para una sola revisión. Divida el trabajo: encadene cambios pequeños, envíe grupos de archivos a distintas personas, separe las refactorizaciones de las funciones nuevas y las correcciones y mantenga las pruebas de cada cambio dentro del mismo cambio.

La revisión queda pendiente durante días. Cada día de espera retrasa el trabajo que depende del cambio, y responder cada varios días con peticiones importantes hace que el proceso parezca más duro de lo que es. Responda en un día laborable; el libro espera observaciones en 24 horas laborables y, si la revisión completa debe esperar, se debe confirmar que se ha visto el cambio. Envíe todos los comentarios juntos en lugar de por partes: pocas cosas molestan más que una ronda nueva de observaciones ajenas a la anterior justo después de haberla atendido.

El gusto personal se presenta como un requisito. Quien prefiere otro bucle u otro nombre y bloquea el cambio por ello enseña a dejar de mejorar el código. La guía de Google fija el orden: los hechos y los datos pesan más que la opinión o el gusto personal; en cuestiones de estilo, la guía de estilo tiene la última palabra; y, si varios enfoques son igualmente sólidos, elige quien escribió el cambio. Etiquete las preferencias como Nit u Optional para distinguirlas de los requisitos.

El comentario trata de la persona. El ejemplo de comentario malo de la propia guía pregunta a quien escribió el código por qué usó hilos si es evidente que la concurrencia no aporta ningún beneficio; la versión buena indica que el modelo de concurrencia añade complejidad sin ninguna mejora de rendimiento que quien revisa pueda apreciar, así que el código debería usar un solo hilo. Haga comentarios sobre el código, nunca sobre la persona que lo desarrolla, y explique el razonamiento de la petición.

Las personas comprueban lo que podría comprobar una máquina. Quienes dedican su atención a los espacios y al orden de las importaciones pasan por alto la cuestión de diseño que solo una persona puede resolver. Traslade cada regla que pueda expresarse por escrito a un linter, un formateador o una comprobación presubmit y reserve para la revisión las decisiones que requieren criterio.

La respuesta se queda en la herramienta de revisión. Se pregunta para qué sirve un bloque de código, se explica en el hilo, se aprueba el cambio y la siguiente persona que lee el código nunca ve la explicación. La guía de Google dice que una pregunta así normalmente debería llevar a reescribir el código con mayor claridad o, al menos, a añadir un comentario en él. El kernel hace la misma observación a quienes envían parches: cuando una pregunta de revisión no cambia el código, la respuesta, por regla general, debe acabar en un comentario o en el registro de cambios.

Lecturas adicionales

- Introducción — la página inicial de la guía de revisión de Google: qué se examina, cómo elegir revisores y cuándo la programación en pareja cuenta como revisión.
- El criterio de revisión de código — el principio de mayor rango, la orientación mediante revisiones y cómo resolver conflictos entre quien escribe y quien revisa.
- CL pequeños — por qué los cambios pequeños reciben mejores revisiones, qué tamaño es excesivo y cuatro maneras de dividir un cambio.
- Revisión de código — el capítulo de Tom Manshreck y Caitlin Sadowski en *Software Engineering at Google*: los beneficios, LGTM y la responsabilidad sobre el código, y las buenas prácticas.
- Revisiones de pull requests — la referencia de GitHub para revisar, aprobar y solicitar cambios y para exigir aprobaciones antes de fusionar.
- Envío de parches: guía esencial para incorporar código al kernel — cómo revisa por correo el kernel de Linux, qué declara una etiqueta Reviewed-by y cómo se debe responder a quienes revisan.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es una revisión de código?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/code-review/