Engineer

refactorización

🛠️ Cambiar la estructura del código sin alterar lo que hace

Qué es

La refactorización consiste en cambiar la estructura interna del código sin cambiar lo que hace para quien lo utiliza. Cambian los nombres, los límites de las funciones y la organización de los módulos; las entradas, las salidas y los efectos secundarios permanecen exactamente como estaban. El verbo es refactorizar, y uno de esos cambios también se llama refactorización.

Martin Fowler define el término en su libro Refactoring y volvió a exponer la definición en su sitio web en 2004. Como sustantivo, una refactorización es un cambio en la estructura interna del software que hace el código más fácil de entender y menos costoso de modificar, mientras su comportamiento visto desde fuera permanece igual. Como verbo, refactorizar es realizar una secuencia de esos cambios manteniendo ese comportamiento externo durante todo el proceso. Ambas partes importan. La finalidad es abaratar el siguiente cambio, y la restricción es que nadie ajeno al código pueda advertir que ha ocurrido algo.

Fowler marca claramente la diferencia: «Refactorizar no es otra forma de decir limpiar el código», escribe en refactoring.com. Nombra una técnica específica. Utiliza reestructuración como término más amplio para reorganizar código por cualquier medio, incluidos los que no conservan el comportamiento en cada paso. Lo que hace de la refactorización una técnica es su tamaño. Cada paso es una transformación pequeña que hace poco por sí sola y tiene pocas probabilidades de salir mal, y el sistema completo sigue funcionando después de cada uno. Una secuencia larga de estos pasos puede aun así sumar un gran cambio de estructura.

Los pasos tienen nombre. El catálogo de refactoring.com, que acompaña a la segunda edición del libro, los enumera con los nombres anteriores a los que sustituyen: Extract Function, extraer una función, también conocido como

Extract Method, y su inverso Inline Function, insertar el cuerpo de la función en sus puntos de uso; Inline Variable, sustituir una variable por su expresión; Rename Variable, renombrar una variable; Move Function, mover una función; Split Phase, separar fases, y decenas más. Un nombre compartido permite a un desarrollador decir en dos palabras qué cambio hizo y a quien revisa contrastarlo con un procedimiento conocido.

Utilice el concepto en una decisión

Aplique «¿Qué es la refactorización?» 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 software que tiene éxito sigue cambiando. Las funcionalidades nuevas y las correcciones se superponen, y cada una tiende a dificultar un poco la siguiente hasta que, como describe Fowler, añadir cualquier cosa nueva lleva un tiempo penosamente largo. La refactorización es el contrapeso: remodela el código para que cada incorporación no añada una complejidad innecesaria. Su beneficio se mide en el coste del siguiente cambio, no en el aspecto del código.

Por eso Fowler la trata como parte de la programación habitual en lugar de como una tarea que aparece en un plan de proyecto. Distingue tres momentos. Antes de añadir una funcionalidad, comprueba si el código tiene una forma que facilite incorporarla y, si no, refactoriza primero, algo que suele resultarle más rápido que introducirla a la fuerza. Cuando la funcionalidad funciona, refactoriza el resultado para que quien vuelva unas semanas después pueda seguirlo sin desconcierto. Y cuando tiene que leer código desconocido para entenderlo, lo refactoriza para que el siguiente lector no tropiece en el mismo lugar. El primero de esos momentos tiene su propio nombre, refactorización preparatoria, y Fowler atribuye a Kent Beck la regla que lo sustenta: para cada cambio, primero remodele el código hasta que el cambio sea fácil, lo que puede ser la parte difícil, y solo entonces realícelo.

La refactorización también mantiene bajo el riesgo de una forma que el trabajo en funcionalidades no puede. Fowler transmite otra metáfora de Beck: los dos sombreros. Con el sombrero de refactorizar, cada cambio conserva el comportamiento y las pruebas siguen pasando durante todo el proceso. Al añadir funcionalidad, se añaden pruebas y fallan las existentes, lo que genera más tensión y riesgo. Un desarrollador que sabe qué sombrero lleva sabe qué significa una prueba fallida: durante una refactorización, significa que el último paso fue incorrecto.

La misma idea permite saldar la mayor parte de la deuda técnica. El artículo de Google sobre la deuda oculta en sistemas de aprendizaje automático sitúa refactorizar el código en primer lugar entre las formas de reducirla. Las herramientas ayudan: muchos lenguajes tienen hoy editores que automatizan refactorizaciones comunes. Fowler considera esas herramientas valiosas, pero no imprescindibles, y cuando faltan recurre a pasos pequeños y pruebas frecuentes.

Cómo se aplica en la práctica

Tomemos una función escrita para este artículo que calcula el total de un pedido e imprime un recibo en el mismo lugar:

```
function printReceipt(order) {
  let total = 0;
  for (const line of order.lines) {
    total += line.price * line.quantity;
  }

  // print the receipt lines
  console.log(`Order ${order.id}`);
  console.log(`Total: ${total}`);
}
```

Llega un requisito nuevo: el mismo total debe aparecer en una factura. En lugar de copiar el bucle, el desarrollador refactoriza primero. El primer paso es Extract Function: el bucle pasa a su propia función, con un nombre que indica lo que calcula, y se ejecutan las pruebas. El segundo extrae la impresión de la misma manera, de modo que el nombre de una función sustituye al comentario, y vuelven a ejecutarse las pruebas. El tercero es Inline Variable: la variable `total` ahora solo se usa una vez, por lo que la llamada ocupa su lugar. Después de cada paso, el programa hace exactamente lo mismo que antes, y cada paso podría confirmarse por separado:

```
function printReceipt(order) {
  printLines(order.id, orderTotal(order));
}

function orderTotal(order) {
  let total = 0;
  for (const line of order.lines) {
    total += line.price * line.quantity;
  }
  return total;
}

function printLines(id, total) {
  console.log(`Order ${id}`);
  console.log(`Total: ${total}`);
}
```

Solo ahora se incorpora la funcionalidad, y consiste en una línea del código de la factura que llama a `orderTotal` . Ninguno de los tres pasos de refactorización necesitó una prueba nueva porque no cambió nada observable; lo demuestran las pruebas que ya cubrían el recibo.

El código de infraestructura también se refactoriza, y allí hay más en juego porque los nombres hacen referencia a recursos en ejecución. La documentación de Terraform advierte de que, de forma predeterminada, dar una dirección nueva a un recurso en la configuración indica a Terraform que destruya el existente y construya otro nuevo. Por eso, renombrar `aws_instance.a` como `aws_instance.web` para facilitar su lectura sustituiría un servidor. Desde Terraform v1.1, un bloque `moved` registra en cambio la dirección anterior, y el siguiente plan actualiza la dirección sin destruir nada:

```
resource "aws_instance" "web" {
  # (configuration unchanged)
}

moved {
  from = aws_instance.a
  to   = aws_instance.web
}
```

Para versiones anteriores, la documentación remite al comando `terraform state mv` para efectuar el mismo traslado. En ambos casos, la refactorización solo es segura una vez leído el plan y comprobado que muestra un traslado y no una

destrucción y una creación.

Mantener esos cambios separados del trabajo en funcionalidades resulta más fácil cuando el historial indica cuál es cuál. La norma de commits descrita en la entrada del portafolio de este sitio sobre aplicar una misma norma de historial a cuatro repositorios empieza cada mensaje de commit con un prefijo convencional que nombra el tipo de cambio, exigido por un hook de mensajes de commit. La página, sobre trabajo realizado en julio de 2026, cuenta 26 refactorizaciones en el historial del repositorio más grande, junto a 156 commits de funcionalidades y 133 correcciones. La misma página reconoce que el hook comprueba el formato y no el contenido. Un commit etiquetado como refactorización sigue teniendo que demostrar mediante sus pruebas y su revisión que no ha alterado el comportamiento.

Dónde surgen los problemas

Los cambios de comportamiento se cuelan con la refactorización. Un desarrollador ordena una función y corrige un error en ella en el mismo commit. Cuando falla una prueba, nadie sabe cuál de las dos cosas lo causó, y quien revisa debe leer cada línea como posible cambio de comportamiento. Lleve un solo sombrero cada vez: refactorice en un commit, cambie el comportamiento en el siguiente y, si falla una prueba al refactorizar, deshaga el último paso en lugar de seguir añadiendo correcciones.

Una gran reestructuración se llama refactorización. Una rama pasa semanas con el proceso de construcción fallando mientras el código «se está refactorizando». Eso es reestructuración en los términos de Fowler y conlleva el riesgo que la refactorización está diseñada para evitar. Divida el trabajo en pasos que dejen el sistema funcionando y puedan fusionarse por separado; donde no sea posible, llámelo reescritura y planifíquelo como tal.

No hay pruebas que demuestren que nada cambió. Conservar el comportamiento es una afirmación, y sin pruebas es una suposición. Antes de tocar código sin pruebas, escriba pruebas que fijen lo que hace hoy, incluso cuando parezca incorrecto, y corrija después sus rarezas en cambios separados. Aquí se aplica la regla orientativa de Fowler en su ejemplo de refactorización preparatoria: cuando sienta la necesidad de ejecutar el código e inspeccionar visualmente su salida, escriba una prueba en su lugar.

Renombrar en el código de infraestructura sustituye el recurso. Un nombre nuevo más claro en Terraform parece una refactorización al revisarlo, y el plan propone discretamente destruir una base de datos y crear otra vacía. Registre cada cambio de nombre y cada traslado hacia dentro o fuera de un módulo mediante un bloque `moved`, y considere cualquier destrucción en el plan de un cambio que se pretende refactorización como un paso fallido.

Se elimina un bloque `moved` antiguo. Los bloques moved se acumulan y quitarlos parece limpieza. La documentación de Terraform lo llama un cambio incompatible: cualquier configuración que aún utilice la dirección antigua obtendría entonces un plan que destruye el objeto en lugar de moverlo. Conserve los bloques históricos, encadénelos cuando un objeto se renombre dos veces y elimínelos únicamente de un módulo privado cuando todos sus usuarios hayan aplicado la versión que los contiene.

La refactorización espera una fase dedicada. Un equipo que programa un «sprint de refactorización» para más adelante sigue añadiendo funcionalidades a un código que se resiste, y es fácil recortar un sprint previsto para después cuando cambia una fecha límite. El consejo de Fowler va en sentido contrario: refactorice como parte de cada cambio, primero para facilitarlo y después para dejar claro el resultado, de modo que ese trabajo nunca necesite un hueco propio en el plan.

Lecturas adicionales

- Definición de refactorización — Las definiciones de Martin Fowler como sustantivo y como verbo, tal como aparecen en su libro.
- Refactoring — El sitio de Fowler sobre el libro, con las razones para integrar la refactorización en la programación diaria y el catálogo completo.
- Extract Function — La entrada del catálogo de una de las refactorizaciones más comunes, con su inversa y su nombre en la primera edición.
- Un ejemplo de refactorización preparatoria — Fowler remodela su propio código en pasos pequeños antes de añadir una funcionalidad y explica los dos sombreros.
- Refactorizar módulos — La documentación de Terraform sobre bloques `moved`, cambios de nombre, división de módulos y por qué deben conservarse los bloques antiguos.

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 la refactorización?» 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/refactoring/