Engineer

Deuda técnica

Coste futuro de una solución rápida adoptada hoy

Qué es

La deuda técnica es el coste futuro de un atajo en el diseño o el código de un sistema: una decisión que ahorra tiempo hoy y hace que los cambios posteriores sean más lentos, más arriesgados o imposibles hasta deshacer el atajo. Los equipos suelen abreviar technical debt como tech debt. La palabra es una metáfora y se eligió porque el atajo se comporta como un préstamo. Permite obtener algo ahora y lo cobra después, mientras siga ahí.

La idea procede del informe de experiencia que Ward Cunningham presentó en OOPSLA en 1992 sobre WyCash, un sistema de gestión de carteras escrito en Smalltalk. Un equipo pequeño mantenía un conocimiento general de unos cuatro megabytes de código fuente, y las partes maduras del programa se habían revisado o reescrito muchas veces. Esa consolidación, sostiene el informe, hacía posible un crecimiento incremental constante; su ausencia supone un peligro serio: código que funciona y satisface al cliente, pero que nunca se ha puesto en orden. Cunningham lo resumió en una frase: «Publicar código tal como se escribió la primera vez es como endeudarse». El propio informe solo habla de deuda y de su analogía con la deuda. Trabajos posteriores utilizan el nombre de dos palabras en inglés, y un artículo de Google de 2015 atribuye la metáfora a Ward Cunningham y la fecha en 1992.

Los investigadores que se reunieron en Dagstuhl en abril de 2016 para acordar qué abarca el término redactaron una definición de trabajo. En ella, la deuda es un conjunto de elementos de diseño o implementación que ahorran esfuerzo a corto plazo, pero dejan el sistema en un estado que encarece los cambios posteriores o hace que algunos queden fuera de alcance. Limitan su impacto a las cualidades internas del sistema, sobre todo a la facilidad para mantenerlo y hacerlo evolucionar. Esto separa la deuda de una funcionalidad que nadie ha construido: el coste de la deuda recae en quienes modifican el sistema y solo llega a los usuarios a través de ellos. El mismo informe conserva el vocabulario

financiero. El capital es lo que ahorró el atajo o lo que costaría hoy una solución mejor. Hay dos tipos de intereses: los recurrentes, pagados en forma de menor velocidad de desarrollo, defectos inducidos y pérdida de calidad, y los que se acumulan, el coste adicional de cada nueva pieza de software construida sobre un código que no está del todo bien. La deuda reside sobre todo en el código, pero también en la documentación, las pruebas y los defectos conocidos.

Utilice el concepto en una decisión

Aplique «¿Qué es la deuda técnica?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La estructura del sistema se define primero sobre el papel para que las piezas encajen y los costes se conozcan antes de escribir código.

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. Un plan de infraestructura para 14 departamentos — 14 departamentos de investigación con un único plan de infraestructura

Un plan de infraestructura para 14 departamentos Póngase en contacto →

Por qué importa

Asumir deuda deliberadamente puede ser una buena decisión. El informe de Cunningham dice que una pequeña cantidad permite a un equipo avanzar más rápido, siempre que se liquide pronto reescribiendo el código, y el artículo de Google coincide en que endeudarse suele ser una buena elección estratégica. Ambos advierten de la deuda que se deja sin resolver. En el informe, los intereses se pagan en tiempo, con cada minuto perdido por un código que no está del todo bien; una implementación que nadie consolida puede volverse tan pesada que paralice organizaciones enteras de ingeniería. Antes de llegar a ese punto, el programa se convierte en algo que ninguna persona domina por completo; aparecen programadores que solo conocen una parte y, finalmente, un producto incapaz de cambiar de rumbo.

El problema es que a menudo la deuda no se elige. El informe de Dagstuhl observa que surge con frecuencia sin que nadie lo pretenda y toma a los desarrolladores por sorpresa; por eso se descubre sobre todo durante el mantenimiento, mucho después de la decisión que la creó. Las causas son corrientes: la presión del calendario, la ausencia de una persona clave o la falta de información suficiente sobre una característica técnica en aquel momento. El artículo de Google añade que la deuda oculta es la peligrosa porque genera intereses compuestos silenciosamente.

La metáfora también da a desarrolladores y responsables de gestión un lenguaje común. El informe de Dagstuhl la considera un mecanismo potente de comunicación, y la visión que plantearon sus participantes incluye gestionar la deuda con el mismo cuidado que los defectos y las funcionalidades nuevas, con una forma de relacionar las preocupaciones de los desarrolladores con las decisiones de los responsables. Un estudio presentado en el seminario muestra qué puede significar en un equipo. Graziela Tonin, Alfredo Goldman y Carolyn Seaman pidieron a equipos de estudiantes, con 60 personas en total, que identificaran y siguieran explícitamente su deuda. Una lista visible permitió negociar con el cliente tiempo para refactorizar, dejó constancia de qué partes del proyecto seguían inmaduras y mostró si mejoraba la calidad del código. La mayoría de los miembros dijeron que se comunicaban más, pensaban más si una deuda era realmente necesaria y refactorizaban con mayor frecuencia. Los autores añaden que, al debatir las decisiones de asumir deuda, algunos equipos acabaron con menos expertos a quienes nadie más pudiera sustituir.

Cómo se aplica en la práctica

La deuda se vuelve gestionable cuando queda escrita donde el equipo planifica su trabajo. Una entrada de registro, redactada para este artículo, puede ser tan breve como esta:

```
TD-14       Los totales de facturas se calculan en tres lugares
Artefactos: billing/web, billing/export, reports/monthly
Causa:      fecha de lanzamiento fija; copiar la regla era más rápido que
            trasladarla a un módulo compartido
Capital:    unos 2 días para trasladar la regla a un solo módulo, con pruebas
Intereses:  cada cambio de regla fiscal se aplica y prueba tres veces;
            en uno de los últimos cinco cambios se omitió una copia
Siguiente:  saldar al próximo cambio de regla fiscal; la exportación de facturas
            electrónicas prevista este trimestre no debe ser la cuarta copia
A cargo de: equipo de facturación
```

Los campos siguen el modelo del informe de Dagstuhl: un elemento vinculado a artefactos concretos, con una causa, un capital e intereses. Escribir los intereses hace útil la entrada porque convierte la sensación de que el código de facturación está desordenado en un coste que se puede valorar. Si las reglas fiscales cambian cuatro veces al año y cada cambio cuesta un día adicional de trabajo y pruebas, los dos días de capital se recuperan en un máximo de seis meses. La exportación prevista representa intereses que se acumulan en el sentido de Dagstuhl: cada pieza nueva construida sobre la regla duplicada

aumenta el precio de corregirla después. Si, por el contrario, el código estuviera congelado y nadie esperara volver a tocarlo, la misma entrada justificaría dejar la deuda como está.

La deuda no es solo cuestión de código. La entrada del portafolio sobre la revisión de procesos internos que ahorró 8.000 horas documenta el trabajo anterior de la fundadora en Mappitall. Pregunta si la deuda técnica está frenando al lector y describe la que se acumula en torno a la forma de trabajar: una arquitectura de software que había crecido por acumulación en vez de por diseño, sistemas que funcionaban pero no eficientemente y una planificación que dejaba a las personas esperando o sobrecargadas. No había ninguna urgencia, y por eso se había dejado todo como estaba. Revisar la arquitectura, los sistemas y la planificación ahorró unas 8.000 horas durante el trabajo realizado entre 2015 y 2022; la página atribuye parte de ese resultado a hacer que los cambios perduraran en la forma de operar del equipo, porque las mejoras que no se consolidan acaban volviendo a la situación anterior.

Dónde surgen los problemas

A todo se le llama deuda técnica. Cuando la expresión se populariza, cualquier molestia recibe esa etiqueta: una biblioteca antigua, una funcionalidad que falta o el estilo de un compañero. El informe de Dagstuhl identifica expresamente ese riesgo y pide definiciones precisas para afrontarlo. Reserve el término para los atajos cuyo coste recaiga en los cambios futuros, registre los defectos como defectos y los deseos como funcionalidades, y rechace una entrada que no pueda identificar su causa y sus intereses.

La deuda solo existe en la cabeza de la gente. Quien toca el código después paga la deuda que nadie escribió, normalmente sin saber por qué tarda tanto el trabajo. Mantenga un registro junto a la lista de trabajo pendiente y consúltelo antes de modificar una parte del sistema, como hicieron los equipos de estudiantes del estudio de Tonin con su lista. Un elemento solo se vuelve visible para quienes deciden cómo se asigna el tiempo cuando está escrito donde miran.

Se devuelve capital donde no se cobran intereses. Limpiar un módulo que nadie modifica parece productivo y no aporta nada, mientras que el código que cambia todas las semanas sigue cobrando. Los intereses solo se acumulan donde se trabaja, así que ordene el registro según la frecuencia con que cambia cada parte y lo que se vaya a construir sobre ella, y liquide primero los elementos caros.

El pago espera a una gran reescritura. La deuda abandonada durante años suele afrontarse con un proyecto para sustituir todo el sistema, que asume nueva deuda mientras el antiguo sigue cobrando. Cunningham aconsejaba pagar pronto, y el artículo de Google enumera las formas habituales: refactorizar, reforzar las pruebas unitarias, eliminar código muerto, reducir dependencias, acotar mejor las API y mejorar la documentación. Ninguna añade una funcionalidad; el objetivo es abaratar el siguiente cambio. Prográmelas como parte del trabajo ordinario, cerca del cambio que las necesita.

Los experimentos se quedan en producción. Probar una alternativa como rama condicional del código de producción cuesta poco en cada experimento, y el artículo de Google describe cómo se acumulan esas ramas hasta que probar todas las combinaciones se vuelve difícil o imposible. Cita Knight Capital, cuyo sistema, según el artículo, perdió 465 millones de dólares en 45 minutos, aparentemente porque rutas de código experimental obsoleto se comportaron de forma inesperada. Revise periódicamente cada rama experimental y cada indicador de funcionalidad, elimine lo que ya no se utilice y valore tanto eliminar código como añadirlo.

Se confunde la puntuación de una herramienta con la deuda. El análisis estático puede contar líneas duplicadas y funciones largas, y un panel hace que la cifra parezca un saldo. El artículo de Google deja claro que la metáfora no proporciona una métrica estricta que seguir a lo largo del tiempo y que un equipo que aún avanza rápido no demuestra tener poca deuda. En Dagstuhl, Forrest Shull informó de que ningún conjunto único de reglas para detectar deuda se correlacionaba de forma consistente con sus síntomas, como un coste de cambio creciente y una velocidad decreciente, en proyectos distintos. Contraste los hallazgos de una herramienta con su propio historial de cambios y defectos, conserve las reglas que predigan costes y descarte las demás. Las propias preguntas del artículo son una prueba mejor, entre ellas cuánto se tarda en poner al día a un miembro nuevo del equipo.

Lecturas adicionales

- El sistema de gestión de carteras WyCash — el informe de experiencia de Ward Cunningham en OOPSLA ‘92, donde aparece por primera vez la analogía de la deuda.
- Gestión de la deuda técnica en la ingeniería de software — el informe del seminario Dagstuhl 16162, con la definición de trabajo, el capital y los intereses y el estudio de listas explícitas de deuda.

- Deuda técnica oculta en sistemas de aprendizaje automático — el artículo de 2015 de Sculley y sus colegas de Google sobre cómo se acumula la deuda, cómo reducirla y por qué se resiste a la medición.
- Refactoring — el sitio de Martin Fowler sobre los pequeños cambios que conservan el comportamiento y que el artículo de Google sitúa en primer lugar entre las formas de reducir deuda.

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 deuda técnica?» 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/technical-debt/