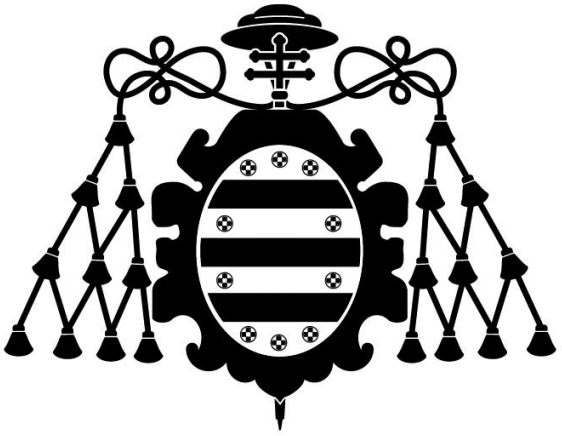
PRESENTACIÓN GRUPO 4 S2

EP. 481. La deuda técnica

Daniel Alberto Alonso Fernández
Javier Novella Tomás
Manuel Palacios Sánchez



Universidad de Oviedo

Introducción-Ipez Ozkaya

Ipez Ozkaya es la invitada del episodio 481 de Software Engineering Radio. Ipez es la Directora Técnica en el Instituto de Ingeniería de Software Carnegie Mellon, desarrolla técnicas para mejorar el desarrollo de software, la eficiencia y la evolución del sistema, trabaja con organizaciones gubernamentales e industriales para mejorar sus prácticas de arquitectura de software, recibió un doctorado en diseño computacional de la Universidad Carnegie Mellon y es la coautora del libro ***Managing Technical Debt: Reducing Friction in Software Development.*** *Junto con* Philippe Kruchten y Robert Nord.

Qué es la deuda técnica

Básicamente es un término enunciado por Ward Cunningham en 1992 que se refiere al costo del retrabajo adicional por elección de una solución más rápida en lugar de la más efectiva. Podría extrapolarse a la elección entre dos opciones y las consecuencias que va a suponer la elección de una de ellas frente a la otra.

Principio 1

Cosificar un concepto abstracto.

En muchos casos se utiliza el término “deuda técnica” como una idea abstracta en la que englobamos diferentes problemas dentro del software. En este primer principio los autores quieren dejar claro que esta idea inicial debe ser cambiada, y hay que tratar a la deuda técnica como un concepto real para poder llegar a la raíz del problema. Además también es importante mencionar que el objetivo de llegar al origen de esa dificultad no es buscar al responsable para echarle la culpa, sino intentar encontrar la mejor solución y aprender de los errores.

Principio 2

Si ninguno de tus intereses sufre consecuencias, probablemente no tengas deuda técnica.

La deuda técnica solo es real en un sistema que cree valor o interés. Aquellos procedimientos cuya deuda técnica no afecta al rendimiento o al interés final del sistema no deben recibir de forma prioritaria una reducción del problema.

Principio 3

Todos los sistemas tienen deuda técnica.

Contradiciendo el punto anterior, los autores comentan que todos los sistemas tienen deuda técnica. Esta deuda técnica no tiene por qué ser real y materializable, puede ser una deuda técnica potencial. Una deuda técnica es considerada potencial siempre que no afecte directamente al desempeño o intereses del sistema, pero que dependiendo del rumbo de distintos factores podría acabar desembocando en una deuda técnica real. Esos factores suelen ser producidos por la incertidumbre al futuro. ¿Qué rumbo querrá tomar nuestro “Product Owner”?, ¿Nuestro sistema será compatible con futuras tecnologías?.

Principio 4

La deuda técnica debe remontarse al sistema.

Todo lo que va mal en mi sistema a causa de decisiones pasadas, es deuda técnica. Por ello se debería poder señalar y aislar la parte concreta del sistema que esté acarreando esta deuda, para así contemplar la posibilidad de minimizarla en medida de lo posible o calcular el retrabajo a hacer en caso de asimilar la deuda.

Una vez localizada la parte del proyecto que genera deuda el procedimiento a seguir pasaría por plantear una alternativa, calcular el trabajo que llevaría cambiarla y ver si es asumible. Normalmente se trata de enfocarse en la arquitectura para disminuir dependencias y aumentar la encapsulación.

Comparando por ejemplo monolitos y microservicios, no es algo tangencial: habría que tener otros factores en cuenta particulares de nuestro proyecto. Como cita la invitada "Dependency explosion happens everywhere".

Principio 5

La deuda técnica no es sinónimo de mala calidad.

Basándonos en el principio fundamental de que la deuda surge de la elección de la arquitectura y de las decisiones tomadas, y de que toda decisión conlleva una parte de deuda, no podríamos determinar si la calidad de un software tiene que ver con su cantidad de deuda. Es decir, una deuda no es mala de por sí, sino que depende sobretodo del fin o utilidad de nuestro software, y si el sacrificio de deuda que hacemos es algo relatable o totalmente despreciable en el contexto.

Un ejemplo podría ser la elección de base de datos Heroku o MongoDB, escojamos la opción que escojamos, la deuda técnica va a estar ahí, lo realmente importante es la magnitud de esta en el caso concreto de nuestro proyecto, el “cúanto nos afecta” la decisión y que conlleva.

La deuda en cierto modo es cuantificable, no ya tanto con un valor, si no más bien de forma aproximada o simplemente de forma objetiva. También podemos abstraernos y pensar en términos de valor económico, tiempo…Pero como tal no hay una unidad de medida absoluta de la deuda.

Principio 6

La deuda técnica tiene el costo de propiedad más alto.

Partimos de la base de que un costo de propiedad no tiene que ser malo (se refiere a lo que cuesta algo, se puede asumir o no, bien sea por viabilidad o por variedad de alternativas).

Siempre, en cualquier proyecto, se trabaja de manera directa o indirecta sobre una acumulación de errores, incluso llegando al punto de capas y capas sobre una implementación erronea, principalmente debido a decisiones de diseño o arquitectura. La reelaboración sobre esta pila acumulada puede conllevar un costo superior.

Este principio, en esencia, trata de resaltar la fuerte relación entre las opciones técnicas y el diseño arquitectónico con respecto a la deuda técnica. Según el podcast: “si desea un alto costo de propiedad, diseñelo mal”.

Principio 7

Todo el código es importante.

-Código de pruebas: esta parte es importante, ya que puede prevenir muchos futuros problemas. Si realizando las pruebas bajamos la dureza del puntaje para que estas pasen, esto puede generar más desventajas que ventajas en un futuro.

-Código externo: un proyecto puede estar con un mal funcionamiento debido a un código que no sea propio. Este código proveniente de APIs de proveedores externos puede provocar problemas o crasheos a todo el código, y simplemente parchear los problemas y no arreglarlos de raíz puede provocar más desventajas.

-Software heredado: el cual puede acarrear problemas de funcionamiento y de seguridad.

Ejemplo de la vida real, desastre del Challenger.

Principio 8

La deuda técnica no tiene una medida absoluta ni de interés.

Este principio trata de explicar que no hay ninguna forma de medir la deuda técnica, ni un programa que mida los costes de realizar un software frente a otro. Existen diagramas en el libro que muestran diferentes tipos de tamaños de sistemas y los tipos de deuda técnica en los que podría incurrir. Esta es la forma fundamental con la que se analizan las consecuencias y el desarrollo ideal.

Principio 9

La deuda técnica depende de la evolución futura del sistema.

Este principio consiste en la toma de decisiones y evolución en función de nuestro uso anticipado del sistema. Esto hace que podamos evaluar en función de ese trabajo anticipado, las partes con deuda técnica y eliminarlas no teniendo que pagar por ellas en el futuro.

Al elegir un diseño o una arquitectura, esto nos limita el futuro desarrollo y afecta en el impacto de la deuda técnica. Puede llegar el punto, en el que plantearse si es posible continuar con las limitaciones del diseño o es necesario relajarlo rediseñando el sistema.