Curso de Maestría "Deuda Técnica y Calidad de Software CSDT"
2022-1

Gestión de la deuda técnica

Ing. Edwin Fernando Vasquez Molina
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá/Colombia
edwin.vasquez-m@mail.escuelaing.edu.co

Abstract. La deuda técnica (TD) es un concepto que ha tomado fuerza en el mundo IT, es un término relativamente desconocido pero del cual se tiene muy buena documentación, este ha evolucionado e incursionado en las metodologías de software. Incorporándose en todas las fases y generando valor al producto. En este artículo se describirán términos asociados a la DT y mecanismos, herramientas o formas de gestionar la deuda técnica, aprendidas en el curso.

Palabras clave—deuda técnica; tipos de deuda técnica; ciclo de vida; software; gestión; mantenimiento del software, Github, Github Actions.

I. Introducción

La deuda técnica es una metáfora que representa en el ciclo de vida de un proyecto de software todas las omisiones, rezagos o posibles costos a futuro al no aplicar o realizar una actividad, esta puede ser actividades en: desarrollo, análisis, gestión o simplemente en comunicación. También se puede entender este concepto con lo que nos dice Martin Fowler *"Technical Debt es una metáfora, acuñada por Ward Cunningham, que enmarca cómo pensar, en cómo lidiar con este cruft, considerándolo como una deuda financiera. El esfuerzo adicional que se necesita para agregar nuevas funciones es el interés que se paga sobre la deuda."[1]*. La deuda técnica se puede entender, aplicar o crear definición desde cualquier punto de vista organizacional ya que es fácil adecuar el término.

Teniendo en cuenta lo anterior, a continuación se describirán algunos conceptos esenciales para entender la DT, ejemplos y un caso práctico desarrollado en el transcurso del curso, estos nos permitirán dar un abrebocas de todo lo que concierne a la deuda técnica en el mundo del desarrollo de software.

II. Antecedentes

Todo proyecto de software o metodología que se utilice para gestionar un proyecto tiene una secuencia de pasos y guías que ayudan al equipo a la toma de decisiones importantes, estos pasos permiten que se cree y se estructure todo el proyecto en hitos, metas u objetivos de largo plazo y así mismo en tareas más pequeñas e iterativas, todas en busca de cumplir ese hito y por supuesto en generar valor al producto final. En todo este proceso o ciclo de vida del software es donde nace o se genera la DT, en las tomas de decisiones o estructuración del proyecto existe la posibilidad de no tener claridad o entendimiento de lo que se necesita y que esto en futuro presente incidentes o updates al proyecto, también la DT surge porque en este flujo se puede permitir omitir a conciencia algunos procesos que en el momento no son necesarios aplicarlos, pero

en un futuro se convertirán en deuda técnica que ya se tiene contemplada.

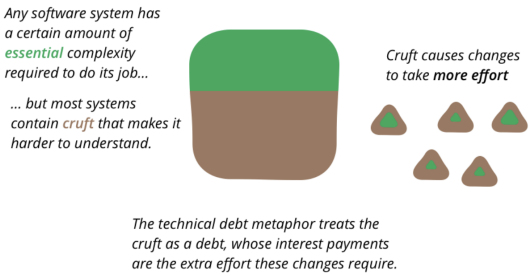

Fig. 1. Representación gráfica de la metáfora de la deuda técnica, hecha por Martin Fowler[1]

Teniendo en cuenta que la DT tiene un proceso de creación y que es importante poder listarla y tenerla en cuenta en el flujo del proyecto, nace la necesidad de poder tipificar esta deuda técnica y de acuerdo con "Towards an Ontology of Terms on Technical Debt" los tipos de deuda técnica se han identificado de acuerdo con los resultados de un mapeo bibliográfico sistemático[2], y a partir de estas definiciones se ha determinado su relación entre ellos.

Tabla 1.

Listado de tipos de deuda técnica

Tipo de DT
Deuda de Arquitectura (Architecture Debt)
Deuda de Construcción (Build Debt)
Deuda de Código (Code Debt)
Deuda de Defectos (Defect Debt)
Deuda de Diseño (Design Debt)
Deuda de Documentación (Documentation Debt)
Deuda de Infraestructura (Infrastructure Debt)
Deuda de Personas (People Debt)
Deuda de Procesos (Process Debt)
Deuda de Requisitos (Requirement Debt)
Deuda de Servicios (Service Debt)
Deuda de Pruebas (Test Debt)
Deuda de Automatización de Pruebas (Test Automation Debt)

III. Aplicación de la deuda técnica

En el transcurso de 12 semanas y más de 20 sesiones de trabajo, pudimos aplicar de varias maneras mecanismos que ayudan a gestionar los tipos de deuda técnica, siempre teniendo como referencia la definición y marco teórico de cada tipo de DT.

La idea era poder aplicar estos mecanismos de gestión a un código fuente público y que pudiéramos evidenciar la evolución en cada sesión, también poder recolectar información, conclusiones y posibles mejoras a estos procesos, siempre teniendo como guía lo que se detalla de la DT.

Para este caso seleccionamos un proyecto Java "WBT"[3] que está alojado en un repositorio Github, el código fuente cumple con la función de un juego de escritorio arcade.

Detalle del proceso de aplicación

Una vez se tuvo elegido el código fuente, iniciamos sesión por sesión la creación de archivos .md (Readmes), en estos documentos detallamos la aplicación iterativa de cada unas de las herramientas o formas de mitigación de la DT.

❖ *Deuda técnica de código*

En las primeras sesiones del curso identificamos y aprendimos cómo gestionar este tipo de deuda técnica, la deuda de código se puede decir o entender como todas aquellas malas prácticas que hacen que la mantenibilidad y entendimiento del mismo sea ineficiente.

En este caso iniciamos con el análisis del código fuente y la identificación de "code smell" presente en el código, así mismo listamos las posibles prácticas de refactoring aplicables al proyecto.

algunos ejemplos encontrados son:

1. Code smell-Comments: este code Smell dice; Un método está lleno de comentarios explicativos.
2. Code Smell-LongMethod: este code Smell dice; Un método contiene demasiadas líneas de código. En general, cualquier método de más de diez líneas debería hacer que comiences a hacer preguntas
3. Refactoring-Notación (CamelCase)
4. Refactoring Replace Magic Number with Symbolic Constant

❖ *Revisión de buenas prácticas:*

En las sesiones posteriores pudimos identificar en nuestro código que técnicas y buenas prácticas de Clean Code o principios programación y prácticas XP se ven aplicadas o cuales no.

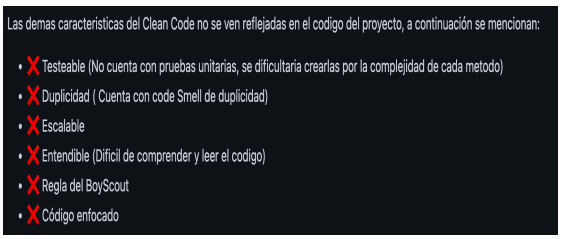

Fig. 2. Fragmento del archivo "Clean Code + XP Practices", prácticas de clean code no aplicadas.

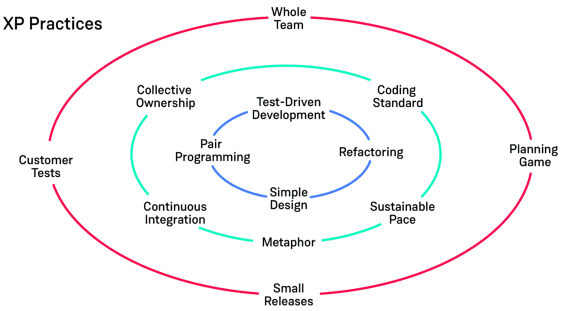

Fig. 3. Imagen del círculo de las prácticas XP.

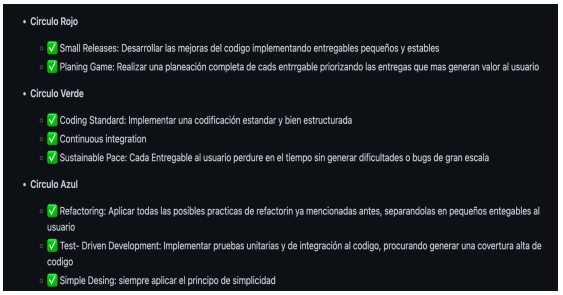

Fig. 4. Fragmento del archivo "Clean Code + XP Practices", aplicación de las prácticas Xp en el código.

❖ *Deuda técnica de pruebas*

La deuda técnica de pruebas la abordamos con la identificación en el proyecto, de pruebas unitarias automatizadas y así mismo con la aplicación construcción de una prueba unitaria.

Todo con el objetivo de entender la importancia en proyecto de la construcción de todo el esquema de pruebas, mejorando la calidad del producto final y la mantenibilidad del código.

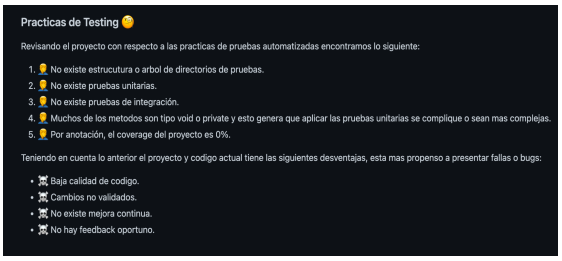

Fig. 5. Fragmento del archivo “Testing Debt”,identificación de aplicación de pruebas.

IV. Conclusiones

La información y documentación que se tiene actualmente de la DT es muy extensa, además es difícil poder llevarla a la práctica y poder entenderla. pero considero que la metodología que se utilizó sesión por sesión propicio a que se pudiera implementar un caso práctico y que esto ayudará a resolver dudas de la información teórica.

Como sabemos poder librarnos de la deuda técnica es algo imposible, pero es importante resaltar que existen muchos mecanismos manuales y automatizados que nos ayudan con la mitigación, podemos adelantarnos y poder evitar que tengamos conflictos a futuro en nuestros proyectos.

Por último resaltar que es importante como personas que estamos involucrados en el mundo IT que debemos conocer y poder identificar cuando podemos tener deuda técnica o cuando la estamos generando y así poder armar planes de acción para poder mitigarla.

Referencias

1. Fowler, M. (s/f). TechnicalDebt. martinfowler.com. Recuperado el 29 de abril de 2022, de https://martinfowler.com/bliki/TechnicalDebt.html
2. Alves, N. S. R., Ribeiro, L. F., L., Caires, V., Mendes, T. S., Spínola, R. O., (2014). “Towards an Ontology of Terms on Technical Debt”
3. Molina, E. F. V. (s/f). CSDT-2022 de https://github.com/IngEdwinV/wbt/blob/codeSmellRefactoring/CSDT-2022-1.md?classId=c16b95df-3b93-4ea9-9b67-762ec3212fd1&assignmentId=437c34b9-61a5-46eb-bbc2-fb5d8721520c&submissionId=1a4809f4-a59b-4fd8-90a8-b53faebf4dbe