Deuda Técnica y Calidad de Software CSDT
2022-1

Gestión de la deuda técnica y calidad de software

Ing. Wilmer Alexander Viatela Bravo
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá / Colombia
wilmer.viatela-b@mail.escuelaing.edu.co

Abstract. *La deuda técnica es un término utilizado para describir las consecuencias generadas debido a las malas prácticas de desarrollo de software. Este documento compila el trabajo realizado durante el programa de calidad de software y gestión de deuda técnica con el objetivo de reconocer los tipos de deuda técnica, analizar el impacto generado por esta, estudiar las mejores prácticas para reducir su impacto negativo y aplicar correctamente los principios de calidad durante las diferentes etapas del desarrollo de software.*

1. Introducción

Deuda Técnica (TD) ha sido el término usado para referirse a distintos problemas presentados durante el ciclo de desarrollo de software causados por la ejecución de malas prácticas durante las etapas de este ciclo. Estos problemas pueden generar errores en el funcionamiento del software y también reprocesos para la corrección de estos errores.[2]

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, y a partir de estas definiciones se ha determinado su relación entre ellos. Este documento es un ejemplo del formato de presentación deseado, y contiene información concerniente al diseño general del documento, familias tipográficas, y tamaños de tipografía apropiados. [1]

2. Tipos de Deuda Técnica

La deuda técnica se ha clasificado en diversos tipos de acuerdo con el proceso del ciclo de vida del desarrollo de software, así [1]:

2.1. Deuda de Arquitectura (Architecture Debt)

Esta deuda hace referencia a los problemas encontrados que pueden afectar los requisitos arquitectónicos del proyecto.

2.2. Deuda de Construcción (Build Debt)

Esta deuda hace referencia a los problemas relacionados con la construcción del proyecto, que generan dificultades y aumentos de tiempo y procesos inneceriamente.

2.3. Deuda de Código (Code Debt)

Esta deuda hace referencia a los problemas encontrados en el código fuente del proyecto que afectan negativamente su mantenimiento.

2.4. Deuda de Defectos (Defect Debt)

Esta deuda hace referencia a los defectos conocidos del proyecto, que pueden ser identificados con los procesos de pruebas realizados, y que no son corregidos a tiempo generando así reprocesos y retrabajo posterior.

2.5. Deuda de Diseño (Design Debt)

Esta deuda hace referencia a las prácticas usadas en el desarrollo del proyecto que no cumplen con los principios de un buen diseño orientado a objetos.

2.6. Deuda de Documentación (Documentation Debt)

Esta deuda hace referencia a los problemas encontrados en la documentación del proyecto, ya sea por documentación inexistente, inadecuada de cualquier tipo.

2.7. Deuda de Infraestructura (Infrastructure Debt)

Esta deuda hace referencia a los problemas de infraestructura que pueden dificultar o generar retrasos en las actividades de desarrollo del proyecto.

2.8. Deuda de Personas (People Debt)

Esta deuda hace referencia a los problemas de personas que pueden dificultar o generar retrasos en las actividades de desarrollo del proyecto, por ejemplo experiencia, curvas de aprendizaje, retrasos en contratación, entre otros.

2.9. Deuda de Procesos (Process Debt)

Esta deuda hace referencia a los procesos ineficientes establecidos a lo largo del ciclo de desarrollo de producto.

2.10. Deuda de Requisitos (Requirement Debt)

Esta deuda hace referencia a los problemas encontrados procedentes del levantamiento de requisitos para el proyecto, por ejemplo que los requisitos no se implementan en su totalidad, se presentan falencias en el levantamiento, no se cumple con todo el alcance del producto.

2.11. Deuda de Servicios (Service Debt)

Esta deuda hace referencia a los problemas encontrados en la selección, elaboración y funcionamiento de los servicios requeridos en el proyecto.

2.12. Deuda de Pruebas (Test Debt)

Esta deuda hace referencia a los problemas encontrados en las actividades de pruebas que pueden generar reprocesos y afectar la calidad de las pruebas por ende del producto final.

2.13. Deuda de Automatización de Pruebas (Test Automation Debt)

Esta deuda hace referencia al trabajo que implica la automatización de las pruebas del proyecto para de esta manera apoyar los procesos de integración continua y demás ciclos del desarrollo.

3. Proyecto de investigación y práctica de deuda técnica

Para identificar la deuda técnica se ha seleccionado el proyecto de código “The Gilded Rose Kata”[3] donde iterativamente se ha progresado en la implementación de buenas prácticas para la mitigación de los impactos relacionados a los diferentes tipos de deuda técnica encontrados.

3.1. Deuda Técnica de Código

El primer tipo de deuda técnica identificado y gestionado en el proyecto es la deuda de código fuente, este tipo de deuda afecta negativamente la lectura ágil del código generando dificultad en su mantenimiento y evolución. Para avanzar en la mitigación de la deuda técnica de código se inicial con la ejecución de un análisis preliminar para identificar “Code Smells” en el código fuente del proyecto seleccionado y las prácticas de Refactoring que puedan aplicarse a dicho proyecto. Con este análisis preliminar realizado se encuentran varios Code Smells en el código fuente del proyecto como:

- Long Methods: La clase principal del proyecto contiene toda la información y métodos requeridos para el desarrollo del proyecto.
- Change Preventers: La actualización de una clase puede generar cambios en los métodos funcionales del proyecto.
- Dispensables: Algunos de los métodos incluidos en las clases principales podrían modificarse para obtener un código más limpio.

También se identifican técnicas de Refactoring para aplicar en el código fuente del proyecto, entre ellas se encuentran:

- Extract Variable
- Extract Method
- Substitute Algorithm
- Change Reference to Value
- Consolidate Conditional Expression
- Add Parameter
- Rename Method
- Rename Variable
- Remove Dead Code
- Update Notation

Para complementar el análisis para la mitigación de la deuda de código se trabaja en la identificación de prácticas de Clean Code, Principios de Programación y Prácticas XP presentes en el proyecto seleccionado y también las que pueden aplicar para reducirla deuda técnica y mejorar la calidad de código de este.

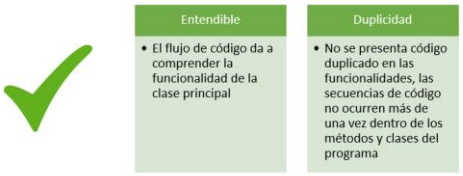

Fig. 1. Prácticas de Clean Code presentes en el proyecto

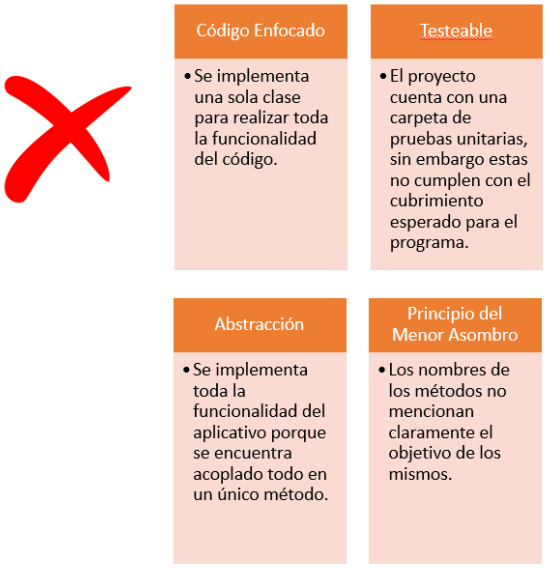

Fig. 2. Prácticas de Clean Code que no se cumplen en el proyecto

Estos principios de programación, Yagni (You Aren´t Gonna Need It) y KISS (Keep it Simple, Stupid), no se están implementando correctamente en el código fuente del proyecto.

```
###### Inicio - Fragmento de Código ######
if item.quality < 50:
                    item.quality = item.quality + 1
                    if item.name == "Backstage passes to a TAFKAL80ETC concert":
                        if item.sell_in < 11:
                            if item.quality < 50:
                                item.quality = item.quality + 1
                        if item.sell_in < 6:
                            if item.quality < 50:
                                item.quality = item.quality + 1
###### End - Fragmento de Código ######
```

Fig. 3. Fragmento de código Principio KISS

Se identifican Prácticas XP como Test Driven Development y Continuous Integration, entre otras, para aportar a la mejora de la calidad del código del proyecto.

3.2. Deuda Técnica de Pruebas

En el proyecto se ha presentado deuda técnica de pruebas que afecta el coverage de pruebas unitarias en el proyecto, a pesar de que existe un archivo de pruebas, las pruebas unitarias no son las adecuadas para el proyecto, los estándares de codificación usados para las pruebas no cumplen los requisitos mínimos para la ejecución y faltan escenarios de pruebas. Para mitigar el impacto de esta deuda se ha propuesto la implementación de prácticas de TDD y también se crean escenarios de pruebas unitarias para mejorar el coverage de pruebas.

3.3. Análisis Estático de Código e Integración Continua

Se realiza la implementación de la herramienta Better Code Hub para realizar un análisis estático de código en el proyecto seleccionado verificando el cumplimiento de diez pautas necesarias para la creación de software mantenible. Una vez implementado el análisis se evidencia que se cumplen 5 de las 10 pautas necesarias y se presentan las novedades que complementan los estudios realizados con anterioridad para revisar la deuda técnica en código del proyecto.

Para la práctica de integración continua se realiza implementan workflow de GitHub para ejecutar el proceso de integración en distintas versiones del lenguaje de programación y realizar un análisis de seguridad del proyecto incrementando así su calidad.

3.4. Deuda de Arquitectura

Para iniciar la correcta gestión y mejora de la deuda de arquitectura se inicia con la identificación de Architectural Smells, una vez hecho este análisis se encuentran diversas novedades que reducen la calidad y mantenibilidad del sistema como:

- No Layers
- The Grand Old Duke of York
- The Grand Old Duke of York

Se propone un modelo de arquitectura para el proyecto y se inicia con la implementación de ATAM (Architecture Trade-Off Analysis Method) como método de evaluación de arquitecturas.

Modelo Preliminar de Arquitectura

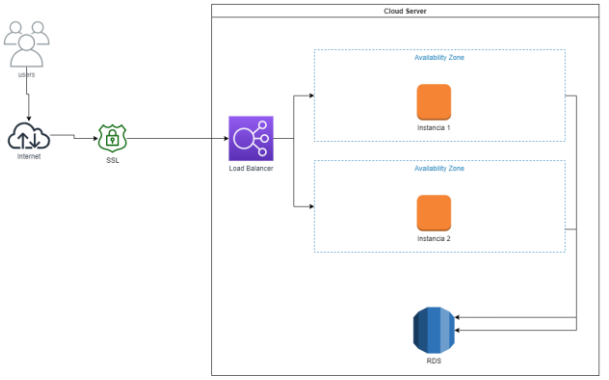

Fig. 4. Modelo de arquitectura

Con la implementación de este método evaluamos las decisiones arquitectónicas en relación con los atributos de calidad y las metas de negocio. También se realiza QAW (Quality Attribute Workshops) para identificar los atributos de calidad que serán drivers del diseño de arquitectura del producto facilitando así la detección temprana de conflictos y mejora de los requerimientos iniciales para el sistema.

Scenario Refinement for Scenario N		
Scenario(s):		Se necesitan ajustar las reglas de calidad y caducidad de los artículos del inventario y asegurar la calidad del sistema.
Business Goals:		Modificabilidad del Sistema, Calidad del Sistema
Relevant QualityAttributes:		Modificabilidad, Testabilidad
Scenario Components	**Stimulus:**	Por normas de la empresa se debe actualizar todo el modelo de reglas de calidad y caducidad
	Stimulus Source:	Modificación en reglamento de la empresa
	Environment:	El usuario administrador debe actualizar los parametros del sistema
	Artifact (If Known):	Sistema de inventario
	Response:	La actualización de los parámetros es realizada de manera ágil en los archivos de configuración y se ejecuta set de pruebas
	Response Measure:	Se actualizan las reglas y el sistema continua funcionando con normalidad
Questions:		¿Se necesitan actualizar las pruebas unitarias del sistema con esta actualización? ¿Cómo se toman backups del sistema en la arquitectura provista?
Issues:		Es posible que se deba actualizar la arquitectura para asegurar los backups del sistema

Fig. 5. Ejemplo de QAW's

4. Conclusiones

La profundización en los estudios relacionados con la deuda técnica es relativamente reciente, este documento describe un caso práctico donde se implementan algunas prácticas y actividades para apoyar a los equipos en la mitigación del impacto de la deuda técnica y la mejora continua de la calidad del producto y procesos de todo el ciclo de desarrollo de software.

Es casi imposible evitar la deuda técnica, intentar mitigarla tampoco es una tarea fácil, sin embargo, el continuo avance en la tecnología y la implementación de nuevas prácticas ágiles en el ciclo de desarrollo de software nos permite actuar con rapidez y mitigar los posibles errores que esta puede generar en nuestros proyectos.

Referencias

[1] 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”

[2] Kruchten, P., Nord, R, L., Ozkaya, I. (2012). “Technical Debt: From Metaphor to Theory and Practice”

[3] Bache, E. “Gilded Rose Refactoring Kata”. https://github.com/emilybache/GildedRose-Refactoring-Kata