Recibido 29 Jul. 2023 *ReCIBE, Año 12 No.2, NOV. 2023*

Aceptado 28 JUL. 2023

Deudas técnicas en el sistema transaccional

Technical debts in the transactional system

Javier Fabricio Cucalón Gaibor[1]
Jacobo Sandoval Gutierrez[1]

[1]Universidad Autónoma Metropolitana

Resumen

En las empresas internacionales de servicios financieros transaccionales ocasionalmente se realizan actualizaciones de programa con errores no identificables por el compilador. Estos errores definidos como deuda técnica provocan fallas en el sistema transaccional al no cumplir el modelo de calidad. Las investigaciones consultadas no han reportado la existencia de nuevas métricas para mitigar las fallas por las deudas técnicas. Por lo anterior, el objetivo de la investigación realizada fue obtener un conjunto de nuevas métricas para incorporarlas al modelo de calidad. La metodología consistió en recopilar una muestra de programas, definir variables y métricas, y evaluar la causa-efecto para encontrar las deudas técnicas. El experimento consistió en obtener de dos empresas las muestras de los programas modificados en el software BASE24-eps, se definieron cinco métricas. El resultado mostró que los programas tienen una mayor probabilidad de encontrar al menos una deuda técnica relacionadas a bug y code smell. La investigación, permite concluir la importancia de incorporar métricas al proceso de calidad.

Palabras clave: Deuda técnica, Métrica, Modelo de Calidad.

Abstract

In international transactional financial services companies, program updates are occasionally made with errors not identifiable by the compiler. These errors, defined as technical debt, cause failures in the transactional system by not complying with the quality model. The investigations consulted have not reported the existence of new metrics to mitigate failures due to technical debts. Therefore, the research objective was to obtain new metrics to incorporate into the quality model. The methodology consisted of compiling a sample of programs, defining variables and metrics, and evaluating the cause-effect to find the technical debts. The experiment obtained samples of modified programs in the BASE24-eps software from two companies, and five metrics were defined. The result showed that the programs have a higher probability of finding at least one technical debt related to bug and code smell. This research allows concluding the importance of incorporating metrics into the quality process..

Keywords: Metrics, Quality model, Technical debt.

1. Introducción

Las empresas internacionales de servicios financieros transaccionales realizan modificaciones sobre productos de software para poder implementar reglas que cumplan las especificaciones de sus negocios. El proceso de desarrollo de software inicia desde la definición del requerimiento hasta el despliegue del cambio en el ambiente productivo, sin embargo, se ha encontrado con frecuencia que dentro del proceso se cumple con el modelo de calidad y aún así, se siguen generando errores conocidos como deudas técnicas en los programas modificados (Gartner, 2023; SONAR, 2023).

Uno de los productos de software es B24-eps, sus ventajas radican en ser un software de clase mundial que esta operando en todos los continentes del mundo, en diversos tipos de empresas como instituciones financieras, bancos, redes de integración de servicios financieros, retails y otras empresas que necesitan integrar medios de pago para servicio de sus clientes y asi estos puedan realizar transacciones financieras a través de los distintos canales de pago como ATMs, POSs, Tellers, botones de pago, webservices, entre otros (ACI-Worldwide, 2023). Las deudas técnicas se explican como los errores, deficiencias y carencias que tiene el código modificado por acciones deliberadas o involuntarias de los programadores, estos no son errores de compilación (Stopford, Wallace, y Allspaw, 2017; Kruchten, Nord, y Ozkaya, 2012). Hay 3 tipos de deudas técnicas, la primera son los bugs, los cuales generan de forma aleatoria las interrupciones en la ejecución de los programas provocando una ausencia temporal o permanente de la disponibilidad del producto de software y en consecuencia una pérdida de dinero para las empresas al no poder ofrecer los servicios a sus clientes. La segunda son los code smells, los cuales generan una complejidad en el mantenimiento futuro de los programas provocando altos costos para las empresas debido al mayor tiempo que los programadores necesitan para realizar las modificaciones. Y la tercera son las reglas de seguridad, las cuales provocan condiciones de vulnerabilidad ante los ataques informáticos en el producto de software y por ello, se aumentan los costos en las empresas por las pérdidas de información o la ausencia temporal o permanente de los servicios a sus clientes (Alves y cols., 2016).

La familia de normas ISO/IEC 25000, conocida como System and Software Quality Requeriments and Evaluation (SQuaRE) es un conjunto de normas cuyo objetivo es la creación de un marco de trabajo común para realizar la evaluación de calidad del producto de software. Esta se conforma por las siguientes divisiones, ISO/IEC 2500n División de Gestión de la Calidad, ISO/IEC 2501n División de Modelo de Calidad, ISO/IEC 2502n División de Medición de Calidad, ISO/IEC 2503n División de Requisitos de Calidad y ISO/IEC 2504n División de Evaluación de Calidad.

Las normas ISO/IEC 25000 contienen la norma internacional ISO/IEC 25010 que provee un modelo de calidad para productos de software, categorizando sus propiedades en ocho características que son: adecuación funcional, eficiencia de rendimiento, compatibilidad, usabilidad, confiabilidad, seguridad, mantenibilidad y portabilidad. Cada característica esta compuesta por un conjunto de sub características relacionadas (ISO, 2014).

1.1. Proceso de modificación

B24-eps es un software que implementa reglas de negocio del sistema de servicios financieros transaccionales basados en los estándares mundiales y así las empresas pueden operar su ecosistema de medios de pago con los cuales trabajan. Sin embargo, en cada región se pueden implementar reglas de negocio particulares, por ejemplo, por normativas del país correspondiente, por las políticas propias de la empresa que tienen el objetivo de aprovechar al máximo los canales transaccionales existentes o simplemente por la implementación de nuevos modelos transaccionales.
Por este motivo, es necesario realizar modificaciones a los programas del software B24-eps que permita implementar estas reglas de negocio particulares que no son parte del producto base. En el proceso de desarrollo de las reglas de negocio en los programas B24-eps se tienen tres fases de trabajo. Las dos primeras fases son: fase de modificación que logra implementar la regla de negocio solicitada en el requerimiento, luego de esto, viene la fase de compilación de los programas modificados, en donde se logran identificar aquellos errores que no cumplen con las reglas de programación del compilador C++. Lo anterior es un proceso iterativo que finaliza cuando se obtiene un programa modificado que cumple con la regla de negocio solicitada y con cero errores de compilación.
En la tercera fase se verifica el cumplimiento del modelo de calidad. El proceso consiste en comparar que ciertas líneas de código del programa modificado coincidan con las métricas de calidad. Las métricas de calidad son un conjunto de sintaxis con diferentes formatos como pueden ser letras, números, longitudes de dígitos, rangos, entre otros o reglas que deben ser escritas en los programas modificados para cumplimiento de la métrica (Atlassian, 2023; Pavlič, Hliš, Heričko, y Beranič, 2022; Mamun, Martini, Staron, Berger, y Hansson, 2019).

1.2. Solución a las deudas técnicas

La norma internacional ISO/IEC 25010 no es aplicable completamente para todos los tipos de productos de software, algunas características o subcaracterísticas no son aplicables o faltan otras nuevas dentro de la norma. Por ejemplo, la subcaracterística de instalabilidad es usualmente no aplicable al tipo de software web services, porque estos son ejecutados remotamente en el lado del servidor. Y el caso de B24-eps sucede algo similar, esta norma no es totalmente aplicable para este producto de software.
En consecuencia, para cubrir la falencia de la norma internacional ISO/IEC 25010 algunos productos de software han desarrollado, a partir de este, sus propios modelos de calidad que le permitan identificar la deuda técnica de los programas modificados. Por consiguiente, B24-eps no tiene su propio modelo de calidad de software.
Una de las herramientas Open Source de análisis de código estático mas comúnmente usada tanto en la academia como en la industria es SonarQube (Saarimaki, Baldassarre, Lenarduzzi, y Romano, 2019). Esta herramienta se puede obtener como un servicio desde la plataforma sonarcloud.io o puede ser descargada e instalada en un servidor privado. SonarQube permite el análisis de los programas verificando que el código cumpla con la reglas establecidas en las métricas . En caso de una violación a cualquiera de las reglas se genera el reporte de una deuda técnica

(Li, Soliman, y Avgeriou, 2022). Al conjunto de métricas para un tipo de lenguaje de programación o producto de software se lo conoce como modelo de calidad. Actualmente SonarQube es usado por 7M+ de usuarios y 400k+ de empresas en los campos de la salud, IT, gobierno, educación, energía, telecomunicaciones, industriales, seguros, etc. alrededor del mundo (Alfayez, Winn, Alwehaibi, Venson, y Boehm, 2023; Katin, Lenarduzzi, Taibi, y Mandić, 2022).

2. Método

El método se concibió de la siguiente manera: La primera etapa consistió en identificar un conjunto de archivos que han sido actualizados o modificados, que no tengan errores de compilación y que cumplan con las métricas estándar del modelo de calidad actual. Seleccionar una muestra representativa aleatoria de la población identificada. Estudiar las líneas de código modificadas por el programador y comparar que cada modificación del programa se encuentre homologada. En caso, de no estar homologada se considera como una probable deuda técnica y por tanto,
se agregará a una lista de métricas. Finalmente, el resto de los archivos serán auditados con las nuevas métricas para obtener una estadística.
Para el trabajo experimental, se eligió como base la norma ISO/IEC 25010 y se tuvo acceso a los archivos en producción de dos diferentes empresas, es decir, aquellos que superaron las pruebas de compilación y se encuentran en ejecución. En total 273 programas fueron recopilados, el 46% corresponde a la extensión *.h y 54% a *.cpp. De estos programas se propuso un intervalo de confianza del 98%, un error de 0.5 y una desviación estándar de 1.5 para conocer $n = 50$ como el número de muestra aleatorias para validar el experimento. Del conjunto muestra se realizó el análisis de las líneas de código para poder desglosar los siguientes criterios.

- Líneas de código totales por tipo de programa. Se refiere a la cantidad de líneas que tiene el programa en texto plano, sin importar si el compilador las utiliza o no.
- Líneas de código efectivas por programa. Son las líneas que el compilador interpreta para interactuar en el sistema.
- Líneas de código comentadas por programa. Son todas las líneas de orientación para el programador que no son utilizadas por el compilador, pero sirven para conocer la evolución del programa.
- Cambios realizados en las líneas de código efectiva. Es la comparación del número de cambios realizados para cumplir una modificación y su repercusión en líneas de código efectivo.
- Cambios realizados en las líneas de código comentadas. Es la comparación del número de cambios realizados para cumplir una modificación y su repercusión en líneas de código comentada.
- Listado de las métricas. Son características no homogéneas que potencialmente son o serán una deuda técnica.

El procedimiento realizado a cada programa seleccionado de forma aleatoria se agrupo por análisis a las líneas de código, análisis de los cambios de código y el cambio no homogéneo, tal cual se muestra en la Figura1.

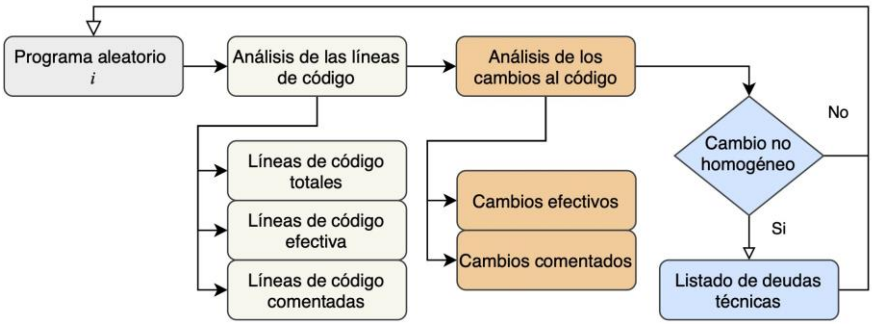

Figura 1: Análisis de los programas para encontrar las deudas técnicas

Los análisis de líneas de código son tareas cuantificables por la sintaxis del propio programa. Sin embargo, los análisis a los cambios al código son tareas que requieren la solicitud de una orden de trabajo y su comparación entre los cambios realizados y que tengan diferencias mínimas para el mismo objetivo o llamado de la función. Adicionalmente, detectar un cambio no homogéneo requiere agrupar las órdenes de trabajo similares y después compararlas entre sí. Estos últimos dos tipos de análisis, requieren de programadores experimentados en el lenguaje técnico y operativo de los sistemas financieros.
Para determinar si las extensiones de los programas son diferentes entre sí, se realizó una prueba *t* con dos medias para comprobar la distribución de toda la población. Además, se realizaron tres pruebas de regresión para determinar la correlación entre los criterios analizados:

- Número de cambios vs líneas de código totales.
- Número de cambios vs líneas de código efectivo.
- Número de cambios vs líneas de código comentado.

Después de obtener la lista métricas de deuda técnica y las relaciones que puedan existir entre los programas. Se realizó un distribución de probabilidad para los escenarios más probables. Y con ello, determinar las futuras deudas que seguirán apareciendo.

3. Resultados

El número de líneas de código por cada tipo de documento estuvieron en un rango desde 65 hasta 12,017 con una media de 957 para la extensión *.h y de 122 hasta 10,783 con una media de 1,211 para *.cpp. El resultado de la prueba para toda la población considerando un intervalo de confianza 95%, con 271 grados de libertad cuando t0 >t1 (0.05 >1.65), se determinó, que no hay diferencia sustancial entre los dos tipos de extensiones.

Para el análisis comparativo entre las líneas de código efectivo independientemente del tipo de extensión se obtuvo un promedio de 94.5% en comparación con el código comentado del 5.5%. El resultado de la prueba para toda la población considerando un intervalo de confianza 95%, con 271 grados de libertad cuando t0 >t1 (0.05 >1.65), se determinó, que no hay diferencia entre los dos tipos de documentos.
En las pruebas de correlación para los casos de líneas de código totales y efectivas en comparación con el número de cambios se obtuvo un coeficiente de correlación de 0.19 y 0.17 respectivamente. Sin embargo, el coeficiente de correlación fue de 0.96 para el caso de líneas comentadas por el número de cambios. En la Figura 2 se muestran la relación del número de cambios en comparación con las líneas de código comentadas para el caso de la extensión CPP.
De los documentos revisados se tuvieron 5 métricas con inconsistencias en los documentos de la muestra. Las características que deberían estar homogéneas son las siguientes métricas:

- Métrica No. 1 - Token Registry Metric: Su existencia y valor a TRUE garantiza que las funciones que setean los valores desde los tokens a los TDEs o desde los TDEs a los tokens en formato binario o carácter y en procesamiento de solicitud o respuesta reemplacen a las funciones del producto en su versión original.
- Métrica No. 2 - Class Prefix Metric: El nombre de las clases customizadas deben empezar con la palabra csm_. Esto garantizará la diferencia con las clases del producto en su versión original.
- Métrica No. 3 - SSID Event Metric: Es obligatorio que el SSID de los eventos de una clase customizada este entre los valores 150-299 incluídos. Esto garantizará la convivencia de las clases customizadas y las del producto en su versión original.

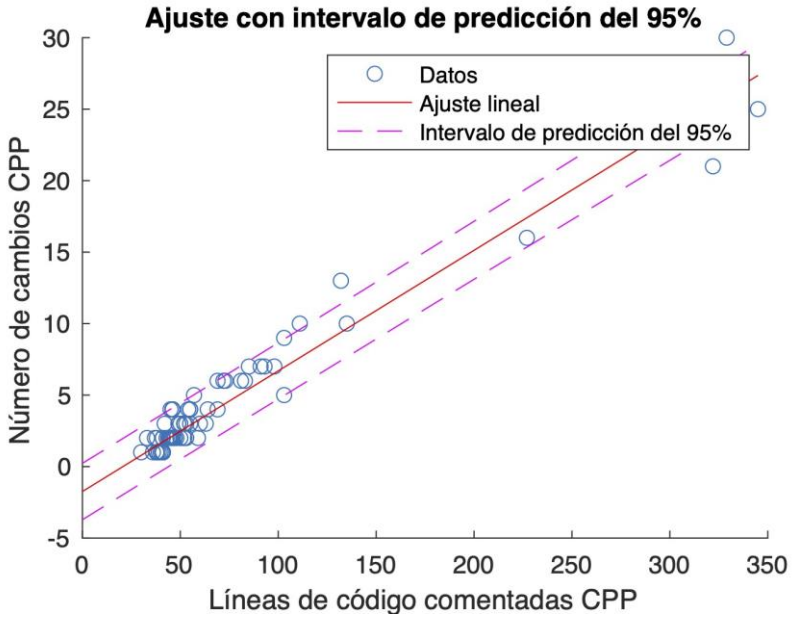

Figura 2: Correlación de los cambios y líneas comentadas

- Métrica No. 4 - TDE Object ID Metric: Es obligatorio que el object ID de un TDE de forma customizada este entre los valores 30,000-31,999 incluídos. Esto garantizará la convivencia de las clases customizadas y las del producto en su versión original.
- Métrica No. 5 – File Last Letter Metric: El nombre de los documentos debe terminar con la letra q y luego seguir la extensión .h o .cpp. Esto se considera un buena práctica para identificación temprana de las clases customizadas en comparación con las del producto en su versión original.

La auditoría aplicada al total de la población mostró que los archivos contienen al menos una de las cinco métricas propuestas en un 89%.
Para el caso de la métrica uno apareció en un 5%, para dos 27%, tres 24%, cuatro 13% y para cinco un 31%. Mientras el 54% presentó máximo una métrica y del restante 46% máximo dos métricas un 85%. Las frecuencias de aparición se muestran en la Figura 3
La distribución de probabilidad considera cinco escenarios. El primero sin deuda técnica, y del segundo hasta el cuarto con 1, 2, 3 y 4 deudas técnicas respectivamente. En la Figura 4 se puede apreciar, una mayor probabilidad de encontrar una, dos y tres deudas técnicas en la auditoría de un programa al azar. En el antepenúltimo escenario se lograría un programa sin deudas técnicas y en último lugar sería menos probable tener más de cuatro deudas técnicas. En tanto, la esperanza para una auditoría realizada al azar tendría un valor de 1.52 de deudas técnicas.
En la Tabla 1 se muestran las clasificaciones de las métricas encontradas en los programas. No se encontró evidencia de alguna deuda relacionada con la seguridad. Dos de las métricas se relacionan con las del tipo smell code. Las del tipo Bug fueron tres métricas. Aunque, los porcentaje de los tipos de deudas fueron equilibradas, con un 42% para Bug y el restante para smell code. Adicionalmente, no se encontraron más de dos tipos de deuda técnica en una métrica.

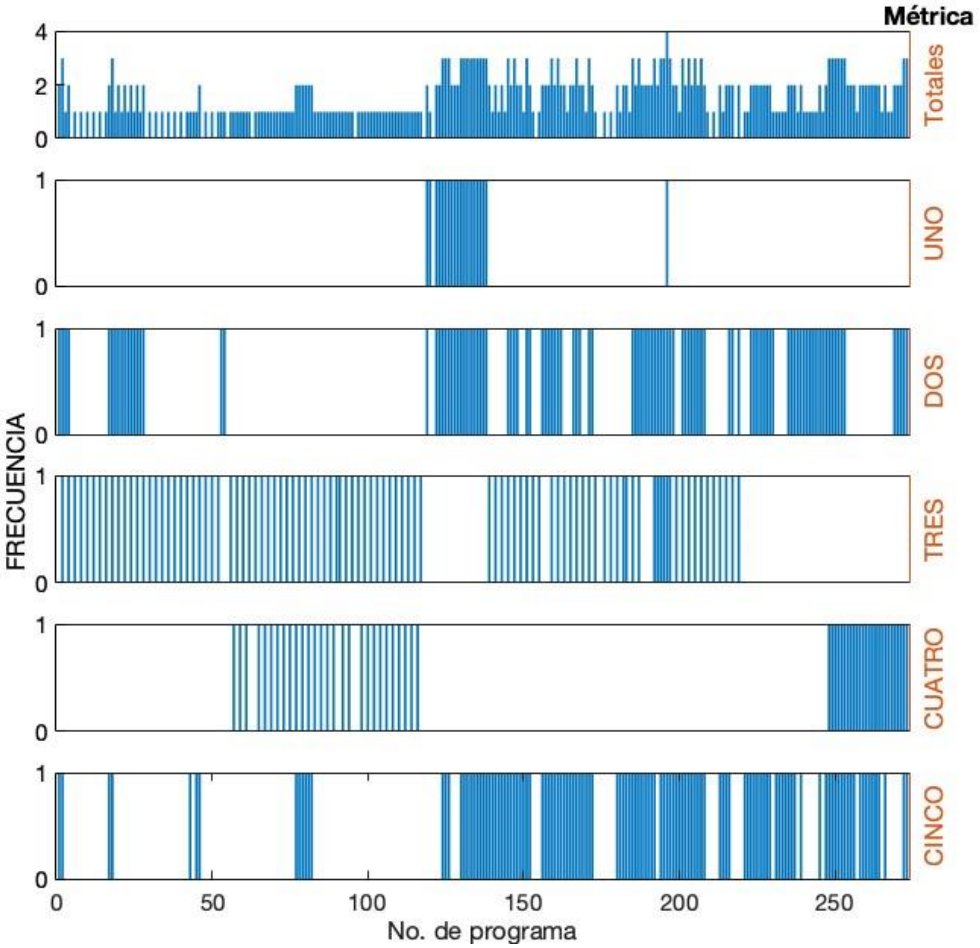

Figura 3: Auditoría para todos los archivos

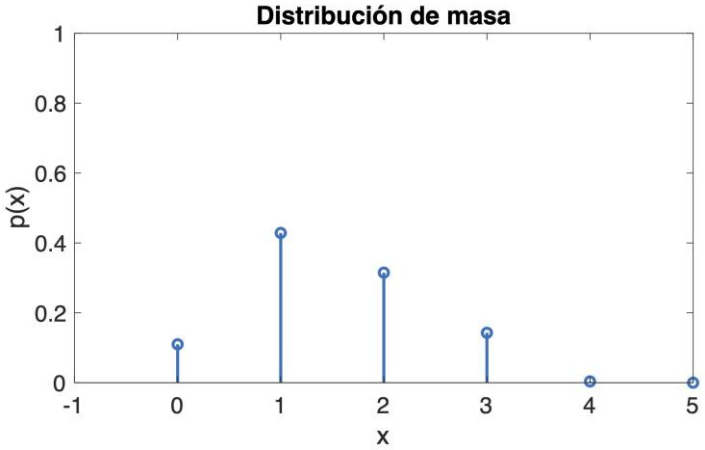

Figura 4: Distribución de probabilidad de las deudas técnicas Tabla 1: Tipo de deuda técnica para cada métrica

Métrica	Bug	Smell code	Seguridad
Token Registry Metric	Si	No	No
Class Prefix Metric	No	Si	No
SSID Event Metric	Si	No	No
TDE Object ID Metric	Si	No	No
File Last Letter Metric	No	Si	No

4. Conclusiones

El desarrollo de la investigación permitió cumplir el objetivo de identificar algunas métricas que necesitan ser incorporadas a los modelos de calidad para evitar dos tipos de deudas técnicas conocidas como bugs y code smells. No se encontró alguna deuda técnica del tipo de seguridad en los programas analizados. La implicación de las métricas tipo bugs, como la No. 1 Token Registry Metric, la No. 3 SSID Event Metric y la No. 4 TDE Object ID Metric, evitarían interrupciones en la ejecución de los programas y la ausencia temporal o permanente de la disponibilidad del producto de software. De la misma manera las métricas tipo code smells, como la No. 2 Class Prefix Metric y la No. 5 File Las Letter Metric, simplificarían en el futuro los mantenimientos a los programas. El porcentaje de correlación con el número de cambios en contraste con las líneas comentadas tuvieron un 96%. Es decir, si se revisan e integran los comentarios en alguna etapa del proceso de compilación o producción del programa se podría minimizar las deudas técnicas.

El método fue válido para los programas de dos diferentes empresas de servicios financieros transaccionales que utilizan la herramienta B24-eps. Los pasos sugeridos en el método pueden ser replicados por cualquier empresa del tipo financiero independientemente de la herramienta utilizada. Con lo anterior, cada empresa podría sugerir alguna actualización de la norma o ratificar las métricas propuestas en esta investigación.

Al menos dos vertientes de trabajos futuros se han encontrado. La primera vinculada a la relación que pueda existir entre la deuda técnica y el tipo de programador. La segunda idea se puede centrar en la creación de un complemento automatizado en la detección de deudas técnica en una plataforma de análisis de código estático como Sonarqube.

5 Referencias

ACI-Worldwide. (2023). *Manage multi-channel transactions with base24-eps.* Descargado de https://recibe.page.link/ACI23

Alfayez, R., Winn, R., Alwehaibi, W., Venson, E., y Boehm, B.(2023). How sonarqube-identified technical debt is prioritized: An exploratory case study. *Information and Software Technology*, *156*, 107147. doi: https://doi.org/10.1016/ j.infsof.2023.107147

Alves, N. S., Mendes, T. S., Mendonça, M. G. D., Spinola, R. O., Shull, F., y Seaman, C.
(2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, *70*, 100-121. doi: https://doi.org/10.1016/ j.infsof.2015.10.008

Atlassian. (2023).*Escaping the black hole of technical debt | atlassian.* Descargado de https://recibe.page.link/Atlassian23

Gartner, I. (2023). *Definition of technical debt - gartner information technology glossary.* Descargado de https://recibe.page.link/Gartner23

ISO. (2014). *Iso 25000 square series.* Descargado de https://recibe.page.link/ISO14

Katin, A., Lenarduzzi, V., Taibi, D., y Mandić, V. (2022). On the technical debt prioritization and cost estimation with sonarqube tool. , 302-309. doi: https://doi.org/10.1007/978-3 -030-97947-8_40

Kruchten, P., Nord, R. L., y Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, *29*, 18-21. doi: https://doi.org/10.1109/MS.2012.167

Li, Y., Soliman, M., y Avgeriou, P. (2022). Automatic identification of self-admitted technical debt from four different sources. *Empirical Software Engineering 2023 28:3*, *28*, 1-38. doi: https://doi.org/10.1007/s10664-023-10297-9

Mamun, M. A. A., Martini, A., Staron, M., Berger, C., y Hansson, J. (2019). Evolution of technical debt : An exploratory study. , *2476*, 87-102. Descargado de https://recibe .page.link/Mamun19

Pavlič, L., Hliš, T., Heričko, M., y Beranič, T. (2022). The gap between the admitted and the measured technical debt: An empirical study. *Applied Sciences 2022, Vol. 12, Page*
7482, *12*, 7482. doi: https://doi.org/10.3390/app12157482

Saarimaki, N., Baldassarre, M. T., Lenarduzzi, V., y Romano, S. (2019). On the accuracy of sonarqube technical debt remediation time. *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 317-324. doi: https://doi.org/
10.1109/SEAA.2019.00055

SONAR. (2023). *Technical debt | definition guide | sonar.* Descargado de https://recibe .page.link/SONAR23

Stopford, B., Wallace, K., y Allspaw, J. (2017). Technical debt: Challenges and perspectives. *IEEE Software*, *34*, 79-81. doi: https://doi.org/10.1109/MS.2017.99