Fecha de presentación: febrero, 2023
Fecha de aceptación: abril, 2023
Fecha de publicación: junio, 2023

44

ANÁLISIS DE LA DEUDA TÉCNICA

EN DESARROLLADORES NOVELES DE SOFTWARE: ENFOQUE DESDE LA PERSPECTIVA ACADÉMICA

ANALYSIS OF TECHNICAL DEBT IN NOVICE SOFTWARE DEVELOPERS: AN ACADEMIC PERSPECTIVE

Edwin Fabricio Lozada Torres[1]
E-mail: ua.edwinlozada@uniandes.edu.ec
ORCID: https://orcid.org/0009-0008-3640-445X
María Angélica Pico Pico[1]
E-mail: ua.mariapico@uniandes.edu.ec
ORCID: https://orcid.org/0000-0002-1807-9613
Jorge Rene Viteri Moya[1]
E-mail: ua.jorgeviteri@uniandes.edu.ec
ORCID: https://orcid.org/0000-0002-5705-4590
[1]Universidad Regional Autónoma de Los Andes Ambato. Ecuador.

Cita sugerida (APA, séptima edición)

Lozada Torres, E. F., Pico Pico, M. A., & Viteri Moya J. R. (2023). Análisis de la deuda técnica en desarrolladores noveles de software: Enfoque desde la perspectiva académica. *Universidad y Sociedad*, 15(S2), 389-397.

RESUMEN

El término “deuda técnica” se utiliza para describir la mala calidad del software que aparece cuando el desarrollador es consciente de ella, pero no toma ninguna medida para controlarla. Debido a los principios que la sustentan, se encuentra estrechamente relacionada con los modelos de desarrollo ágil. Por ello, el presente estudio se enfoca en analizar la percepción de la deuda técnica en programadores noveles y desarrollar estrategias para mejorar los resultados en el desarrollo de software, mediante el diseño de campo y modelación AHP de Saaty. Los resultados revelaron que los programadores noveles subestiman el valor de la deuda técnica y carecen de conciencia sobre su impacto en la calidad del software. Se proponen estrategias como la enseñanza de buenas prácticas desde los primeros cursos, al enfocarse en modularidad, diseño limpio y uso adecuado de patrones. Del mismo modo se destaca la necesidad de promover la conciencia sobre la calidad del código y adoptar estándares de codificación. En conclusión, se resalta la necesidad de abordar la deuda técnica en programación de software y propone estrategias efectivas para mejorar los resultados en programadores noveles. Además, se fortalece la calidad del software y la formación de profesionales conscientes y competentes.

Palabras clave: Visión, deuda técnica; desarrolladores noveles; perspectiva académica; software.

ABSTRACT

The term “technical debt” is used to describe the poor quality of software that arises when the developer is aware of it but takes no action to control it. Due to the principles it is based on, it is closely related to agile development models. Therefore, this study focuses on analyzing the perception of technical debt in novice programmers and developing strategies to improve software development outcomes through field design, and Saaty’s AHP modeling. The results revealed that novice programmers underestimate the value of technical debt and lack awareness of its impact on software quality. Strategies are proposed, such as teaching good practices from the early courses, emphasizing modularity, clean design, and proper use of patterns. The importance of promoting awareness of code quality and adopting coding standards is also highlighted. In conclusion, the need to address technical debt in software programming is emphasized, and effective strategies are proposed to improve outcomes in novice programmers, thereby enhancing software quality and the training of conscientious and competent professionals.

Keywords: Vision, technical debt; junior developers; academic perspective; software.

INTRODUCCIÓN.

Los modelos de desarrollo ágiles se distinguen por la entrega frecuente de funcionalidades en plazos definidos, lo que a veces resulta en la omisión de ciertas características o la falta de una supervisión adecuada en cuanto a la calidad del producto. Cuando esto ocurre, se genera una deuda técnica que puede manifestarse en forma de deficiencias o incumplimientos en el producto final.

La metodología ágil de desarrollo se ha vuelto muy popular entre los desarrolladores en la actualidad y también se utiliza ampliamente en el entorno académico. Sin embargo, cumplir con los tiempos en las entregas obliga a trabajar con una alta improvisación y ausencia de control, al afectar la calidad del software (Flores-Cerna et al., 2021).

De igual manera, los profesionales en desarrollo de software que utilizan los modelos ágiles son los más propensos a experimentar los efectos de la deuda técnica. Estos efectos incluyen problemas relacionados con la baja calidad del código interno y la limitada capacidad de mantenimiento, donde se encuentran estrechamente vinculados con las actividades de programación. (Rios et al., 2021).

Se debe agregar que, según lo señalado en el estudio realizado, a pesar de ser un problema persistente entre los desarrolladores de software, la deuda técnica constituye un área de oportunidad para mejorar la calidad del producto y su mantenibilidad en el futuro. (Silva et al., 2019) Poe ello, se debe destacar que los profesionales de software no la perciben adecuadamente, de modo que no se gestiona convenientemente.

De manera similar, si las actividades planificadas para un proyecto de desarrollo de software no se realizan adecuadamente o se quedan incompletas, se genera la deuda técnica. Este factor puede tener un impacto significativo en la calidad del producto y, en particular, en la salud financiera del proyecto, (Melo et al., 2022). Además de disminuir el valor del producto, es significativo considerar los costos asociados al mantenimiento del software y el impacto negativo en el cliente debido a retrasos, problemas de calidad y la necesidad de mantener el sistema operativo. (Avgeriou et al., 2018).

En referencia, según (Melo et al., 2022), existen diversas causas que pueden originar la deuda técnica, por ende, es necesario su identificación y medición temprana. Esta deuda puede surgir tanto de manera intencional como no intencional, al afectar tanto al equipo de desarrollo como a los stakeholders involucrados en el proyecto. Es transcendental destacar que la deuda técnica puede surgir por malas prácticas de ingeniería en el desarrollo de software o por decisiones estratégicas intencionales. Por lo tanto, es necesario estar atentos a su detección y gestión para minimizar su impacto en la calidad del producto y en el ámbito financiero.

Sobre este aspecto referente al ámbito de la programación, se visualiza que los programadores noveles puedan identificar y manejar de manera adecuada la deuda técnica. Dado que las habilidades requeridas para desarrollar software incluyen la creación de soluciones eficientes, donde es esencial que los programadores novatos reconozcan la importancia del diseño y de la calidad del producto final. Según una investigación realizada por (Fuentes-Rosado & Moo-Medina, 2017), los programadores noveles pueden enfrentarse a una amplia gama de dificultades y deficiencias. Estas incluyen fobia a los problemas complejos, lógica incompleta, desconocimiento del lenguaje y de las herramientas del entorno de desarrollo (IDE), falta de motivación y dificultades en la gestión del tiempo.

En cambio, hay que insistir que los programadores noveles reciban una adecuada formación en el manejo de la deuda técnica, sobre la base de desarrollar productos de calidad y evitar problemas en el futuro. En este sentido, los autores (Fuentes-Rosado & Moo-Medina, 2017) hacen hincapié en la importancia de que los estudiantes comprendan los pasos necesarios para crear productos de calidad y puedan aplicarlos de manera efectiva en su práctica diaria.

En ese mismo sentido, es necesario que en el entorno académico se fortalezcan las competencias mínimas mediante un modelo de enseñanza que contribuya en la adquisición de capacidades para el desarrollo de aplicaciones informáticas y así contribuir con profesionales cualificados en el área (Romero et al., 2019). En efecto, se deben usar estrategias y técnicas para favorecer el aprendizaje y que contribuyan para que el programador novel puede solucionar los problemas tradicionales relacionados con el desarrollo de software (Pellicer et al., 2020). En el marco de las observaciones anteriores, se plantea en analizar la percepción de la deuda técnica en programadores noveles y desarrollar estrategias para mejorar los resultados en el desarrollo de software de los estudiantes de las carreras de Software de la ciudad de Ambato, mediante el diseño de campo y modelación del método AHP de Saaty.

MATERIALES Y MÉTODOS

En este estudio, se recopiló información existente referentes a resultados de encuestas y entrevistas realizadas y de los resultados obtenidos en proyectos de software en funcionamiento presentado por los estudiantes

universitarios de último nivel de la carrera de Software en varias instituciones educativas de la ciudad de Ambato. De los resultados y la información recopilada se visualizan en el estudio las de mayor impacto relacionados con la percepción de la deuda técnica en el desarrollo de proyectos de software académicos. De igual manera la obtención de la información relevante sobre el conocimiento y las opiniones de los estudiantes ayuda a identificar áreas de mejora en su formación como futuros desarrolladores de software.

Proceso Jerárquico Analítico Saaty técnica del análisis multicriterio (AHP de Saaty)

Para la modelación y selección de la mejor estrategia se utilizó el método Proceso Analítico Jerárquico (AHP Saaty) propuesto por Thomas Saaty 1980. Es uno de los métodos más extendidos para resolver problemas de toma de decisiones de múltiples estrategias. Esta técnica modela el problema que conduce a la formación de una jerarquía representativa del esquema de toma de decisiones asociado. Esta jerarquía presenta en el nivel superior el objetivo que se persigue en la solución del problema y en el nivel inferior se incluyen las distintas alternativas a partir de las cuales se debe tomar una decisión. Los niveles intermedios detallan el conjunto de criterios y atributos considerados.

A fines de los años setenta el profesor Saaty, doctor de matemáticas de la Universidad de Yale, creó un modelo matemático denominado proceso jerárquico analítico Proceso Jerárquico Analítico (Analytical Hierarchy Process, AHP) como una forma efectiva de definir medidas para tales elementos y usarlas en los procesos de toma de decisiones. El AHP es una teoría orientada hacia el responsable de la toma de decisiones y sirve para identificar la mejor alternativa de acuerdo con los recursos asignados. Este método puede aplicarse a situaciones que involucran factores de tipo técnico, económico, político, social y cultural. Es decir, pretende ser una herramienta científica para abordar aquellos aspectos que son difícilmente cuantificables, pero que a veces requieren una unidad de medida. Algunos autores plantean que el AHP no ha sido bien comprendido, ya que va más allá de ser una simple metodología para situaciones de elección. Se plantea entonces, que la mejor manera de entender el método es describir tres funciones básicas: estructurar la complejidad, medir en una escala y sintetizar. A continuación, se describen éstas de una manera breve:

- Estructuración de la Complejidad. Saaty buscó una manera para resolver el problema de la complejidad, y utilizó la estructuración jerárquica de los problemas en sub-problemas homogéneos.
- Medición en escalas. El AHP permite realizar mediciones de factores tanto subjetivos como objetivos a partir de estimaciones numéricas, verbales o gráficas, lo cual le provee una gran flexibilidad y gran variedad de aplicaciones en campos tan distintos unos de otros.
- Síntesis. Aunque el nombre incluya la palabra Análisis, el enfoque del AHP es totalmente sistémico. aunque analiza las decisiones a partir de la descomposición jerárquica, en ningún momento pierde de vista el objetivo general y las interdependencias existentes entre los conjuntos de factores, criterios y alternativas, por lo tanto, este método está enfocado en el sistema en general, y la solución que presenta es para la totalidad, no para la particularidad.

El proceso se basa en varias etapas. La formulación del problema de la toma de decisiones en una estructura jerárquica es la primera y principal etapa. Esta etapa es donde el tomador de decisiones debe desglosar el problema en sus componentes relevantes. La jerarquía básica está compuesta por: metas u objetivos generales, criterios y alternativas. La jerarquía está construida de manera que los elementos sean del mismo orden de magnitud y puedan relacionarse con algunos del siguiente nivel (figura 1 y 2) (ver tabla 1).

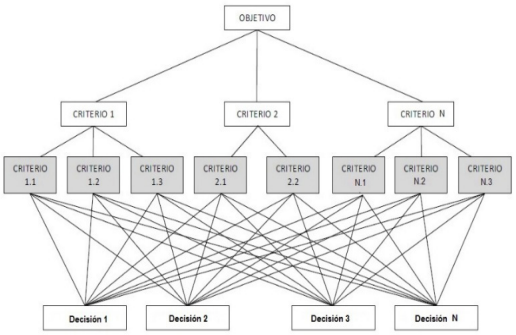

Figura 1. Esquema de un árbol genérico que representa un proceso de jerarquía analítica. Fuente: AHP Saaty, 2008.

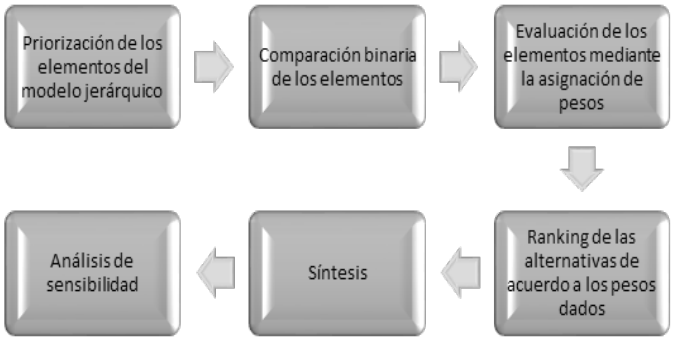

Figura 2. Metodología AHP de Saaty. Fuente: Elaboración propia.

Tabla 1. Escala de evaluación de Saaty (Tasa juicio verbal)

Escala	
9 Extremadamente más preferido	3 Moderadamente más preferido
7 Muy poderosamente más preferido	1 Igualmente preferido
5 Poderosamente más preferido	

Fuente: Saaty, 2008

A continuación, se presenta un algoritmo para el cálculo de éste (este debe aplicarse para todos los criterios:

- Para cada línea de la matriz de comparación por pares determinar una suma ponderada con base a la suma del producto de cada celda por la prioridad de cada alternativa o criterio correspondiente.
- Para cada línea, dividir su suma ponderada por la prioridad de su alternativa o criterio correspondiente.
- Determinar la media λ_{max} del resultado de la etapa anterior.
- Calcular el índice de consistencia (CI) para cada alternativa o criterio (ver tabla 2).

$$CI = \frac{\lambda_{max} - m}{m - 1} \qquad (1)$$

- Donde m es el número de alternativas
- Determinar el Índice Aleatorio (IA) de la tabla 3
- Determinar el índice de cociente de consistencia (la razón entre el índice de consistencia y el índice aleatorio)

Tabla 2. Índice aleatorio para el cálculo del coeficiente de consistencia

Número de alternativas para la decisión n	Índice aleatorio	Número de alternativas para la decisión n	Índice aleatorio
3	0.58	7	1.32
4	0.9	8	1.41
5	1,12	10	1,49
6	1,24		

Fuente: Elaboración propia

Para el completar la modelación del método se obtuvo información general de los participantes y se indagó sobre sus estrategias de resolución de problemas. Además, se evaluó el conocimiento y las prácticas de los estudiantes en relación con la deuda técnica. Estos datos permiten mejorar la calidad de los proyectos y la formación en gestión de la deuda técnica en el ámbito académico.

RESULTADOS Y DISCUSIÓN

De la información obtenida se evaluó la participación de los estudiantes de ingeniería en el estudio, al considerar el contexto académico y la alta tasa de deserción en los niveles iniciales. La cooperación de los docentes fue fundamental para promover la participación de los estudiantes, al resaltar la necesidad de la comunicación y colaboración entre ellos. Estos estudios son clave para evaluar el desempeño académico y mejorar la calidad educativa, al fomentar una cultura de evaluación continua. Se obtuvo información de dos importantes universidades ecuatorianas, UNIANDES y la Universidad Técnica de Ambato.

Desarrollo de Software

Se evaluó el modelo de desarrollo de software más utilizado por los estudiantes y se encontró que la mayoría utiliza metodologías ágiles, mientras que un porcentaje reducido emplea modelos tradicionales e híbridos. Este enfoque refleja la tendencia actual en la industria, ya que las metodologías ágiles se valoran por su capacidad de adaptación y mejora de la calidad del producto. Además, se examinó la generación de la documentación, donde se visualiza que la mayoría de los estudiantes la elabora siempre o con frecuencia, para la comprensión del software y facilitar el mantenimiento. Por último, se exploró la autoevaluación de los programadores noveles en su rol como desarrolladores, al revelar que la mayoría se considera competente (para un 61%), seguido de principiantes (para un 32%) y expertos en menor proporción (para un 7%) como se puede observar en la Figura 3.

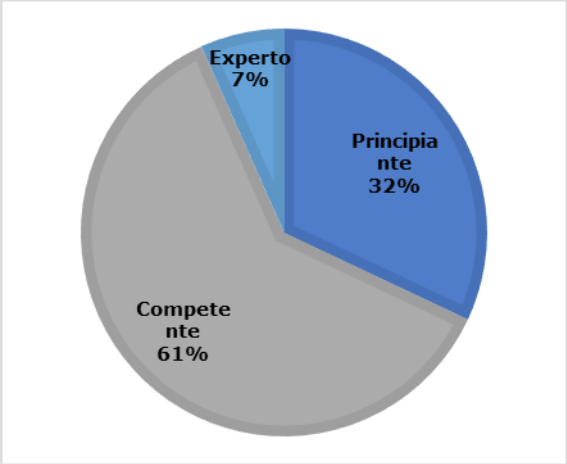

Figura 3. Rol de programador. Fuente: Encuesta sobre percepción de la deuda técnica.

Se exploraron las percepciones de los estudiantes universitarios sobre los problemas al entregar un producto con errores o defectos en el desarrollo de software. Aunque todos los programadores noveles reconocieron la existencia de problemas en este aspecto, no hubo un consenso del 100% en las respuestas. Esto indica una variabilidad en la percepción de los estudiantes, lo que sugiere la necesidad de un análisis más detallado para identificar los problemas más relevantes en este contexto.

De la información aportada se revelaron varios problemas, al destacarse el "código mal escrito" como la respuesta más común, seguida de "defectos en el funcionamiento del software". Estos problemas pueden tener un impacto significativo en la calidad del software entregado y deben abordarse durante el proceso de desarrollo. Por ende, se recomienda realizar un análisis más exhaustivo de la información y considerar las opiniones de los estudiantes para identificar los problemas más relevantes en esta área (ver tabla 3). Este análisis contribuye al entendimiento científico de los desafíos en la entrega de productos con errores o defectos en el desarrollo de software y proporciona información valiosa para mejorar la calidad de los productos entregados en el ámbito académico.

Tabla 3. Principales problemas que encuentra al entregar un producto con errores o defectos.

Problema	N. Estudiantes
Código mal escrito	45
Defectos en el funcionamiento del software	34
Funciones requeridas, pero no implementadas	27
Presencia de defectos conocidos que no se corrigieron	21
Problemas de calidad de código triviales	17
Tareas planificadas, no realizadas o inconcusas	15
Aspectos de baja calidad	13

Fuente: Encuesta sobre percepción de la deuda técnica.

De los resultados se analizó el tiempo dedicado por los estudiantes universitarios a la corrección de errores y mantenimiento en el desarrollo de software. Los resultados revelaron que más de la mitad de los programadores (51%) destinan entre el 25% y el 50% del tiempo total empleado en desarrollar el software a la corrección de errores. Además, el 21% de los programadores utilizan entre el 50% y el 75% de su tiempo en esta actividad, lo que indica que algunos estudiantes dedican una cantidad considerable de tiempo a la corrección de errores y mantenimiento. Por otro lado, el 19% de los noveles emplean entre el 0% y el 25% de su tiempo total en la corrección de errores, lo que podría indicar una falta de consideración o dedicación a esta tarea.

Además, el 9% de los programadores destinan entre el 75% y el 100% de su tiempo total a la corrección de errores, lo que sugiere que podrían enfrentar problemas significativos en la gestión del proceso de desarrollo de software. Estos hallazgos resaltan la prioridad de la corrección de errores y el mantenimiento en el desarrollo de software. Por tanto, sugieren la necesidad de mejorar la calidad del software para reducir la cantidad de errores en el producto final, lo que a su vez puede disminuir el tiempo y los costos asociados al mantenimiento. Esta observación proporciona información relevante para la optimización de los procesos de desarrollo de software y la mejora de la eficiencia en el ámbito académico (ver figura 4).

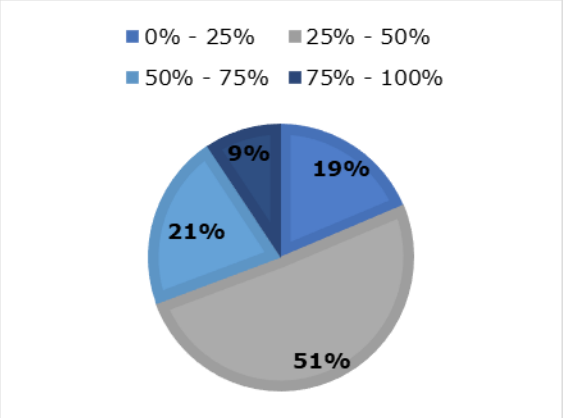

Figura 4. Tiempo empleado en la corrección de errores. Fuente: encuesta sobre percepción de la deuda técnica.

Deuda Técnica

Los resultados obtenidos revelan un nivel de conocimiento y conciencia significativo por parte de los programadores noveles sobre la prioridad de corregir errores en el proceso de desarrollo de software (ver tabla 4). El hecho de que un alto porcentaje de estudiantes esté consciente de la necesidad de corregir defectos y errores en el software funcional entregado demuestra una comprensión adecuada de la importancia de la calidad del software y su impacto en la satisfacción del cliente y la reputación de la empresa desarrolladora. Sin embargo, es importante señalar que aún existe un porcentaje significativo de estudiantes que no está consciente de esta situación. Ello sugiere la necesidad de una mayor educación y concienciación acerca de la necesidad de la corrección de errores en el proceso de desarrollo de software. Por ende, es fundamental que los estudiantes comprendan que corregir errores es una parte esencial del proceso de desarrollo de software y que su impacto en la calidad del producto es fundamental para el éxito del proyecto y la

satisfacción del cliente. Este hallazgo destaca la necesidad de fortalecer la formación en este aspecto y promover una cultura de calidad en el desarrollo de software.

Tabla 4. Responsabilidad de los programadores en la corrección de defectos y/o errores software.

Problema	N. Estudiantes
Está consciente	36
Lo percibe	34
No está consciente	5

Fuente: encuesta sobre percepción de la deuda técnica.

La respuesta de los programadores noveles en relación con la necesidad de abordar los problemas pendientes en un software funcional demuestra una actitud responsable y comprometida con la calidad del producto entregado. Es significativo destacar que la presencia de deuda técnica que se entregan junto con el software puede tener consecuencias negativas en la satisfacción del usuario final, la reputación de la empresa y el rendimiento del sistema.

La respuesta mayoritaria que destaca la necesidad de implementar controles para minimizar o eliminar la deuda técnica sugiere que los programadores noveles son conscientes de abordar este problema de manera proactiva y evitar que se acumule durante el ciclo de vida del software. Esto podría lograrse mediante el uso de técnicas y herramientas de gestión de calidad de software, como pruebas automatizadas, análisis estático de código y revisiones de código. Por otro lado, la mayoría de los estudiantes (52%) no está familiarizada con el término “deuda técnica” en el desarrollo de software, aunque un porcentaje considerable (24%) indica haber escuchado o leído algo al respecto.

AHP de Saaty

De los resultados obtenidos de la investigación exploratoria son fundamentales para comprender la realidad académica y tomar decisiones para mejorar la educación superior. Esto sugiere la necesidad de difundir y educar más sobre este concepto en el ámbito académico y promover una comprensión más amplia de sus implicaciones y estrategias para su manejo. Para ello, se presentan 6 estrategias para gestionar y reducir la deuda técnica en el ámbito académico, con el fin de mejorar los resultados en el desarrollo de software:

E1	Enseñanza de buenas prácticas de desarrollo de software: Introducir y enfatizar desde los primeros cursos conceptos clave como el modularidad, el diseño limpio, la legibilidad del código y el uso adecuado de patrones de diseño. Promover la importancia de la calidad del código y fomentar la adopción de estándares de codificación.
E2	Implementación de revisiones de código: Establecer un proceso sistemático de revisión de código en el entorno académico, donde los estudiantes presenten y reciban retroalimentación constructiva sobre sus proyectos. Esto permitirá identificar y corregir posibles problemas y deficiencias en el código, al reducir así la acumulación de deuda técnica.
E3	Fomento de la documentación adecuada: Educar a los estudiantes sobre la importancia de una documentación clara y completa. Enfatizar la documentación de los requisitos, el diseño, las decisiones de implementación y las pruebas realizadas. Proporcionar pautas y plantillas para facilitar la creación de documentación de calidad.
E4	Incorporación de pruebas automatizadas: Introducir la práctica de realizar pruebas automatizadas en el desarrollo de software desde las etapas iniciales. Enseñar a los estudiantes a escribir pruebas unitarias y de integración, y a realizar pruebas continuas durante todo el ciclo de vida del desarrollo. Esto ayuda a detectar y corregir errores de manera temprana, al evitar la acumulación de deuda técnica.
E5	Uso de herramientas de análisis estático de código: Promover el uso de herramientas de análisis estático de código que puedan identificar problemas comunes, como código duplicado, complejidad excesiva o violaciones de buenas prácticas. Estas herramientas pueden ayudar a los estudiantes a mejorar la calidad de su código y a mantenerlo libre de deuda técnica.
E6	Establecer la colaboración entre estudiantes y docentes: Fomentar un ambiente de colaboración donde los estudiantes puedan buscar orientación y asesoramiento de los docentes en cuanto a buenas prácticas de desarrollo y gestión de la deuda técnica. Proporcionar espacios regulares de consulta y revisión

Estas estrategias, combinadas y adaptadas según las necesidades y características del entorno académico, pueden contribuir significativamente a la gestión y reducción de la deuda técnica. Al promover una mentalidad de calidad y

buenas prácticas desde el inicio de la formación académica, se sientan las bases para futuros profesionales del desarrollo de software conscientes y comprometidos con la entrega de productos de software de alta calidad.

De igual modo se determina las estrategias con mayor potencial mediante el uso de la modelación del método AHP Saaty. A partir de la modelación se obtiene la tabla resultante con los pesos luego de haber efectuado la matriz de comparación binaria del AHP Saaty (Tabla 5 y 6).

Tabla 5. Matriz normalizada.

Criterios	F1	F2	F3	F4	F5	F6	PESO
F1	0.36	0.51	0.27	0.35	0.24	0.38	0.35
F2	0.12	0.17	0.20	0.24	0.24	0.25	0.20
F3	0.09	0.06	0.07	0.06	0.04	0.06	0.06
F4	0.12	0.09	0.13	0.12	0.12	0.13	0.12
F5	0.18	0.09	0.20	0.12	0.12	0.06	0.13
F6	0.12	0.09	0.13	0.12	0.24	0.13	0.14

Fuente: Elaboración propia.

Tabla 6. Análisis de la consistencia del ejercicio.

Criterios	Valores propios aproximados	
F5	6.335201524	**Valor propio= 6.2200143 IC=0.04 RC=0.04<0.10 Consistente**
F2	6.270634575	
F3	6.193329406	
F4	6.200994874	
F1	6.090299875	
F6	6.229625263	

Fuente: Elaboración propia.

Por lo tanto, basado en el análisis del método AHP de Saaty, se concluye que las estrategias más efectivas para gestionar y reducir la deuda técnica en el ámbito académico, con el fin de mejorar los resultados en el desarrollo de software, constituyen la enseñanza de buenas prácticas de desarrollo de software y en segundo lugar la implementación de revisiones de código. Estas estrategias deberían recibir una mayor atención y implementarse de manera prioritaria en el currículo y los métodos de enseñanza de la carrera de Software.

Por ello, se propone las siguientes acciones para implementar la estrategia de enseñanza de buenas prácticas de desarrollo de software en el ámbito académico:

- Diseño curricular: Revisar y actualizar el plan de estudios de la carrera de Software para incluir de manera explícita y secuencial la enseñanza de las buenas prácticas de desarrollo de software. Establecer cursos específicos o módulos dedicados a este tema desde los primeros semestres, con una progresión gradual de los conceptos y técnicas.
- Material didáctico: Desarrollar y proporcionar material didáctico actualizado que aborde los conceptos clave de las buenas prácticas de desarrollo de software. Esto puede incluir libros de texto, tutoriales, guías de estilo de codificación y recursos en línea. El material debe ser accesible y comprensible para los estudiantes, para facilitar su aprendizaje y aplicación práctica.
- Metodología de enseñanza: Emplear enfoques pedagógicos activos y prácticos que fomenten la participación y el aprendizaje interactivo de los estudiantes. Utilizar ejemplos reales, estudios de caso y proyectos prácticos que permitan a los estudiantes aplicar las buenas prácticas en situaciones concretas. Fomentar la colaboración entre estudiantes para el intercambio de conocimientos y experiencias.
- Evaluación: Incorporar la evaluación de las buenas prácticas de desarrollo de software en los criterios de evaluación de los cursos correspondientes. Utilizar diferentes métodos de evaluación, como proyectos individuales o en

equipo, exámenes teóricos y prácticos, revisiones de código y análisis de la calidad del software producido por los estudiantes. Proporcionar retroalimentación constructiva y orientación para mejorar continuamente sus habilidades y conocimientos.

- Recursos y ejemplos prácticos: Facilitar a los estudiantes el acceso a herramientas de desarrollo de software y entornos de programación que promuevan las buenas prácticas. Proporcionar ejemplos prácticos y soluciones de referencia que ilustren la aplicación de las buenas prácticas en diferentes contextos y proyectos. Promover la participación en comunidades y grupos de desarrollo de software donde los estudiantes puedan compartir sus experiencias y recibir retroalimentación de profesionales de la industria.
- Sensibilización y formación docente: Capacitar a los docentes en las buenas prácticas de desarrollo de software y en técnicas efectivas de enseñanza relacionadas. Organizar talleres, seminarios y actividades de actualización profesional para mantenerse al tanto de los avances en el campo y mejorar sus habilidades pedagógicas. Fomentar la colaboración entre docentes y profesionales de la industria para enriquecer la enseñanza con casos prácticos y ejemplos relevantes.

Los desarrolladores de software novatos tienden a utilizar predominantemente modelos ágiles en las fases del ciclo de vida del desarrollo, pero también consideran los modelos tradicionales e híbridos. Algunos estudios sugieren una integración de los modelos iterativos y tradicionales en proyectos grandes o con procesos de desarrollo a largo plazo (Gharajeh, 2019). Aunque los modelos ágiles no enfatizan la documentación extensa (Yu, 2018), la mayoría de los desarrolladores novatos aún generan documentación. Sin embargo, se recomienda utilizar solo la documentación necesaria según los defensores de las metodologías ágiles.

En cuanto al rol del estudiante como programador, la mayoría se considera competente, pero un número considerable se autodenomina principiante (Habib & Romli, 2021). Los estudiantes deben enseñarse en técnicas de alto nivel para resolver problemas de codificación de manera efectiva y tener éxito en la solución de problemas (Castro & Fisler, 2020).

En relación con los problemas que se deben abordar en el desarrollo de software, existe debate sobre si es mejor retrasar la entrega del producto hasta corregir los errores o entregarlo primero y corregir los defectos posteriormente (Hossain, 2018). La calidad del software debe ser una prioridad, ya que la industria del software sufre pérdidas considerables debido a problemas de calidad. Las causas más comunes de entregar un producto con defectos o errores son el código mal escrito y los defectos en la funcionalidad (Rios et al., 2018).

Los desarrolladores novatos dedican entre un 25% y un 50% del tiempo total de desarrollo a la corrección de errores y mantenimiento del software (Noyori et al., 2019). No existe una regla para predecir el tiempo necesario para corregir errores, y posponer su solución para futuras versiones del software disminuye la probabilidad de resolverlos (Akbarinasaji et al., 2018).

En cuanto a la percepción de la deuda técnica, los desarrolladores la reconocen, pero no toman medidas significativas para controlarla. Se necesita una mejor comprensión y difusión del concepto de deuda técnica para mejorar el conocimiento y comprensión de este fenómeno entre los desarrolladores novatos (Silva et al., 2019; Rios et al., 2020).

Por ende, es necesario implementar estrategias y acciones que contribuya a introducir y enfatizar desde los primeros cursos las buenas prácticas de desarrollo de software. De modo que permite ofrecer a los estudiantes los conocimientos y las habilidades necesarias para producir código de calidad y adoptar estándares de codificación adecuados. Además, contribuye a mejorar los resultados en el desarrollo de software en el ámbito académico, al formar a futuros profesionales con una base sólida en buenas prácticas y promover la excelencia en la industria.

CONCLUSIONES.

Se demuestra en el trabajo que deuda técnica es un desafío significativo en el desarrollo de software, y su comprensión y gestión son fundamentales para garantizar la calidad y el éxito de los proyectos. Los programadores noveles deben ser conscientes de los riesgos asociados con la acumulación de deuda técnica y de la importancia de abordarla de manera proactiva para evitar impactos negativos en la calidad del software y en la productividad del equipo de desarrollo.

Los modelos ágiles de desarrollo de software son ampliamente utilizados y ofrecen beneficios en términos de flexibilidad y adaptabilidad. Sin embargo, es importante destacar que la adopción de enfoques ágiles no debe conducir a la falta de documentación adecuada. La documentación es esencial para garantizar la comprensión y el mantenimiento del software a largo plazo, independientemente de la metodología utilizada. Los programadores noveles deben ser conscientes de la necesidad de equilibrar la agilidad con la documentación adecuada, al buscar un enfoque que se adapte a las necesidades específicas de cada proyecto.

La formación de programadores noveles debe incluir la enseñanza de buenas prácticas de desarrollo de software desde el inicio de su educación. La introducción temprana de conceptos clave, como la modularidad, el diseño limpio y la legibilidad del código, así como el fomento de la adopción de estándares de codificación, puede sentar las bases para el desarrollo de habilidades sólidas y la producción de software de alta calidad. Los programas académicos deben proporcionar recursos didácticos actualizados y promover la colaboración entre estudiantes, la evaluación continua y la retroalimentación constructiva como parte integral del proceso educativo.

REFERENCIAS BIBLIOGRÁFICAS

Akbarinasaji, S., Caglayan, B., & Bener, A. (2018). Predicting bug-fixing time: A replication study using an open source software project. journal of Systems and Software, 136, 173-186.

Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). Dagstuhl Reports 6, 4 (2016), 110–138.

Castro, F. E. V., & Fisler, K. (2020). Qualitative analyses of movements between task-level and code-level thinking of novice programmers. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (pp. 487-493).

Flores-Cerna, F., Sanhueza-Salazar, V. M., Valdés-González, H. M., & Reyes-Bozo, L. (2022). Metodologías ágiles: un análisis de los desafíos organizacionales para su implementación. ***Revista científica***, (43), 38-49.

Fuentes-Rosado, J. I., & Moo-Medina, M. (2017). Dificultades de aprender a programar. ***Revista educacion en ingenieria, 12***(24), 76-82.

Gharajeh, M. S. (2019). Waterative model: An integration of the waterfall and iterative software development paradigms. Database Syst. J, 10, 75-81.

Habib, B., & Romli, R. (2021). A systematic mapping study on issues and importance of documentation in agile. In 2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS) (pp. 198-202). IEEE.

Hossain, M. (2018). Challenges of software quality assurance and testing. International ***Journal of Software Engineering and Computer Systems, 4***(1), 133-144.

Melo, A., Fagundes, R., Lenarduzzi, V., & Santos, W. B. (2022). Identification and measurement of Requirements Technical Debt in software development: A systematic literature review. ***Journal of Systems and Software, 194***, 111483.

Noyori, Y., Washizaki, H., Fukazawa, Y., Ooshima, K., Kanuka, H., Nojiri, S., & Tsuchiya, R. (2019). What are good discussions within bug report comments for shortening bug fixing time? In 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS) (pp. 280-287). IEEE.

Pellicer, Y. S., Zea, M. C., Pérez, R. A., Blanco, Y. C., & Brito, M. G. L. (2020). Visualización Dinámica, Una Opción Para La Enseñanza-Aprendizaje De La Programación De Computadoras. HOLOS, 2, 1-20.

Rios, N., Freire, S., Pérez, B., Castellanos, C., Correal, D., Mendonca, M., ... & Spinola, R. O. (2021). On the relationship between technical debt management and process models. IEEE Software, 38(5), 56-64.

Rios, N., Spínola, R. O., Mendonça, M., & Seaman, C. (2018, October). The most common causes and effects of technical debt: first results from a global family of industrial surveys. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-10).

Rios, N., Spínola, R. O., Mendonça, M., & Seaman, C. (2020). The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. Empirical Software Engineering, 25, 3216-3287.

Romero, H. J. R., Olivera, L. E. C., Mora, C. G. C., & Morales, J. R. P. (2019). Deficiencia de adquisición de competencias mínimas en estudiantes de desarrollo de software: hacia un nuevo modelo de enseñanza pedagógico praxeológico. ***Inclusión y Desarrollo, 6***(2), 85-97.

Silva, V. M., Junior, H. J., & Travassos, G. H. (2019). A taste of the software industry perception of technical debt and its management in Brazil. ***Journal of Software Engineering Research and Development, 7***, 1-1.

Yu, J. (2018). Research process on software development model. In IOP Conference Series: Materials Science and Engineering (Vol. 394, No. 3, p. 032045). IOP Publishing.