MITIGACION DE COSTOS DE DEFECTOS EN EL PROCESO DE DESARROLLO DE SOFTWARE

EDGAR FERNANDO CELIS RUBIANO

UNIVERSIDAD EAN

SEMINARIO DE INVESTIGACION

LUIS ARMANDO COBO CAMPO

JUNIO 06 DEL 2022

Tabla de contenido

Tabla de figuras

Resumen

Este documento es el resultado de la lectura, revisión y análisis de libros, artículos de reconocida calidad académica e investigativa que tratan sobre el proceso de pruebas de calidad del software y los costos que se derivan a partir de esta actividad. Se ha recopilado y seleccionado información para argumentar y sustentar el porqué de la importancia del proceso de pruebas de calidad de software. Se revisó en forma exhaustiva las bases de datos accessengineeringlibrary, elibro-net.bdbiblioteca, proquest.

Palabras clave: calidad, diseño de prueba, niveles de prueba, software, testing, costos software, costos defecto.

Problema de Investigación.

En temas de la gestión de la calidad es común encontrar términos como defecto, falla, falta o error. Cada uno de ellos tiene significados e impactos distintos, tanto en el mismo sistema de software, como en las percepciones de los usuarios. Además, cada uno de ellos se usa en distintos tiempos, según la etapa del proceso de desarrollo de software en donde se haya introducido (inyección del defecto) o su descubrimiento en etapa de operación al ejecutar el sistema (falla).

Tomando como base la discusión que presenta (April, 2018)Una falla es la manifestación de una falta (defecto) en el ambiente de operación. Una falla es la terminación de la capacidad de un componente de realizar la función para la cual fue diseñado. Las fallas se originan porque defectos (faltas) no fueron detectados por las actividades de revisión técnica o de pruebas. Un error se puede encontrar en la documentación, en las instrucciones de código fuente, en la ejecución del código, o en cualquier parte del ciclo de vida del desarrollo de software. Los errores se convierten en defectos cuando los documentos, y productos en general, se aprueban sin que el equipo de desarrollo los haya notado.

Vamos a definir un error como cualquier cosa que haga que nuestro software no cumpla con las expectativas del cliente (ya sea interno o externo). Así podemos tener errores de muchos tipos. Desde los obvios errores de programación, hasta tomar mal los requisitos. (Huerta, 2018)

Una clasificación muy común para el origen de los errores es la siguiente:

- **Requisitos**: No entender lo que pide negocio. O negocio no saber lo que pide. O que los requisitos se documentan mal y por ello el programador hace algo que no es lo correcto. O cualquier otro tipo de error relacionado con requisitos.
- **Diseño**: Errores en el planteamiento del diseño de la aplicación que luego llevan a cambios cuando son detectados.
- **Código**: Errores directamente en la programación. No porque el programador lo entienda mal, sino porque introduce un bug.
- **Otros**: Errores por culpa del control de versiones, o por otros motivos que no están clasificados en los tres anteriores.

James Martin, publicó en 1984 «An Information Systems Manisfesto», en el que calculaba el porcentaje de errores según la clasificación anterior. Los datos que se obtuvieron son demoledores (Martin, 1984):

- Más de la mitad de los errores provienen de requisitos. Y más de la mitad de estos vienen por documentarlos mal y luego entenderse mal.
- Menos del 10% vienen de errores de código.

Por lo tanto, la mayoría de los errores vienen de las primeras fases del proyecto. Así que detectarlos lo antes posible es fundamental.

El costo de los errores depende sobre todo del momento en el que se detectan. Un error detectado en el momento en el que ocurre tiene un coste muy bajo. El costo crece cuando pasa más tiempo entre que se produce y se detecta. El peor escenario posible es un error introducido al comenzar el proyecto y detectado una vez ya entregado. Esto en una metodología en cascada clásica no era tan raro de ver. De hecho, era desafortunadamente algo común. Hablamos de un error en la toma de requisitos que se detecta después de puesto en producción. Estos errores pueden tener unos costes asombrosos, teniendo que rehacer mucho código. (Huerta, 2018)

La NASA hace un estudio profundo del costo de los errores dependiendo del momento en el que se detectan. Las cifras que se entregan son preocupantes (Stecklein, 2004):

- Un error de software detectado al final del proyecto es aproximadamente 100 veces más caro que uno detectado durante la toma de requisitos.
- Un error de software detectado en producción puede llegar a ser hasta 1000 veces más caro que uno detectado durante la toma de requisitos.
- Si lo errores están relacionados con los sistemas, el factor de coste entre un error detectado al tomar requisitos y en producción puede llegar hasta más de 1.500 veces.

De acuerdo con este planteamiento queremos responder la pregunta ¿Cómo mitigar los costos de defectos en el proceso del desarrollo de software?

Objetivos

Objetivo General

Determinar las formas de disminuir los costos asociados a los defectos generados en el desarrollo de software.

Objetivos específicos

- Identificar en que parte del ciclo del desarrollo de software se originan la mayor cantidad de defectos.
- Buscar nuevas metodologías para la reducción de costos en las pruebas de desarrollo de software.
- Analizar que nuevas actividades contribuyen a la reducción de defectos en el desarrollo de software.

Justificación

Normalmente en los proyectos tradicionales el SDLC (System Development Life Cycle) deja al final ciertos procesos como pruebas, revisiones de seguridad, integraciones entre componentes de software, merges de grandes bloques de código lo cual puede ser un riesgo para la puesta en producción del proyecto o simplemente muy tarde y ahí ya nada que hacer.

¿Qué pasa cuando al finalizar un software se detecta un BUG crítico que requiere días o semanas de desarrollo? o ¿cuándo se consigue una vulnerabilidad alta que requiere un cambio transversal en la arquitectura del software? O ¿cuándo el software a poner en producción no cumple con los criterios de calidad establecida?, toca buscar más tiempo y hacer un refactor que puede costar más tiempo que el inicialmente planeado.

Por esto es importante identificar en que parte del ciclo del desarrollo de software es donde se nos originan la mayor cantidad de defectos, cuando estamos trabajando en un proyecto de software normalmente los detectamos en la parte de pruebas o en la parte productiva y necesitamos que alguien asuma la responsabilidad. Normalmente es el equipo de desarrollo,

asumimos que ahí es donde se origino el error, donde se encubo y muchas veces por el afán, la urgencia la prioridad nos enfocamos en corregir, probar y cerrar. Pero no nos esforzamos en identificar realmente el origen del defecto y de esta manera tomar las acciones preventivas que nos ayuden a mitigar estos defectos y de esta manera reducir costos al momento de corregir defectos.

Marco teórico

La comprensión del software es un requisito previo imperativo e indispensable para las actividades de desarrollo de software, como el mantenimiento, las pruebas y la gestión de la calidad (Krüger, 2019) (Xia, 2017)

A medida que crece un sistema de software, su funcionalidad y las interacciones de los componentes aumentan en tamaño y complejidad. Sin una comprensión adecuada del sistema de software y sus interacciones internas, agregar o cambiar una función aumenta los riesgos de introducir errores y cambios de comportamiento potencialmente indeseables. Este problema se complica aún más cuando se investiga la evolución del sistema, es decir, se comparan los cambios entre varias versiones de software. La evolución del software es responsable del 50%-90% del costo total de desarrollo (Ghezzi, 2002) (Fernández-Sáez, 2018)

Un defecto de software es una desviación de las especificaciones del software o de las expectativas del usuario final y puede generar resultados o fallas impredecibles. El análisis de CISQ (CISQ-HK, 2018) encontró que en 2018, el software de baja calidad costó más de $ 2,8

billones solo en los Estados Unidos, de los cuales el 16,87 % se gastó en fallas conocidas/reparadas, mientras que el 37,46 % fueron pérdidas por fallas de software. Además, esto también demuestra que encontrar y corregir defectos son actividades de desarrollo de software costosas causadas por efectos internos o externos. Los costos internos, como desperdicio, raspado y/o reelaboración, ocurren antes de que el software entre en producción. Los costos externos del software listo para producción incluyen los costos de reproducción, descubrimiento, reparación y verificación de defectos y sus ubicaciones. (Walunj, 2022)

Predicción de defectos y aprendizaje profundo

Se han desarrollado varios métodos para predecir con prontitud las ubicaciones más probables de defectos en grandes bases de código y se pueden categorizar como modelos de clasificación o de regresión (Kamei Y, 2016)

Estos se centran en enfoques que se correlacionan con código potencialmente defectuoso. Las técnicas de Machine Learning han dado servicio a la mayoría de los modelos de redicción de defectos. Esas técnicas derivan varias características del código de software y las limentan a clasificadores estándar como Support Vector Machine, Naïve Bayes, Random Forests y Decision Tree. Los investigadores han estado diseñando cuidadosamente funciones que pueden distinguir el código defectuoso del código no defectuoso, como el tamaño del código, la complejidad del código, las métricas de abandono del código, el cambio de código métricas de proceso (Walunj, 2022)

DebtFree : minimizando el costo de etiquetado en la identificación de deuda técnica autoadmitida usando aprendizaje semi-supervisado

Cuando los desarrolladores apresuran el código, ese código a menudo contiene deuda técnica (TD), es decir, decisiones que luego deben pagarse con más trabajo. Como paso inicial para comprender y resolver los TD, muchas investigaciones (Guo, 2019) detectan primero el TD intencionalmente documentado (a través de comentarios), es decir, el TD autoadmitido (SATD). Los SATD son cruciales para identificarlos ya que están difundidos en la base de código puede sobrevivir a largo plazo Complicar la mantenibilidad del software

Una técnica de aprendizaje no supervisada que aprende patrones a partir de datos no etiquetados es una dirección prometedora en la identificación de SATD. Sin embargo, sin supervisión, la técnica por sí sola puede ser ineficaz. Como se ilustra en la Figura 1 , nuestro enfoque consiste en demostrar primero que los métodos anteriores se pueden ampliar o integrar con el aprendizaje no supervisado para reducir en gran medida el esfuerzo de etiquetado y al mismo tiempo reconocer los SATD de manera efectiva. Este método propuesto, llamado DebtFree (Tu-H & Menzies, 2022)

Figura1: Flujos de trabajo de **DebtFree** = Pseudo-Labeling

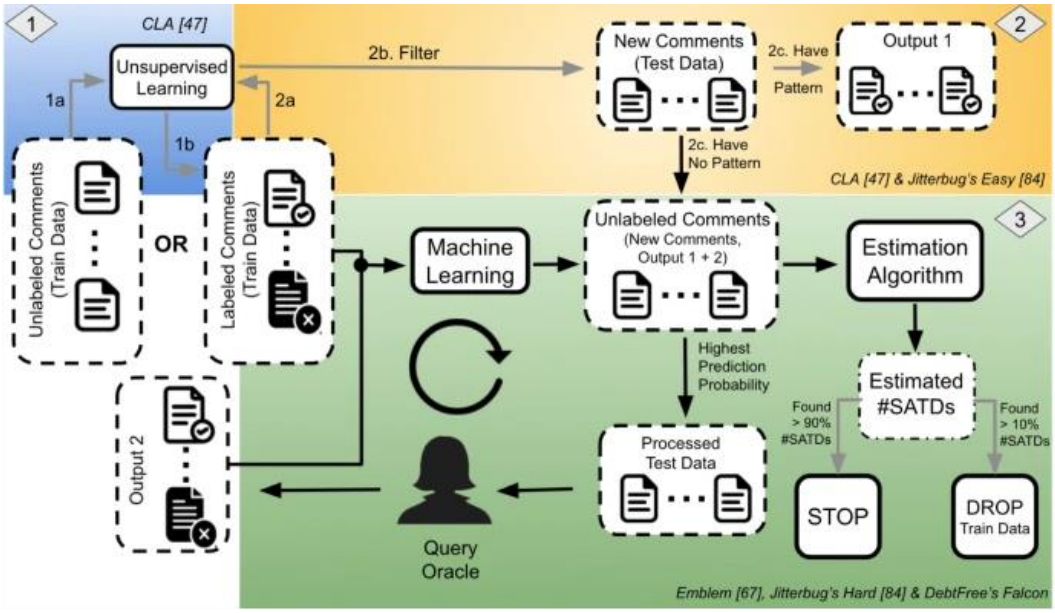

Ilustración 1 DebtFree = Pseudo

Adapatado de DebtFree = Pseudo-Labeling, Nam J, Kim S, 2015, link-springler https://link-springer-com.bdbiblioteca.universidadean.edu.co/article/10.1007/s10664-022-10121-w

Ingeniería a la izquierda.

Normalmente en los proyectos tradicionales el SDLC (System Development Life Cycle) deja al final ciertos procesos como pruebas, revisiones de seguridad, integraciones entre componentes de software, merges de grandes bloques de código lo cual puede ser un riesgo para la puesta en producción del proyecto o simplemente muy tarde y ahí ya nada que hacer. (Alvarez, 2018)

Cuando hablamos de ingeniería a la izquierda lo que buscamos es mover todos estos procesos a etapas tempranas del desarrollo, incluso a la planeación del proyecto, incorporándolos como criterios de aceptación de historias de usuarios y en su DoD (Definition of Done). Los diferentes actores como tester, éthical hackers deben ser invitados a reuniones de gromming para que se puedan detectar todos esos riesgos de manera temprana. Sumado a eso es necesario contar con un

pipeline automatizado que permita generar procesos de integración continua, pruebas automatizadas, pruebas de seguridad automatizadas y despliegues automáticos. (Alvarez, 2018)

De ahí surgen roles como Automatizadores de Pruebas (Test Automation) que buscan crear frameworks de pruebas automatizadas, automatizan pruebas unitarias, pruebas e2e (End to End), pruebas de API, pruebas de performance, entre otras. SecOps que son los que buscan integrar diferentes herramientas de seguridad y los DevOps que se encargan de generar la integración continua, despliegues automáticos y la generación de los KPI y documentación de manera automática. esto se ejecuta automáticamente cada vez que se tiene una versión del software. (Alvarez, 2018)

Todo esto hace sentido cuando se trabaja con metodologías ágiles que promueven la entrega temprana de valor y generar componentes de software de manera periódica. Si se logra esto se logra la generación de valor de manera continua poniendo nuevas funcionalidades del software con una calidad y seguridad innegociable. (Alvarez, 2018)

Metodología

El presente estudio tiene como propósito identificar como mitigar los costos de defectos en el proceso de desarrollo de software. Para cumplir con lo anterior es necesario realizar un estudio de tipo cualitativo-narrativo, que permitan recolectar información de calidad que contribuyan en el alcance de los objetivos de investigación teniendo en cuenta los siguientes aspectos

Enfoque, alcance y diseño de la investigación

A partir del planteamiento del problema, se delimitó un enfoque con perspectiva cualitativa de Tipo narrativa, ya que como lo definen Merriam y Tisdell, este tipo de investigación está orientada a la compresión, exploración y análisis de las experiencias de la población a estudiar, por medio de narrativas sobre sus experiencias de vida. Asimismo, profundizar en la interpretación que estos le dan a las preguntas y circunstancias planteadas. Con respecto a lo anterior, se analizaron las vivencias (como establece el enfoque cualitativo) de los participantes con respecto a sus experiencias en el ciclo de desarrollo de software, se detalla en los objetivos y criterios planteados al inicio de la investigación para así resaltar las problemáticas dentro de las situaciones expuestas.

Variables

Las variables fueron seleccionadas según los requerimientos, el enfoque y el alcance del presente trabajo, son la percepción que tienen los ingenieros de software respecto al origen de los defectos en el ciclo de desarrollo de software y el conocimiento en nuevas metodologías que permitan mitigar los costos de defectos de software.

Un defecto de software, error o simplemente fallo (también conocido por el inglés, bug) es un problema en un programa de computador o sistema de software que desencadena un resultado indeseado y costos.

En términos operacionales con la pregunta ¿En qué etapas del ciclo de vida del desarrollo, se inyectan la mayor cantidad de defectos? Queremos identificar en que etapas del ciclo de

desarrollo de software se inyectan la mayor cantidad de defectos en un software, teniendo en cuenta las fases de Pruebas, Codificación, Diseño y Requisitos

En términos operacionales con la pregunta ¿Conoce las pruebas de desplazamiento a la izquierda "Shift-Left"? queremos identificar si los ingenieros de software conocen esta mueva metodología cuyo objetivo es la mejora de la calidad en el desarrollo de software y la reducción de costos al momento al momento de identificar un defecto o BUG.

Población y muestra

La población que mejor se ajusta al enfoque, alcance y diseño de la investigación es un grupo de Ingenieros con experiencia en el ciclo de desarrollo de software de la ciudad de Bogotá, Colombia. Se toma un tamaño de muestra de 20 personas para una encuesta.

El muestreo utilizado es no probabilístico del tipo deliberado, el cual nos permite tomar como base el conocimiento de los entrevistados o consultados y el propósito de este estudio

En este caso, utilizamos una muestra intencional ya que los entrevistados cumplen con el conocimiento y experiencia específica que son necesarios para realizar la investigación.

Preguntas encuesta.

¿En qué etapas del ciclo de vida del desarrollo, se inyectan la mayor cantidad de defectos?

- Pruebas
- Codificación

- Diseño
- Requisitos

¿Conoce las pruebas de desplazamiento a la izquierda "Shift-Left"?

- Si, claro.
- No, nunca
- He escuchado algo

Análisis de datos

Se realiza entrevista al experto Oscar Urrego, quien es Ingeniero Electrónico de la Universidad Francisco José de Caldas, actualmente se encuentra cursando una Maestría en Ingeniería de Software en la Universidad de los Andes. tiene más de 8 años de experiencia en el área de pruebas, tiene altos conocimientos y experiencia en la parte en la formación de equipos de desarrollo de software.

A continuación, se relacionan los apartados los cuales se consideran de un gran valor teniendo en cuenta los objetivos de la presente investigación.

Pregunta: De acuerdo con su experiencia en que parte del ciclo de desarrollo de software se insertan la mayor cantidad de defectos.

Cuando estamos diseñando es donde se inserta la mayor cantidad, 56% en requerimiento y

20% en diseño, es decir que en las dos primeras fases que es donde se define todo y no se ha escrito una línea de código es donde se encuentran los defectos y es 100 veces más costoso solucionar.

Pregunta: Usted como líder de equipos de desarrollo y teniendo en cuenta que la mayor cantidad de defectos se insertan en los que requerimientos. Que recomienda usted para disminuir estos defectos y así reducir los costos.

Para esto mi recomendación es trabajar con nuevas metodologías como Ingeniería a la izquierda. Básicamente la ingeniería a la izquierda nace o se adoptó el término hace mucho tiempo en el 2001 y fue enfocado principalmente para pruebas de software. Lo que pasa normalmente es que encontramos un defecto en etapas digamos que maduras del proceso del ciclo de vida del desarrollo de software es muy costoso. Ya hemos visto las gráficas de Google de Amazon, hay muchas compañías hay una especialmente en IBM que lo costeo perfectamente y si tenemos precisamente un defecto o encontramos un defecto en la codificación, en ese momento que fue un estudio del 2017 costaba más o menos 80 dólares corregirlo, si lo hacemos en la etapa de construcción de software ya varia aproximadamente a 240 dólares, si lo encontramos en la fase de las pruebas de aceptación en las pruebas de seguridad costaba alrededor de 960 dólares, pero si lo encontrábamos en las etapas de productivas ya cuando desplegamos el cliente cuesta alrededor de 7600 dólares.

La ingeniería a la izquierda es adoptada para metodologías sobre todo las metodologías ágiles, que es lo que nos dice las metodologías agiles, primero hagamos las cosas más

pequeñitas, si seguimos haciendo las cosas grandes no va a ser posible, porque siempre tenemos que probar o analizar un monstruo y si siempre tenemos que analizar un monstruo va a ser demasiado ineficiente, así hagamos automatización, así la automatización la hagamos y la hagamos juiciosamente no va a ser suficientemente, si la seguimos teniendo a la derecha en nuestros procesos. Entonces una recomendación inicialmente y los que nos dictan las metodologías ágiles hagamos piecitas más pequeñas, hagamos cosas más pequeñas que de verdad podamos tener el foco en lo que queremos probar y en la calidad que queremos tener básicamente.

Pregunta: Tú tienes una compañía que tiene un montón de procesos a la derecha que digamos es el proceso tradicional ¿Que le recomiendas usar? ¿Cómo comienza el uso de la Ingeniería a la izquierda?

Lo primero que hay que hacer es sensibilización a todo nivel de las organizaciones llevar esta sensibilización desde los líderes, managers e inversionistas si es necesario hasta el desarrollador o la señora que nos ayuda con el aseo en la compañía, ella también lo tiene que saber, si, ella tiene que tener claro que estos costos son bastantes rudos, eso no es una manera organiza de sacar software con calidad en los tiempos establecidos, tenemos que tener esta política fuertemente arraigada, por otro lado, hay que también ver bien precisamente esas cuestiones técnicas que esto conlleva, digamos que especializar al equipo, siempre precisamente entrenarlo, empezar a unir estas nuevas tendencias que nos permiten probar antes de empezar cualquier cosa.

Pregunta: entonces si alguien dice esto de la Ingeniería a la izquierda es buena idea voy a comenzar a promocionar esto en mi organización, mi compañía ¿Cuáles son los roles que comienzan con esta movida? ¿Una persona, un líder, un developer?

Creo que ingeniería es la más adecuada junto con analytics para mostrar y hacer esa sensibilización si, digamos que son las dos áreas que yo pondría para mover la sensibilización en toda la organización, pero como lo decíamos es un programa cultural tiene que saberlo el presidente, en la primera instancia, hasta la señora de servicios generales que nos ayuda en la organización, creería que tanto ingeniería como analytics que tiene ese peso de los datos y podemos mostrar con excelentes argumentos los beneficios de la ingeniería a la izquierda deberían ser los que comiencen esto.

Pregunta: ¿Cuáles son las complicaciones de este proceso?

Bueno, que hay que empezar a quitar , quitar el rol de QA, a eso hay que quitarlo la calidad es responsabilidad de todos, no de una sola persona, si, y hay que tener equipos especializados de ingeniería, de desarrollo en calidad, hay una encuesta muy chévere en empresas de EEUU que entrevistaron más de las 100 grandes de EEUU y sólo el 19% tiene equipos de ingeniería de calidad, ingenieros de software dedicados a dar lineamientos y a poner las herramientas para los equipos de desarrollo, entonces este movimiento hay que hacerlo con ese tipo de equipos, que cual es el problema que hay, que no son sencillos de conseguir, un ingeniero de software que le guste 100% la calidad es un unicornio en muchas ocasiones y no es fácil.

Evaluación y análisis.

Se generan las siguientes encuestas de acuerdo con la información recopilada producto de la entrevista al experto, con el fin de identificar la percepción respecto a la inyección de defectos de parte ingenieros con experiencia en el desarrollo de software.

Entre los valores observados en la encuesta1 que contiene la respuesta a la pregunta ¿En que etapa del ciclo de vida del desarrollo, se inyectan la mayor cantidad de defectos? el más significativo es el que corresponde a la codificación teniendo una diferencia significativa con los requisitos y diseño.

Encuesta1

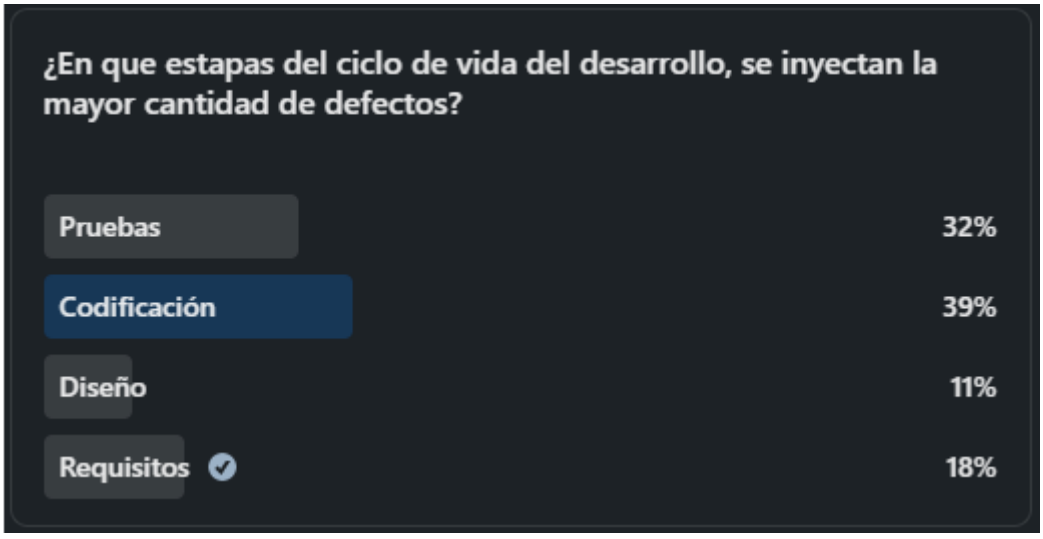

Ilustración 2 Encuesta1 Fuente: Elaboración propia

Entre los valores observados en las encuesta2 que contiene la respuesta a la pregunta ¿Conoce las pruebas de desplazamiento a la izquierda “Shift Left” ?, el único que es significativo es el que corresponde a No, nunca. reconociéndose una diferencia significante en sólo una de las variables

Encuensta2

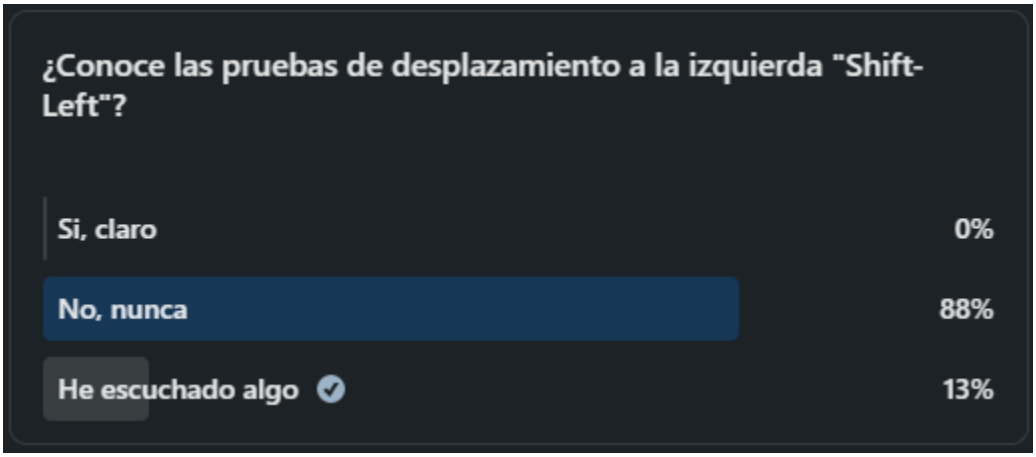

Ilustración 3 Encuesta2 Fuente: Elaboración propia

Discusión de resultados.

Dentro de los objetivos de la presente investigación, queríamos Identificar en que parte del ciclo del desarrollo de software se originan la mayor cantidad de defectos, para esto dentro de la encuesta que realizamos, se realizó la pregunta ¿En qué etapas del ciclo de vida del desarrollo, se inyectan la mayor cantidad de defectos?, a lo que los ingenieros de desarrollo contestaron que estos se originaban en la codificación y las pruebas. Igualmente, el experto entrevistado nos argumentaba que donde se originan la mayor cantidad de defectos es en el diseño o los requerimientos.

Dentro de la encuesta se plantearon dos preguntas, la primera ¿En qué etapas del ciclo de vida del desarrollo, se inyectan la mayor cantidad de defectos?, con esta el objetivo era identificar de acuerdo a la experiencia de los encuetados donde identificaban que se generan la mayor cantidad de defectos, como resultado de esta se da que la mayor cantidad se de defectos se relacionan con la etapa de Codificación y pruebas en su respectivo orden

Dentro de la encuesta planteamos una segunda pregunta. ¿Conoce las pruebas de desplazamiento a la izquierda "Shift-Left"? con esto queremos identificar si para los ingenieros de software estas nuevas metodologías son conocidas, de acuerdo con los resultados de esta pregunta, encontramos que no es conocida.

También dentro de los objetivos de esta investigación queríamos buscar nuevas metodologías para la reducción de costos en las pruebas de desarrollo de software. Para esto encontramos que la Ingeniería a la izquierda es una nueva metodología la cual su objetivo es reducir los costos de defectos realizando pruebas desde el inicio del ciclo de desarrollo de software, esto de acuerdo con lo planteado por el experto entrevistado.

Finalmente, en esta investigación queríamos analizar que nuevas actividades contribuyen a la reducción de defectos en el desarrollo de software. Dentro de las actividades encontradas vemos que realizar pruebas en el inicio del desarrollo de software como lo plantea la Ingeniería a la izquierda contribuyen para aumentar la calidad del software y la reducción de defectos en etapas posteriores como pruebas y/o producción, también que encontramos otra actividad es la automatización de pruebas, esto permitiendo ejecutar escenarios de una manera optimizada y permitiendo generar calidad en el software.

Conclusiones

El presente trabajo de investigación analizó el panorama relacionado con la pregunta de Investigación. Si hay diferentes medidas que se pueden tener cuenta para mitigar los costos de defectos. De acuerdo con la entrevista con el experto se argumentaba que en el ciclo de desarrollo de software donde mas inyectan defectos es en la parte de levantamiento de requerimientos o diseño, que son la parte inicial del ciclo del desarrollo de software. La literatura revisada en el marco de este trabajo reveló que planteamientos para la reducción de costos en el desarrollo de software se enfocan en la parte de codificación, pero no se observa que se trabaje para mejorar la parte de diseño o requerimientos que son la parte inicial del proceso. El planteamiento de usar nuevas metodologías como Ingeniería a la izquierda permitirían innovar en el desarrollo de software con el objetivo de siempre mejorar la calidad.

Se identifica que es necesario investigar mas sobre Ingeniería a la izquierda ya que como se observa en el resultado de las encuestas esta metodología no es muy conocida por los que

ingenieros de software y con el objetivo de mitigar los costos de defectos esta puede ser una buena herramienta para el futuro del desarrollo de software.

Dentro de la encuesta se plantearon dos preguntas, la primera ¿En qué etapas del ciclo de vida del desarrollo, se inyectan la mayor cantidad de defectos?, con esta el objetivo era identificar de acuerdo a la experiencia de los encuetados donde identificaban que se generan la mayor cantidad de defectos, como resultado de esta se da que la mayor cantidad se de defectos se relacionan con la etapa de Codificación y pruebas en su respectivo orden, totalmente opuesto a lo planteado con la Ingeniería a la izquierda que plantea que la mayor cantidad de defectos de inyectan en etapas de diseño y requerimientos que son etapas más tempranas dentro del ciclo de desarrollo de software.

Dentro de la encuesta planteamos una segunda pregunta. ¿Conoce las pruebas de desplazamiento a la izquierda "Shift-Left"? con esto queremos identificar si para los ingenieros de software estas nuevas metodologías son conocidas, de acuerdo a los resultados de esta pregunta, encontramos que no es conocida.

De acuerdo con lo anterior y como resultado de esta investigación encontramos que existen nuevas metodologías que tiene el objetivo de generar software de calidad, y que mitigan los costos de encontrar defecto en etapas maduras del desarrollo de software. Se recomienda para trabajos futuros investigar sobre como se puede implementar de una manera exitosa la ingeniería a la izquierda

Referencias

April, A. (2018). *Software Quality Assurance* (Revised ed.). Wiley-IEEE Computer Society PR.

Ben Charrada, E., Koziolek, A., & Glinz, M. (2015). Supporting requirements update during software evolution. *Journal of Software: Evolution and Process*, *27*(3), 166–194. https://doi.org/10.1002/smr.1705

CISQ HK (2018) The cost of poor-quality software in the us: a 2018 report. https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/The-Cost-of-Poor-Quality-Software-in-the-US-2018-Report.pdf

Fernández-Sáez AM, Chaudron MR, Genero M (2018) An industrial case study on the use of uml in software maintenance and its perceived benefits and hurdles. Empir Softw Eng 1–65

Ghezzi C, Jazayeri M, Mandrioli D (2002) Fundamentals of software engineering. Prentice Hall PTR

Guo, L., Lei, Y., Xing, S., Yan, T., & Li, N. (2019). Deep Convolutional Transfer Learning Network: A New Method for Intelligent Fault Diagnosis of Machines With Unlabeled Data. *IEEE Transactions on Industrial Electronics*, *66*(9), 7316–7325. https://doi.org/10.1109/tie.2018.2877090

Haskins, B., Stecklein, J., Dick, B., Moroney, G., Lovell, R., & Dabney, J. (2004). 8.4.2 Error Cost Escalation Through the Project Life Cycle. *INCOSE International Symposium*, *14*(1), 1723–1737. https://doi.org/10.1002/j.2334-5837.2004.tb00608.x

Huerta, J. M. (2018). *Coste de los errores en proyectos de software*. Jose Huerta. https://josehuerta.es/gestion/proyectos/calidad/coste-de-los-errores-en-proyectos-de-software

Kamei Y, Shihab E (2016) Predicción de defectos: logros y desafíos futuros. En: 2016 IEEE 23rd conferencia internacional sobre análisis, evolución y reingeniería de software (SANER), vol 5, pp 33–45

Krüger J (2019) Tackling knowledge needs during software evolution. In: Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 1244–1246

Kruger, J., Koziolek, A., & Glinz, M. (2015). Supporting requirements update during software evolution. *Journal of Software: Evolution and Process*, *27*(3), 166–194. https://doi.org/10.1002/smr.1705

Losee, B. (1985). An information systems manifesto. *Information Processing & Management*, *21*(4), 372. https://doi.org/10.1016/0306-4573(85)90068-8

Merriam, S., & Tisdell, E. (2019). The Development of Questioning Skills through <i>Hikmah</i> (Wisdom) Pedagogy. *Creative Education*, *10*(12), 2593–2605. https://doi.org/10.4236/ce.2019.1012187

Merriam, S. B. (2009). Qualitative research: A guide to design and implementation. San Francisco, CA: Jossey-Bass.

Tu, H., Menzies, T. DebtFree : minimización del costo de etiquetado en la identificación de deuda técnica autoadmitida mediante el aprendizaje semisupervisado. Empir Software Inglés 27, 80 (2022). https://doi-org.bdbiblioteca.universidadean.edu.co/10.1007/s10664-022-10121-w

Walunj, V., Gharibi, G., Alanazi, R., & Lee, Y. (2022). Defect prediction using deep learning with Network Portrait Divergence for software evolution. *Empirical Software Engineering*, *27*(5). https://doi.org/10.1007/s10664-022-10147-0

Xia X, Bao L, Lo D, Xing Z, Hassan AE, Li S (2017) Measuring program comprehension: a large-scale field study with professionals. IEEE Trans Softw Eng 44(10):951–976