UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"DISEÑO E IMPLEMENTACIÓN DE UNA ESTRATEGIA DE GESTIÓN PARA MEJORAR LA EFECTIVIDAD DE LAS PRUEBAS DE REGRESIÓN EN EL SOFTWARE"

TESIS

PARA OBTENER EL GRADO DE
MAESTRO EN INGENIERÍA EN SOFTWARE

PRESENTA

ING. EDGAR LÓPEZ CRUZ

DIRECTOR DE TESIS
DR. IVÁN ANTONIO GARCÍA PACHECO

CODIRECTORA DE TESIS
DRA. CARLA LENINCA PACHECO AGÜERO

HUAJUAPAN DE LEÓN, OAX. A 16 DE FEBRERO DE 2026

Tesis presentada el 16/02/2026 ante los siguientes sinodales:

Dr. Ignacio Arroyo Fernández

M.E.C. Josué Neftalí García Matías

M.M.I. Juan Gabriel Ruiz Ruiz

M.T.C.A. Omar Martínez Osorio

Director de tesis:

Dr. Iván Antonio García Pacheco

Codirectora de tesis:

Dra. Carla Leninca Pacheco Agüero

Dedicatoria

A *Robin Hugo López Osorio* y *Justina Blanca Cruz Zárate*, gracias por darme todo para construir mi propio camino; lo que soy y lo que he alcanzado nace de su esfuerzo. Juntos lo hemos logrado.

Agradecimientos

A mis hermanos y hermanas, por ser mi equipo de vida y mi respaldo seguro. Gracias por estar ahí, por las risas que aligeraron la carga en los momentos pesados y por recordarme que, pase lo que pase, siempre contamos los unos con los otros.

A mi entrañable amigo, M.E.D.I. Heriberto Ildefonso Hernández Martínez, por tu amistad incondicional y lealtad a prueba de todo. Gracias por estar presente en las buenas y en las malas, y por ser esa compañía firme en tantas etapas de mi camino. Este logro también celebra nuestra amistad, ya que fuiste tú quien me motivó a comprometerme con este objetivo que el día de hoy he culminado.

A mi director y codirectora de tesis, Dr. Iván Antonio García Pacheco y Dra. Carla Leninca Pacheco Agüero, por su orientación, paciencia y compromiso arduo durante este proceso. Gracias por compartir su experiencia conmigo y por guiarme para darle forma y sentido a este proyecto. Su criterio fue fundamental para llegar a este resultado.

A las y los directivos, programadores, *testers*, jefes de proyectos, y analistas de todas las organizaciones que participaron desinteresadamente en la evaluación empírica presentada en esta tesis.

A mi mejor amiga Lore, por tener siempre las palabras motivadoras y el oído dispuesto para mí. Gracias por escucharme en los momentos de estrés, por calmar mis dudas y por celebrar mis avances con tanta alegría como si fueran tuyos.

A todas aquellas personas quienes creyeron en mí, amigos y compañeros que confiaron en mi capacidad y me impulsaron con sus palabras de ánimo. Gracias por apostar por mí, su confianza fue razón para no detenerme.

Finalmente, a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI), antes Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT), por brindarme el apoyo económico durante el periodo que comprendió la realización de mis estudios de maestría.

Índice

Lista de tablas

Lista de figuras

Resumen

Las pruebas de regresión que se realizan durante el desarrollo de un producto de software garantizan que las modificaciones realizadas a este, no generen nuevos errores que afecten las funcionalidades que ya habían sido implementadas. Por lo tanto, estas pruebas son esenciales para mantener la estabilidad y la calidad del software a medida que éste evoluciona a lo largo del tiempo. En este sentido, dado que la gestión de las pruebas de regresión es una parte esencial del ciclo de vida del desarrollo de software, es necesario definir una estrategia para planificarlas y controlarlas de manera efectiva. Sin embargo, la definición de una estrategia para gestionar las pruebas de regresión a menudo es realizada por organizaciones maduras que disponen de recursos financieros, personal capacitado, herramientas de soporte y experiencia, dificultando así su uso en las pequeñas organizaciones desarrolladoras de software. Por lo tanto, esta tesis propone el diseño e implementación de una estrategia de gestión para mejorar la efectividad de las pruebas de regresión en el contexto de las pequeñas organizaciones desarrolladoras de software. Se plantea, de esta manera, una estrategia basada en la combinación efectiva de tres elementos que establecen niveles apropiados de calidad en un proceso de software: la gestión del conocimiento, la mejora de los procesos de software, y el soporte de una herramienta computacional.

1. Introducción

El Capítulo 1 de esta tesis proporciona evidencia sobre la existencia de una problemática importante que limita las posibilidades de las organizaciones desarrolladoras de software de pequeño tamaño para planificar, ejecutar y realizar las pruebas de regresión. Por lo tanto, se establecen las bases de la investigación a través de limitaciones, delimitaciones, hipótesis y objetivos.

1.1. Contexto del problema

De acuerdo con Pressman y Maxim (2019), el software puede verse como una entidad intangible conformada por componentes lógicos, reglas y datos que permiten que una computadora realice tareas específicas, en contraposición de los componentes físicos denominados *hardware.* De hecho, una característica fundamental para diferenciar el software del hardware es que "*el software no se desgasta"*. Sin embargo, Sommerville (2021) menciona que el desarrollo de los productos de software muchas veces termina siendo un rotundo fracaso, puesto que las técnicas y los métodos de la Ingeniería de Software (IS) no son siempre los apropiados o no encajan de forma correcta en cada proyecto. Por lo que es fundamental que dentro de la IS se desarrollen alternativas que faciliten la generación de productos de software que sean fiables y seguros, dado que el tener una IS exitosa permitiría el desarrollo de sistemas escalables (i.e., desde sistemas básicos hasta sistemas complejos) con calidad y que sean entregados a tiempo.

En la actualidad, desafortunadamente muchos de los proyectos de desarrollo de un producto de software terminan convirtiéndose en retos enormes (Pressman y Maxim, 2019). A pesar de esta situación, el software sigue siendo parte importante del día a día de las personas, ya que prácticamente se encuentra presente en muchos aspectos de la vida cotidiana. De hecho, las personas, sectores de la industria, empresas, gobiernos, y demás organizaciones, tienen una dependencia cada vez mayor con el software, lo que claramente conlleva a entender que, si éste tiene una baja calidad, se padecerán consecuencias negativas que pueden llegar a ser, incluso, graves. Por lo tanto, es crucial hacerles frente para mejorar considerablemente la calidad del desarrollo de software (Salahat et al., 2023).

En este sentido, existen distintas formas de contextualizar el significado del término "calidad del software", y esto se debe a las diferentes perspectivas que cada participante tiene desde sus necesidades particulares. Algunas de las definiciones de "calidad del software" más conocidas son aquellas proporcionadas por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, por sus siglas en inglés) y por algunos de los autores más relevantes de la literatura especializada sobre la IS, como lo son Roger Pressman y Bruce Maxim:

- Para el IEEE, la calidad del software es "*el grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario*" (IEEE, 2014).

- Para Pressman y Maxim (2019), la definición se refiere a "*la calidad de un sistema, aplicación o producto sólo es tan buena como los requerimientos que describen el problema, el diseño que modela la solución, el código que conduce a un programa ejecutable y las pruebas que ejercitan al software para descubrir errores*".

Como se puede observar, cada una de estas definiciones establece criterios diferentes que son perfectamente adaptables al contexto del desarrollo de un producto de software. Ahora bien, dado que el software es intangible no permite visualizar su calidad como la de cualquier otro producto que es fabricado a partir del ingenio humano. Por lo tanto, "la calidad del software" también es intangible y regularmente se asocia con el uso y la consolidación de buenas prácticas con respecto a la IS. En este sentido, Farley (2021) argumentó que la calidad en los servicios y/o productos de software se puede lograr a través de la aplicación de cuatro principales elementos: métodos de IS, técnicas de gestión de proyectos, acciones de control de calidad, y aseguramiento de la calidad del software. La investigación de Ndukwe et al., (2023) amplió esta visión argumentando que la calidad del software debe verse desde cuatro enfoques que se describen de la siguiente manera:

- Calidad del proceso. Involucra las actividades, tareas, entradas/salidas y todos aquellos procedimientos que impliquen desarrollar y mantener el software. Bajo esta premisa, se entiende que el apegarse a las buenas prácticas descritas en las normas, modelos y/o metodologías asegura la incorporación de calidad a un proceso de desarrollo de software.
- Calidad del producto. Establece el uso de diferentes medidas con el fin de asegurar la calidad del software visto como un producto. Entre las más destacadas, por ejemplo, se encuentran las establecidas por el estándar ISO/IEC 25010:2023 (ISO, 2023) para la definición de características de calidad como la funcionalidad, confiabilidad, usabilidad, eficiencia, capacidad de mantenimiento, y portabilidad. Es importante enfatizar que, de acuerdo con Aizprua et al. (2019), la calidad del producto no depende totalmente de la utilización de un estándar, sino más bien de cómo se aplican las medidas que el documento especifique.
- Calidad del equipo. Establece que la influencia del equipo de desarrollo para el éxito de un proyecto es crucial, ya que se trata de las personas encargadas de desarrollar el software. En este sentido, las herramientas, técnicas y demás procesos implicados son de utilidad para el equipo de desarrollo, ya que les facilita en gran medida el trabajo. Sin embargo, dichas personas son un componente no lineal clave para el desarrollo del software.
- Calidad del desarrollo del software. Introduce la idea de que la calidad en el desarrollo de software implica la consideración de un conjunto de características del servicio/producto producido que permiten satisfacer las expectativas del cliente o usuario (ISO 9000, 2015; ISO/IEC 5055, 2021).

Considerando lo anterior, es importante mencionar que el concepto de calidad ha trascendido a lo largo de la historia y ha influido en varios aspectos de la vida del ser humano. Por ejemplo, con la llegada de la primera guerra mundial, se prestó mayor atención al aspecto de la calidad del armamento, por lo que se inició el control de la calidad mediante la inspección. Una vez que se logró cierta estabilidad en las inspecciones, se inició la concepción formal del desarrollo de la calidad, donde el eje central se basaba en la automatización de dicha inspección. Posteriormente, durante 1920 y 1940, las empresas de tecnología industrial como lo eran *Bell System* y *Western Electric*, lideraron el mercado bajo el esquema de control de calidad. Sin embargo, debido a los tantos defectos encontrados en sus productos, tomaron la decisión de integrar departamentos de ingeniería de inspección para tratar

de solventar todos los inconvenientes que se les presentaban. En 1924, uno de los líderes del departamento de ingeniería de inspección, el matemático Walter Shewhart, fue el encargado de diseñar una gráfica estadística con la finalidad de identificar, primero, y posteriormente, controlar las variables de los productos desarrollados. Este avance contribuyó a la evolución del "control de la calidad por inspección" a "control estadístico de la calidad". Con el surgimiento de este enfoque, se generó la posibilidad de introducir el control de la calidad en los medios de producción en serie (Galin, 2018), puesto que se mejoraba considerablemente el costo-beneficio ya que se elevaba la producción y se reducía lo más posible los errores en las líneas de producción (Horcas et al., 2019). De hecho, una de las aportaciones más significativas, de acuerdo con el mismo Shewhart, no solo fue mejorar la calidad de los productos sino también el surgimiento de un rol administrativo. Esta evolución condujo al diseño del ciclo PHVA (Planear-Hacer-Validar-Actuar), que años más tarde sería conocido bajo el nombre de Ciclo de Deming (Dudin et al., 2015), y el cual sigue siendo utilizado en la actualidad para introducir los sistemas de gestión de la calidad en los procesos productivos.

A inicios de la segunda guerra mundial, la industria consideraba el control estadístico de la calidad como un instrumento importante lo que conllevó a un crecimiento gradual. En la década de los 50', justo después de la segunda guerra mundial, se empezaron a reportaron avances importantes en el desarrollo de software, por ejemplo, en los EE. UU. se destinaron recursos de tiempo y dinero al desarrollo de sistemas para el control de tiro y de navegación aérea. A diferencia del hardware, que en aquella época estaba dedicado a cada una de las distintas aplicaciones que se creaban, la evolución del software ya era una parte medular de los sistemas y condujo a la realización de pruebas exhaustivas que buscaban gestionar de cierta manera su calidad, una vez que se había finalizado su desarrollo. A pesar de que no se tenían registros documentales sobre las fallas y los errores que se cometían, sí se registró información que evidenciaba la estabilidad de los requisitos en los sistemas, algo totalmente diferente a la mayoría de los desarrollos de software de la actualidad. Posteriormente, aproximadamente entre 1963 y 1966, se destacó un suceso importante que marcaría un paso importante para la evolución de la calidad: IBM desarrolló un sistema operativo capaz de garantizar la calidad de los productos de software a partir de los problemas que surgían debido al sobre presupuesto y los tiempos adicionales para poder finalizar con los proyectos (Chopra, 2018).

Dado este acontecimiento, las investigaciones condujeron a una nueva actualización en el desarrollo de la calidad, llegando a lo que se le conoce actualmente como "aseguramiento de la calidad" (Moyano-Hernández y Sandoval, 2021). De acuerdo con el estándar IEEE 1012 (2016), el de la Calidad del Software (SQA, por sus siglas en inglés) proporciona "*un patrón planificado y sistemático de todas las acciones necesarias para brindar la confianza necesaria de que un producto de software cumple con los requisitos técnicos establecidos*". En este sentido, el proceso de SQA involucra un conjunto de actividades que permite que todos los involucrados en un proyecto de software hayan implementado correctamente todos los procedimientos y procesos (Maxim y Kessentini, 2016).

Sin embargo, y a pesar de toda esta evolución, la mayoría de los desarrolladores de software no cuentan con un panorama claro sobre la estimación del esfuerzo que implica el realizar un proceso de SQA. En este sentido, Anand y Udin (2019) argumentaron que era necesario realizar pruebas al software conforme el equipo de desarrollo avance en la integración de nuevos cambios durante el desarrollo, ya que esperar hasta que el código esté terminado es un error grave. En consecuencia, después de haber corregido los errores detectados en las pruebas, es necesario volver a probar, ya que es posible no haberlos corregido todos o bien que, durante la corrección, se hayan introducido nuevos errores. Es decir, no existe la certeza de que, durante la realización de pruebas a lo largo del ciclo de vida del software, no exista error alguno; no obstante, la automatización de las pruebas puede conducir a mejores resultados (Gamido y Gamido, 2019). Durante el ciclo de vida del desarrollo de software,

por ejemplo, es necesario realizar cambios constantemente debido a la necesidad de corregir los errores que no fueron detectados en las fases anteriores o bien para introducir mejoras al producto. Así pues, dadas las frecuentes modificaciones que afectan, de forma positiva o negativa, al funcionamiento del software, éste debe trascender con el propósito de competir considerando la demanda actual en el mercado (Palomo y Gil, 2020).

1.2. Importancia del problema

Como parte del proceso de SQA, las pruebas de regresión en el software son utilizadas por las organizaciones como una estrategia para asegurar la calidad del producto. Esta estrategia se aplica en una versión modificada del software, es decir, se utiliza el conjunto de pruebas ejecutado en la versión anterior. De acuerdo con Qasim et al. (2021), esto conlleva a la reejecución de las pruebas de regresión durante cada iteración, ejecutando de forma repetitiva la prueba anterior durante la nueva iteración, todo ello con la finalidad de asegurar que los defectos identificados y corregidos no hayan introducido nuevos errores que provoquen anomalías en los productos de software. Sin embargo, Azizi (2021) argumentó que no todos los productos de software están sujetos a la ejecución de todo el conjunto de pruebas o de las mismas pruebas previamente ejecutadas.

En este contexto, la investigación de Abdulwareth y Al-Shargabi (2021) mostró que la automatización podría ayudar a reducir, de forma considerable, el esfuerzo requerido para realizar las pruebas de regresión sobre productos de software que requieren de un mantenimiento frecuente, con el fin de no afectar la calidad del producto ante los cambios. En consecuencia, las pruebas de regresión permiten que las organizaciones identifiquen aquellas diferencias introducidas desde la última versión del software para clasificarlas en defectos o mejoras (Thota et al., 2020) y, de esta manera, reducir el riesgo de obtener resultados imprevisibles y problemas funcionales como consecuencia de la agregación de nuevo código. Así pues, la importancia de realizar este tipo de pruebas reside en la evaluación de los nuevos cambios o modificaciones, puesto que permiten verificar que no se hayan introducido defectos que impacten negativamente en el rendimiento del software, reducen el tiempo y costo en el desarrollo de futuras versiones del software, y mejoran su calidad (Kandukuri, 2020). De hecho, Bertolino et al., (2020) argumentaron que las pruebas de regresión son capaces de mostrar fallos corregidos en versiones anteriores del software, que resurgen por las modificaciones que se hayan hecho al código.

Con el objetivo de resaltar la importancia que tienen las pruebas de regresión en la actualidad, la Tabla 1 presenta algunas ventajas y desventajas que han sido reportadas en la literatura especializada.

Tabla 1. Ventajas y desventajas de utilizar las pruebas de regresión en la industria.

Ventajas	Desventajas
Las pruebas de regresión garantizan que, una vez que se han agregado nuevas funcionalidades en el software, éstas operen de forma correcta, reduciendo así el riesgo de obtener resultados inesperados (Rehan, 2021).	La pérdida de tiempo, si se realizan pruebas de regresión manuales, ya que los *testers* necesitan de una formación adecuada para familiarizarse con el sistema que se requiere probar (Fisal et al., 2021).

Ventajas	Desventajas
El probar funcionalidades anteriores coadyuva a garantizar que el software siga funcionando adecuadamente. Además, al llevar a cabo esta actividad se mejora su calidad y, por ende, se disminuye la probabilidad de introducir errores en cada versión posterior del mismo (Shin et al., 2018).	Se pueden ocasionar retrasos en busca de una solución que reemplace las pruebas exhaustivas[1], única opción que permite cuantificar el impacto de los cambios en el sistema (Jung et al., 2020).
Realizar pruebas de regresión adecuadas asegura que cada nueva característica liberada no contenga problemas funcionales. Con ello se reduce el tiempo entre cada liberación para llevar a cabo pruebas beta con los usuarios de una forma más rápida (Samad et al., 2021).	En muchas ocasiones es inasequible identificar un caso de prueba o entorno que permita mostrar los errores en las nuevas versiones, a causa de los cambios realizados en el código (Kulkarni, 2021).
Las pruebas de regresión facilitan el mantener la compatibilidad entre distintas versiones del software (Chaudhary et al., 2023).	Si el conjunto de pruebas de regresión identifica un mayor número de errores que los introducidos en la nueva versión del software, se puede generar un problema importante, ya que el proceso de corrección se extenderá y por ende el plazo de entrega será más largo (Panda, 2020).
Las pruebas de regresión reducen el tiempo de desarrollo de nuevas versiones del software al identificar defectos de forma rápida, eliminando posibles retrasos en la liberación por la disposición de recursos (Ali et al., 2019).	Si después de una nueva actualización el conjunto de pruebas de regresión no puede identificar los defectos corregidos, se puede ocasionar el retraso de la nueva liberación o, en el peor escenario, generar quejas por parte de los usuarios (Sidhu y Sehra, 2022).

De acuerdo con Steve McConnell[2], la menor modificación al código de un software puede generar errores, faltas o fallos en el mismo y, por ende, surge la necesidad de priorizar y ejecutar pruebas de regresión que identifiquen funcionalidades críticas o aquellas en las que se hayan realizado cambios. De acuerdo con este especialista, un caso ideal sería el realizar pruebas de regresión a todas y cada una de las funcionalidades. Sin embargo, en el mundo real esto es imposible debido al costo y tiempo que dicha tarea implicaría para las organizaciones desarrolladoras de software, en particular para aquellas que son denominadas como "pequeñas"[3]. Esto conduce a pensar que, a pesar de todos los avances en la detección de fallos en el software, este tipo de organizaciones aún enfrenta problemas importantes para establecer un proceso de SQA que sea eficiente puesto que, independientemente de su tamaño, ninguna organización de software está eximida del compromiso que se tiene de generar

[1] Las pruebas exhaustivas no son más que el proceso de probar en el software cada una de las entradas válidas e inválidas tomando en cuenta todas las posibles condiciones (Ponzio et al., 2016).

[2] Steve C. McConnell es autor de diversos libros de texto sobre la Ingeniería de Software. Sus aportaciones a la escritura limpia de código lo llevaron a ser nombrado una de las tres personas más influyentes en la industria de software junto con Bill Gates (fundador de Microsoft) y Linus Torvals (creador de Linux).

[3] De acuerdo con la Secretaría de Economía de México, una "pequeña organización" es una compañía que cuenta con una plantilla de entre 11 y 50 trabajadores para desarrollar productos y/o servicios con el fin de producir ingresos que van de los $4,000,000 a los $100,000,000. En el contexto del desarrollo de software en el país, este tipo de organizaciones representa entre el 95% y 99% de la oferta existente el mercado.

productos de calidad (Faustino y Mejía, 2020). Esta situación es de una importancia significativa si se asume que una inmensa mayoría de las organizaciones desarrolladoras de software en México (entre el 95% y 99%), y prácticamente en todo el mundo, son precisamente de pequeño y mediano tamaño, las cuales se caracterizan, lamentablemente, por no contar con procesos definidos puesto que el éxito de los proyectos que emprenden depende en gran medida del "heroísmo" de los empleados (Micheli y Oliver, 2017; Mejía et al., 2022). Aunado a esto, la mayoría de estas organizaciones carecen de recursos, presentan dificultades para autoorganizarse y difícilmente pueden generar software de calidad dentro del tiempo y presupuesto establecidos. De hecho, de acuerdo con Tuape y Ayalew (2019) y Ragkhitwetsagul et al. (2022), esta problemática provoca que dichas organizaciones enfrenten altos índices de fracaso o excedan el uso de recursos en el retrabajo para cada proyecto de software, lo que conduce a menudo a incrementar el esfuerzo y por ende la inversión, afectando así negativamente el retorno de esta. Por lo tanto, es necesario realizar investigaciones que permitan que este tipo de organizaciones maduren e incrementen la calidad de sus productos a través de propuestas que se ajusten a su forma de trabajo.

Considerando específicamente la realización de pruebas de regresión en sus productos de software, es importante mencionar que estas organizaciones carecen de especialistas que empleen técnicas y/o enfoques que faciliten la obtención de beneficios en el corto y largo plazo, puesto que a menudo es difícil adaptarlas a sus características específicas (Tuape et al., 2021). Peor aún, el uso de enfoques actuales como el Desarrollo Guiado por las Pruebas (TDD, por sus siglas en inglés) (Bakhitiary et al., 2020), la minimización de casos de prueba (Bajaj y Sangwan, 2020), la selección de casos de prueba (Ma et al., 2021), la priorización de casos de prueba (Mukherjee y Patnaik, 2021), la incorporación de la Inteligencia Artificial a través del uso de algoritmos de aprendizaje computacional (Marijan, 2023), redes neuronales (Lin et al., 2020), heurísticas de búsquedas (Khari et al., 2020), o incluso la programación en pares (Liu et al., 2020); que conducen a la optimización de los casos de prueba, a menudo es difícil e incluso imposible de realizar. Esto representa un vacío importante relacionado con la manera en que se realizan las pruebas de regresión en estas organizaciones; sin embargo, existe evidencia que expone una problemática más profunda. Por ejemplo, la investigación de Rojas-Montes et al. (2015), enfatizó el hecho de que las pruebas de regresión realizadas en las organizaciones de pequeño tamaño ocupan un 30% del esfuerzo total destinado a las pruebas del software. Dicho esfuerzo se suele realizar tanto para cada ocasión en que se realiza un cambio en el código, lo que puede provocar que se introduzcan nuevos defectos en otros módulos, como para cuando se pretende realizar una mejora a una funcionalidad específica. Bajo este escenario, se asume que la dedicación de tanto esfuerzo se debe, en gran medida, a que el *tester* no realiza un análisis previo de los elementos a probar, no planifica y no establece una política de pruebas. Es decir, se carece de una estrategia de gestión adecuada que permita a estas organizaciones el definir y simplificar la ejecución de las pruebas de regresión.

De manera similar, la investigación de Blanquicett et al., (2018) argumentó que, dado que la industria de software funciona con tiempos de entrega bastante limitados, en muchos casos las pequeñas organizaciones omiten la definición y realización de una estrategia para las pruebas de regresión con el fin de evitar retrasos que comprometan al proyecto. Como una posible solución, estos investigadores consideran que es necesario que estas organizaciones dispongan de herramientas computacionales que les permitan definir una estrategia de pruebas y que les faciliten, al mismo tiempo, gestionar tales pruebas a través de, por ejemplo, la generación de reportes y el intercambio de información.

En este mismo orden de ideas, Khaleel y Anan (2023) argumentan que aún existen desafíos importantes que obstaculizan la planificación e implementación de estrategias para gestionar las pruebas de regresión durante el desarrollo de software dentro de este tipo de organizaciones. Entre

estos obstáculos se encuentra la falta de motivación, tanto de los *testers* como de los desarrolladores, puesto que se considera erróneamente que las pruebas de regresión no proporcionan un beneficio inmediato a la organización, por lo que, al no gestionarlas, no se controla el tiempo necesario para su realización, lo que conlleva a descuidar actividades críticas que pueden generar riesgos y, como consecuencia, problemas. En este sentido, el realizar las pruebas de regresión de forma tradicional o manual en las pequeñas organizaciones desarrolladoras de software, representa altos costos de mano de obra; lo que básicamente implica subsidiar el pago de más de un *tester* para ejecutarlas. Aunado a esto, sin una estrategia de gestión, los costos se incrementan considerablemente puesto que no existe un control sobre qué pruebas proporcionarán mayor valor, considerando el tiempo y los recursos con los que se cuenta en ese momento. Por otro lado, la visibilidad en cuanto a las medidas y los objetivos que se pretenden lograr con las pruebas de regresión no son claras o no están completamente definidos en estas organizaciones. Esto por su parte implica poner en duda si los *testers* que realizan las pruebas de regresión lo hacen de forma adecuada, lo cual es evidente que causará impacto en la estabilidad y la calidad de los productos de software que se crean.

1.3. Necesidad de solución

De acuerdo con las investigaciones realizadas por Parsons et al., (2014) y Sharma y Kumar (2022), a pesar de que se han realizado diversos esfuerzos por automatizar las actividades que conllevan a la realización de las pruebas de regresión, específicamente las operativas u organizativas, aquellos aspectos que podrían considerarse estratégicos (e.g., políticas de adopción, gestión, evolución, medición) han sido estudiados en menor medida. Por añadidura, tales investigaciones no se enfocan en las necesidades de las pequeñas organizaciones desarrolladoras de software.

Considerando lo anterior, Ali et al., (2019) argumentaron que la definición de una estrategia de pruebas de regresión variará según factores del contexto externo, como los tipos de productos y niveles de cumplimiento normativo, y factores del contexto interno, como la filosofía de las pruebas y la madurez de los equipos y organizaciones. Por lo tanto, es importante considerar que estos factores contextuales impactan en aspectos de la práctica como la inversión en infraestructura y la gestión del cambio y el riesgo.

Así, Govil y Sharma (2021) consideran que debido a que la definición de una estrategia efectiva para maximizar los beneficios de las pruebas de regresión no es una tarea fácil, surge la necesidad de diseñar mecanismos que consideren a los riesgos y prioricen los conjuntos de pruebas con la finalidad de reducir el tiempo de ejecución de tales pruebas y mejorar, como consecuencia, la motivación de los *testers* para optimizar su esfuerzo. En este sentido, con el objetivo de facilitar la definición, selección e implementación de estrategias para realizar y gestionar las pruebas de regresión, diferentes investigaciones han creado propuestas que bien podrían indicar una pauta para comenzar con una investigación enfocada a las organizaciones desarrolladoras de software de pequeño tamaño.

La investigación de Kandil et al. (2015), por ejemplo, introdujo la posibilidad de adoptar una estrategia para la realización y gestión de las pruebas de regresión en entornos ágiles de desarrollo que se basó en dos métodos específicos: (1) la reducción del conjunto de casos de prueba, que considera las similitudes que existen entre los proyectos y los distintos casos de prueba que se generan a partir de las historias de usuario, y (2) la priorización de los conjuntos de prueba ya reducidos mediante el uso de parámetros ágiles ponderados provenientes del usuario. En este sentido, la reducción del conjunto de pruebas se introdujo con el propósito de disminuir el tiempo y esfuerzo requeridos para realizar las pruebas de regresión. Esta estrategia conlleva a identificar el subconjunto de casos de prueba que mostrará una mayor eficiencia que la que posee el conjunto original. Los

resultados de implementar dicha estrategia en tres diferentes conjuntos de pruebas fueron recogidos a través de tres medidas: el porcentaje de Reducción del Conjunto de Pruebas (TSR, por sus siglas en inglés), la Pérdida de Capacidad de Detección de Fallos (FDC, por sus siglas en inglés), y el Porcentaje Promedio de Detección de Fallos (APFD, por sus siglas en inglés). Los datos recogidos demostraron una mejora promedio del TSR del 6%, mientras que la FDC se mantuvo en un promedio de 96.5% para los tres conjuntos de datos utilizados. En cuanto a la priorización, los resultados mostraron una mejora del APFD en un 0.802, utilizando diferentes ponderaciones para los parámetros proporcionados para los tres conjuntos de datos diferentes. En conclusión, se puede decir que la contribución significativa de esta investigación fue la definición de medidas o parámetros ágiles para evaluar las pruebas de regresión en proyectos ágiles, llevadas a cabo específicamente mediante la combinación de la reducción y priorización de los conjuntos de pruebas.

La investigación de Romano et al., (2018) presentó un enfoque de selección de pruebas de regresión basado en la recuperación de información simple, denominado SPIRITuS por sus siglas en inglés. Dicho enfoque utilizaba información de cobertura de los métodos de codificación y un modelo de espacio vectorial para seleccionar los casos de prueba que se debían ejecutar. La experimentación con este enfoque se realizó con 389 versiones defectuosas de 14 programas de código abierto implementados en lenguaje Java©. Básicamente se buscó analizar la compensación entre el número de casos de prueba seleccionados del conjunto de pruebas original y la efectividad de la detección de fallos. En este sentido, SPIRITuS utilizó una cantidad de casos de prueba significativamente menor que la seleccionada por otros enfoques utilizados, mostrando una ligera reducción en la capacidad de detección de fallos.

Por otro lado, la investigación de Fisal et al., (2021) propuso un algoritmo Binario de Murciélago[4] Adaptado Multiobjetivo (ABBA, por sus siglas en inglés) para resolver el problema de la TSR. En este sentido, el Algoritmo Binario Original de Murciélago (OBBA, por sus siglas en inglés) fue adaptado para mejorar sus capacidades de exploración durante la búsqueda de una superficie óptima de Pareto. La eficacia del ABBA se evaluó utilizando seis programas Java© de diferentes tamaños. Los resultados experimentales mostraron que, para la misma tasa de descubrimiento de fallos, el ABBA es capaz de reducir el tamaño del conjunto de pruebas más que el OBBA y el algoritmo de Optimización del Enjambre de Partículas Binarias (BPSO, por sus siglas en inglés).

De manera similar, Song et al., (2022) propusieron un enfoque para automatizar el proceso de pruebas de regresión, a partir del historial de evolución del código. Dicha estrategia se basó en los fallos encontrados, ya que éstos manifiestan el cómo se introduce y corrige un error e incorporan más especificaciones al sistema (i.e., tanto de la versión original como la corregida), para facilitar el análisis de los fallos. Como se puede observar, la estrategia es diferente a la presentada por la investigación descrita en el párrafo anterior, puesto que se enfoca más en la recuperación de información a partir del historial de evolución del código. Es decir, se implementó la idea de (1) identificar confirmaciones de correcciones de regresión a partir del historial de evolución del código, (2) migrar la prueba y sus dependencias de código a lo largo del historial y (3) minimizar la sobrecarga de compilación durante la búsqueda de regresión. Teniendo este objetivo en mente se creó la herramienta computacional "RegMiner" la cual recopiló 1,035 regresiones de 147 proyectos en ocho semanas y mostró, además, una precisión y recuperación aceptables sobre conjuntos de datos generados a partir de las pruebas de regresión.

[4] De acuerdo con Ma y Wang (2018), un algoritmo de murciélago representa un algoritmo novedoso de optimización inteligente biónico para simular el comportamiento de búsqueda de alimento y el principio de ecolocalización de los murciélagos.

Finalmente, Minhas et al., (2023) propusieron una estrategia basada en la utilización de listas de comprobación para proporcionar un mecanismo de seguimiento sobre las actividades primordiales de las pruebas de regresión, así como también estructurar el proceso que se lleva a cabo durante la planificación y ejecución de dichas pruebas. Con dicha investigación fue posible identificar actividades que los profesionales consideran importantes al planificar, realizar y analizar las pruebas de regresión. Así, los investigadores diseñaron listas de comprobación basadas en tales actividades con el fin de ayudar a los *testers* a tomar decisiones informadas durante la realización de las pruebas. Finalmente, se aplicaron entrevistas para obtener la percepción de especialistas sobre las listas de comprobación propuestas y se obtuvo que el 80% consideró que éstas cubren los aspectos esenciales para la realización y gestión efectiva de las pruebas de regresión.

Como se puede observar, las propuestas presentadas, si bien son novedosas, están alejadas de la realidad que afrontan las pequeñas organizaciones desarrolladoras de software, puesto que todas estas requieren que los programadores y *testers* posean conocimientos amplios sobre técnicas, medidas y algoritmos propios de las pruebas de regresión. Por lo contrario, en el contexto de estas organizaciones, son demasiados los retos a los que los *testers* se enfrentan durante la implementación y ejecución de las pruebas de regresión, y una razón primordial es la ausencia de procesos establecidos que sean eficientes. De hecho, durante la investigación que se realizó para la redacción del presente documento, no fue posible encontrar una propuesta enfocada a las pequeñas organizaciones desarrolladoras de software. No obstante, algo que puede dar buenos resultados es la adaptación de prácticas adecuadas que disminuyan los esfuerzos y el tiempo que les toma a los *testers* de estas organizaciones llevar a cabo las pruebas de regresión. Esta idea coincide con los argumentos de Parsons et al. (2014), quienes consideraron que el uso de enfoques mixtos puede facilitar la identificación de factores influyentes en cuanto a la adopción y aplicación de las pruebas de regresión dentro de una organización desarrolladora de software de pequeño tamaño. Sin embargo, es importante mencionar también, que estos mismos investigadores enfatizan en el hecho de que uno de los factores claves importantes para que esta labor funcione, es considerar la madurez que coexiste dentro de la organización, lo que conlleva a definir prácticas eficaces en cuanto a las pruebas de regresión así como la coherencia que existe entre estas prácticas y la gestión tanto de la información (e.g., conocimiento) como del cambio en los procesos ya definidos en la organización (Khatibsyarbini et al., 2018).

Considerando todo lo anteriormente mencionado, en esta tesis se definió, implementó y evaluó una estrategia de gestión que pretende mejorar la efectividad de las pruebas de regresión en el software que se desarrolla dentro de una organización de pequeño tamaño.

1.4. Delimitaciones y limitaciones de la tesis

Esta propuesta de investigación estuvo delimitada por los siguientes aspectos:

- Dada la carencia de estudios enfocados al contexto de las pequeñas organizaciones desarrolladoras de software, la estrategia de gestión propuesta en esta tesis es el resultado del análisis y comparación de investigaciones realizadas en el contexto de las empresas grandes, por lo que se requirió de la adaptación de prácticas que demostraron resultados favorables.
- La estrategia de gestión se enfoca únicamente a las pruebas de regresión que se realizan para mejorar el producto dentro de un ciclo de vida del desarrollo de software.
- La estrategia de gestión y la herramienta computacional que se implementó solamente es útil en el contexto de las organizaciones desarrolladoras de software de pequeño tamaño.

Además, las principales limitaciones que se presentaron durante la realización de la investigación fueron las siguientes:

- La evaluación empírica que se realizó sobre la herramienta computacional diseñada dependió de la cantidad de información que las organizaciones participantes tenían registrada en su repositorio de información sobre las pruebas de regresión.
- A pesar de que se creó una propuesta que se ajusta a las características de las organizaciones pequeñas, es posible que los resultados de la evaluación empírica se hayan visto influenciados por los conocimientos previos del personal con relación al proceso de pruebas de regresión o su motivación, considerando que se presentó cierto rechazo por el cambio en su cultura laboral.

1.5. Hipótesis

La hipótesis de investigación que condujo el desarrollo de esta tesis se enuncia de la siguiente manera:

Hi: *"La implementación de una estrategia de gestión de las pruebas de regresión, en una pequeña organización desarrolladora de software, permitirá mejorar su efectividad".*

1.6. Objetivos del trabajo

Considerando lo anterior, este proyecto de tesis se desarrolló en el marco del siguiente objetivo general y los correspondientes objetivos específicos que se establecen a continuación:

1.6.1. Objetivo general

"Diseñar, implementar y evaluar una estrategia de gestión de las pruebas de regresión con el fin de mejorar su efectividad en el contexto de una pequeña organización de desarrollo de software".

1.6.2. Objetivos específicos

Para lograr este objetivo general, fue necesario alcanzar los siguientes objetivos específicos:

1. Se realizó una investigación bibliográfica sobre las estrategias de gestión del proceso de pruebas de regresión en el software, que son utilizadas actualmente por las organizaciones maduras.
2. A partir de los hallazgos de la investigación, se diseñó una estrategia de gestión que involucra las políticas, procedimientos, técnicas, y medidas que mejor se adaptan al contexto de una pequeña organización desarrolladora de software.
3. Se implementó la estrategia de gestión propuesta en una herramienta computacional que facilitó la definición/modificación del proceso de pruebas de regresión dentro de la organización.
4. Finalmente, se evaluó la efectividad de la estrategia considerando las medidas definidas para tal fin (objetivo específico 2). Para esto se definió un caso de estudio que facilitó la obtención y análisis de datos cualitativos y cuantitativos.

1.7. Metodología de trabajo

A pesar de que las pequeñas organizaciones desarrolladoras de software representan a la inmensa mayoría de los proveedores de productos y/o servicios de software en México, y prácticamente en todo el mundo, la definición y ejecución de procesos en éstas son habitualmente todo un desafío. De acuerdo con Tuape et al. (2021), estas compañías se caracterizan por utilizar procesos de baja calidad que no están definidos ni institucionalizados para todos los proyectos o, más aún, no están fundamentados en sus propias necesidades. Lo que afecta gravemente el desarrollo eficiente de todo proyecto de software que se pretenda emprender en estas organizaciones, dificultando así sus esfuerzos por lograr una madurez que derive, como principal resultado, en una mejora sustancial de la calidad en sus procesos y productos. Aunado a esto, los investigadores identificaron que la adopción y adaptación de herramientas tecnológicas para soportar la ejecución de los procesos puede ayudar a estas organizaciones en la definición de una forma productiva de trabajo. Sin embargo, también es importante considerar dos premisas: (1) la tecnología debe agilizar la ejecución del proceso en el contexto organizacional, lo que implica un entendimiento profundo de las características de la propia organización, del proceso, y de las capacidades de las herramientas en sí, y (2) la tecnología debe ayudar a que estas organizaciones alcance, eventualmente, la madurez.

La investigación de Yao et al., (2020), por ejemplo, demostró que la capacidad de las pequeñas organizaciones desarrolladoras de software para crear e innovar productos se ve afectada principalmente por las herramientas tecnológicas que utilizan, las cuales no se adaptan a los entornos en donde se operan, afectando así, entre tantas cosas, el intercambio de conocimientos. Aspecto importante en las pequeñas organizaciones, ya que tiene un impacto positivo en las prácticas de la IS siempre y cuando se dé tanto en el conocimiento tácito (i.e., el que proviene de la estructura organizativa, de las políticas y directrices, del liderazgo de los mandos medios) como para el explícito (i.e., el que se comparte entre los operativos a través de los proyectos).

En este sentido, un estudio más reciente de Tuape et al., (2022) identificó qué factores pueden afectar la definición y/o adopción de procesos eficientes de IS a partir de entrevistas a profesionales de diversas pequeñas organizaciones desarrolladoras de software de cuatro países. Con el análisis de las respuestas surgieron cinco características no técnicas que complementan la falta de experiencia y el soporte tecnológico (i.e., riesgo, ventaja competitiva, resiliencia, capacidad innovadora y capacidad de gestión). Los autores aportaron dos contribuciones más que bien podrían ayudar a que las pequeñas organizaciones le den más formalidad a su práctica habitual de la IS: (1) diseñar un marco teórico para describir a los procesos de software (e.g., *testing*) de tal manera que la organización disponga de los conocimientos fundamentales para el ejercicio de la práctica y (2) una directriz que combine el marco teórico con una herramienta tecnológica que facilite la adaptación, adopción e institucionalización[5] de los procesos.

Considerando lo anterior, el desarrollo de esta tesis siguió las fases mostradas en la Figura 1.1, las cuales se describen en las siguientes secciones del documento.

[5] De acuerdo con Martínez-Ruiz et al. (2009), la institucionalización de los procesos de software es un paso importante que deben realizar las organizaciones en estricto apego a políticas y procedimientos establecidos, si desean mejorar dichos procesos. Además, la institucionalizado se da a través del arraigo gradual de los procesos, considerando el compromiso y la coherencia en su realización (Chrissis et al., 2011).

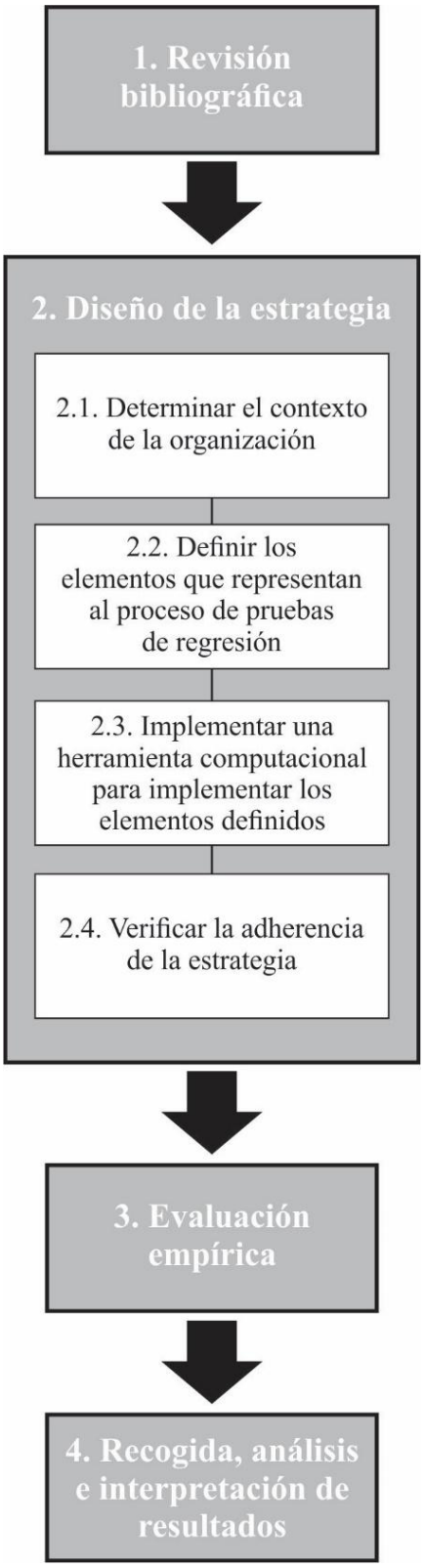


Figura 1.1. Fases para el desarrollo de una solución alternativa.

1.7.1. Revisión bibliográfica

La realización de pruebas sobre el software requiere de la identificación de una serie de elementos que, al integrarse, consolidan lo que se conoce como una estrategia. Michael Porter (1997), conocido mundialmente por sus teorías económicas y estrategias de negocios, considera que una estrategia es el resultado de crear una posición única y valiosa en el mercado a través de la definición de un conjunto formado por diversas actividades. De esta manera, se argumenta que, si dichas actividades fueran efectivas para generar todo tipo de producto en cualquier industria, se podrían satisfacer las necesidades de los clientes y las empresas podrían desempeñar fácilmente su trabajo y su eficacia operativa determinaría una mejora en su desempeño frente a la competencia. Por lo tanto, se considera que la esencia de una estrategia es elegir qué actividades realizar y cuáles no para establecer una ventaja competitiva en el mercado. En el contexto del desarrollo de software, la investigación de Wierunga y Daneva (2015) coincidió en gran medida con esta definición de Michael Porter puesto que se argumentó que una estrategia es la combinación de decisiones que se toman tempranamente para crear una lista de qué hacer y qué no hacer para realizar eficientemente un proceso o producto de software.

Sin embargo, como se ha mencionado a lo largo de este documento, el proceso para llevar a cabo las pruebas de regresión en las pequeñas organizaciones de desarrollo de software fracasa debido principalmente a dos razones: (1) la falta de una definición adecuada del proceso y, por ende, la carencia de institucionalización y (2) el desinterés del personal por controlar la calidad en el software. Esto conlleva al desarrollo de investigaciones que generen guías o recomendaciones formales que permitan que estas organizaciones eliminen la mala gestión del proceso para la realización de las pruebas de regresión con el propósito de evitar consecuencias graves, como el cierre o la quiebra de la organización (Shikta et al., 2021). Pero ¿qué elementos deben incluirse en una estrategia de gestión de las pruebas de regresión que se enfoque a las pequeñas organizaciones?

Con el objetivo de responder esta pregunta, fue necesario, como primera fase de esta tesis, realizar una investigación documental que sirviera como punto de partida para el diseño y desarrollo de una estrategia de gestión que se ajustara a las características de tales organizaciones. Por lo tanto, se siguieron las recomendaciones y guía puntual de Esquirol-Caussa et al., (2017) y Ocaña-Fernández y Fuster-Guillén (2021) para utilizar la revisión bibliográfica como metodología principal de investigación para analizar los conceptos, enfoques teóricos, estudios y antecedentes de las estrategias de gestión de las pruebas de regresión en la industria de software con el fin de identificar cuáles de estos elementos podían adecuarse a las características de las pequeñas organizaciones.

1.7.2. Diseño e implementación de la estrategia

Una vez que se identificaron los elementos que podían conformar la estrategia de gestión enfocada a pequeñas organizaciones desarrolladoras de software, se inició la segunda fase de la tesis. Dicha fase es la parte medular de esta tesis y, por ende, la que consumió la mayor parte del tiempo. En este sentido, se consideró realizar las siguientes cuatro actividades:

1. Determinar el contexto de las organizaciones. Se analizó el contexto de las pequeñas organizaciones desarrolladoras de software para comprender su entorno, las condiciones, el método, y los artefactos que se utilizan para realizar las pruebas de regresión. De esta manera, los elementos identificados en la primera fase podrían ajustarse a un entorno específico para aplicarse de forma asertiva en el desarrollo de software. En este sentido, se siguieron las recomendaciones de Vos et al., (2012) y Espinosa-Curiel et al., (2013) para conducir la evaluación de un proceso de software, es decir, de la realización de pruebas de regresión.

2. Definir los elementos que representarían al proceso de pruebas de regresión. De acuerdo con Marin Díaz et al., (2020) y García et al., (2023) una estrategia de gestión de las pruebas en el software puede conformarse con las buenas prácticas de modelos que hayan demostrado ser eficientes a lo largo del tiempo, estándares y normas acatados a nivel internacional, e incluso con la experiencia obtenida por los académicos/investigadores y especialistas de la industria. Aunado a lo anterior, estas estrategias de gestión pueden integrar la descripción de los objetos de prueba, los casos de prueba, enfoques de pruebas, políticas, procedimientos, responsabilidades, tipos de defectos y fallos que le permitan a una pequeña organización el entendimiento del proceso. Tomando en cuenta lo anterior, se consideró el diseño de una estructura que permitiera la definición de una estrategia que fusione los elementos identificados en la actividad anterior con el fin de diseñar políticas, roles, mecanismos y procedimientos ajustados a las características de la pequeña organización desarrolladora de software (véase Figura 1.2). Además, se diseñó e implementó una herramienta computacional que facilita la comunicación del equipo, la ejecución y medición del proceso, así como la institucionalización de éste.

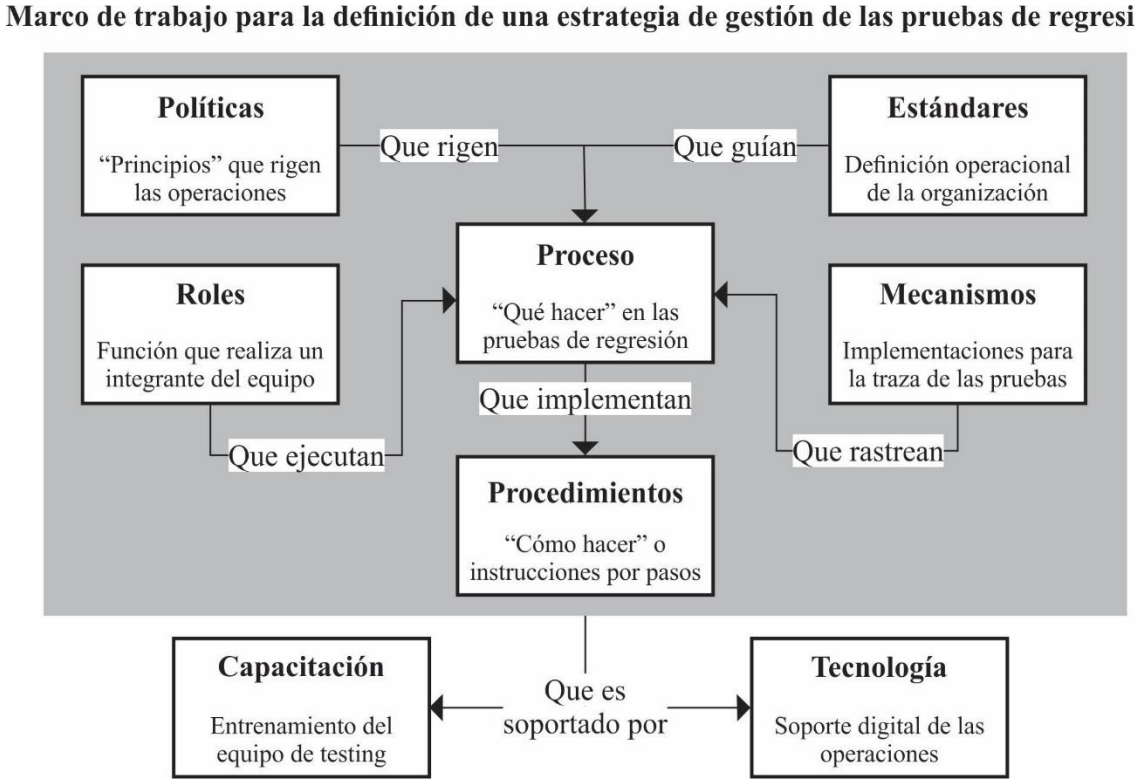


Figura 1.2. Propuesta tentativa de estructura para una estrategia de gestión de las pruebas de regresión.

3. Implementar la estrategia de gestión en una herramienta computacional. Como se mencionó en la actividad anterior, se desarrolló una herramienta computacional que permite la implementación de la estrategia de gestión con el fin de que, tanto los *testers* como los líderes de proyectos de la pequeña organización, puedan planificar y controlar el desarrollo de las pruebas de regresión del software. De esta manera, la herramienta computacional permite la adopción de una estrategia de gestión para cada proyecto que se emprenda, priorizando la medición y la obtención de lecciones aprendidas (véase Figura 1.3). Las políticas,

procedimientos, roles y mecanismos descritos por la estrategia contribuyen con la definición de un proceso estándar para las pruebas de regresión adecuado a la organización desarrolladora de software. En este sentido, las políticas establecen cómo es que los roles involucrados en las pruebas deben realizar los procedimientos marcados por el proceso. Tanto el uso de los procedimientos como la experiencia acumulada por las personas que desempeñen tales roles permitirán la generación de lecciones aprendidas que serán divulgadas como conocimiento en la organización, buscando así promover la institucionalización del proceso. Aunado a esto, a través de un tablero de control, se puede implementar esta estrategia en diferentes proyectos para asegurar tanto la calidad del proceso, como del producto generado en las pruebas de regresión.

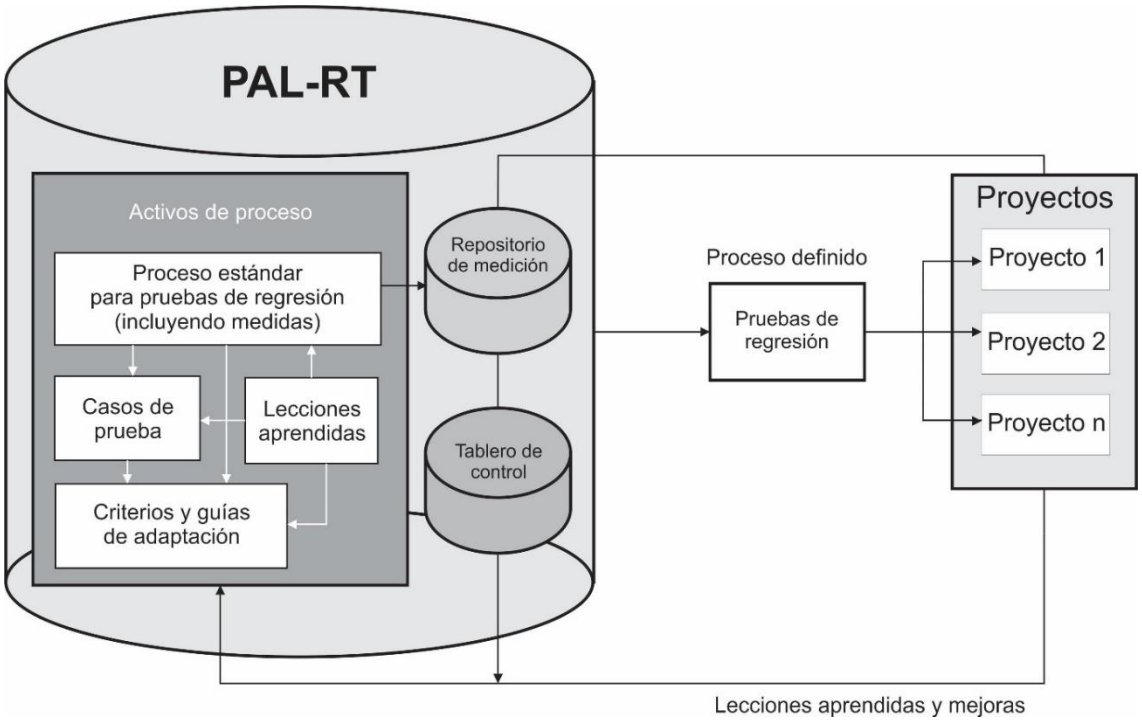


Figura 1.3. Integración de la estrategia definida con la herramienta computacional.

4. Verificar la adherencia de la estrategia. Finalmente, el proceso establecido, con la estrategia propuesta y la herramienta computacional creada, fue evaluado para determinar su adherencia en pequeñas organizaciones desarrolladoras de software. Regularmente dicha adherencia se evalúa considerando procesos de verificación y/o auditoría que involucran a los investigadores y personal de la organización. En este sentido, se consideraron las recomendaciones de Russell (2006) para verificar que el proceso que se ha definido para las organizaciones se ajusta a la documentación que se haya generado para asegurar su uso correcto. Es decir, se verificó que la interacción descrita en la Figura 1.3 se alineara con la cultura de trabajo de las organizaciones.

1.7.3. Evaluación empírica

Con el objetivo de demostrar la efectividad de la solución propuesta, fue necesario realizar una evaluación empírica bajo un enfoque mixto (i.e., tanto cualitativa como cuantitativa) que permitiera la obtención de datos subjetivos y objetivos para validar la hipótesis establecida en la tesis. En este sentido, en el contexto de la IS, los investigadores pueden evaluar sus propuestas de forma objetiva y científica mediante la realización de estudios empíricos. De acuerdo con Wohlin et al., (2003), existen tres maneras para realizar este tipo de evaluaciones: aplicación de encuestas, realización de casos de estudio, y conducción de experimentos formales. En este escenario, existen tres estrategias para desarrollar un caso de estudio:

1. Comparar los resultados obtenidos de una nueva propuesta contra una la línea base.
2. Desarrollar dos proyectos en paralelo ("proyectos gemelos"), eligiendo uno de ellos como línea base.
3. Aplicar la nueva propuesta a los componentes seleccionados y comparar con los resultados de los componentes a los que no se les había aplicado la nueva propuesta.

Por lo tanto, se planteó el diseño y realización de un caso de estudio relacionado con el desarrollo de dos proyectos en paralelo dentro de una pequeña organización desarrolladora de software. Esto implica que se realizó la gestión de las pruebas de regresión en un proyecto ficticio considerando dos equipos: uno de control (que usó el método tradicional para realizar dicha gestión) y uno experimental (que usó la herramienta computacional resultante de esta investigación). De esta manera se recogió información para confirmar si la estrategia propuesta mejora o no la efectividad de las pruebas de regresión. Es decir, la investigación tuvo, evidentemente, un enfoque cuantitativo que se caracteriza por el análisis numérico y la medición de datos y que, como consecuencia, genera resultados comparables entre diferentes variables o grupos (Zúñiga et al., 2023). Sin embargo, cabe aclarar, que, dado que este enfoque podría no capturar la complejidad y riqueza de la experiencia humana, también se planteó la introducción de un enfoque cualitativo. Por lo tanto, se realizó, al mismo tiempo, un análisis cualitativo de la información recogida con la evaluación empírica mediante el diseño y aplicación de cuestionarios a ambos equipos (i.e., control y experimental) con el fin de determinar sus percepciones sobre la estrategia diseñada.

1.7.4. Recogida, análisis e interpretación de los resultados

Finalmente, todos los datos recogidos durante la evaluación empírica fueron analizados estadísticamente para identificar diferencias notables entre los grupos de control y experimental.

1.8. Estructura de la tesis

En el Capítulo 2 se presentan el marco teórico y estado del arte de la investigación actual relacionada con el tópico abordado en esta tesis. Además, se presenta el diseño y resultados de un estudio realizado sobre pequeñas organizaciones desarrolladoras de software con el fin de obtener una visión general de las pruebas de regresión que sirva como base para el diseño de una propuesta acorde a sus necesidades.

En el Capítulo 3 se describe la estrategia propuesta para realizar y gestionar las pruebas de regresión en el contexto de las pequeñas organizaciones desarrolladoras de software. En este sentido,

se presenta la implementación de una herramienta computacional para poner en práctica dicha estrategia en, al menos, una pequeña organización desarrolladora de software.

En el Capítulo 4 se presentan el diseño, ejecución y resultados de un caso de estudio para evaluar empíricamente la utilidad de la estrategia propuesta en la organización participante.

En el Capítulo 5 se presenta una discusión sobre la investigación realizada y se exponen las principales conclusiones derivadas de la tesis.

La sección de Bibliografía resume las referencias bibliográficas base y adicionales utilizadas para el desarrollo de la tesis.

Por último, el Anexo A lista todos los acrónimos usados en el documento.

1.9. Publicaciones generadas

A continuación se proporciona la información del producto científico generado como resultado de la presente tesis de grado:

Autoras y autores:	Edgar López Cruz, Iván Antonio García Pacheco, Carla Leninca Pacheco Agüero
Título del artículo:	Resultados iniciales de una evaluación cualitativa sobre la adherencia de una estrategia para mejorar la realización y gestión de las pruebas de regresión en pequeñas organizaciones desarrolladoras de software.
Nombre de la revista:	Revista de Investigación en Tecnologías de la Información
DOI:	https://doi.org/10.36825/RITI.13.31.002
Volumen:	13
Número:	31
Páginas:	4-17
Mes y año:	Octubre 2025

2. Marco teórico y estado del arte

El Capítulo 2 de esta tesis proporciona una base conceptual que permite contextualizar el problema identificado y sus características, con el objetivo de presentar de manera clara una vía alternativa de solución. En este sentido, se abordan a continuación diversos conceptos e investigaciones relevantes que son fundamentales para caracterizar y fundamentar la realización de la presente tesis.

2.1. El ciclo de vida y las pruebas de software

Tanto la literatura especializada como la comunidad de investigadores en el área consideran que el software está representado por un conjunto de instrucciones y datos que permiten que un dispositivo electrónico realice actividades específicas. En este sentido, se ha asumido también la existencia de dos posibles categorías fundamentales para el software que es creado en la actualidad: el software de sistema y el software de aplicación. De acuerdo con Heller (2020), el software de sistema proporciona funcionalidades básicas que le permiten a los usuarios interactuar con el hardware de la computadora (e.g., sistemas operativos, controladores de dispositivos, utilerías del sistema), mientras que el software de aplicación es visto como todo aquel programa informático que es diseñado para resolver problemas específicos de los usuarios, de tal manera que se logre mejorar su productividad (e.g., procesadores de texto, hojas de cálculo, traductores, gestores de proyectos) o le facilite la realización de tareas de diseño gráfico, contabilidad, entre otras.

Por otro lado, Sommerville (2021) argumenta que el software puede ser propietario o de código abierto. El software propietario, como su nombre lo da a entender, está restringido y es gestionado por una sola entidad y aquellos usuarios que deseen adquirirlo tendrán que hacerlo por medio de la compra o renta de una licencia; mientras que el software libre (o también llamando *open source*) permite a sus usuarios el acceder al código fuente para modificarlo según los términos que se establezcan. No obstante, Pressman y Maxim (2019) afirman que, independientemente de que tipo de software se trate, su desarrollo implica un proceso que abarca desde su concepción y diseño hasta su implementación y mantenimiento. Así pues, Palomo y Gil (2020) aseveran que todo software que sea creado como un producto comercial debería desarrollarse de acuerdo con las normas, modelos, y/o directrices que son definidas y utilizadas en el sector industrial, además de comercializarse siguiendo las prácticas adecuadas del *marketing*. Aunado a esto, se considera que los documentos generados durante el desarrollo del software deben ser revisados y aprobados por el cliente, mientras que las personas que desempeñan el rol de líder dentro de las organizaciones serán los responsables de supervisar la organización, el desarrollo y la monitorización del proyecto. Por lo tanto, dicha documentación juega un papel importante en el proyecto, puesto que será útil para que, entre otras cosas, las partes involucradas validen su constante compromiso en cada paso que se dé. Considerando esta lógica,

como cualquier otro producto, el software debe crearse siguiendo un conjunto sistemático y ordenado de pasos que aseguren su terminación, es decir, lo que se denomina un "ciclo de vida". En el contexto de la IS, el ciclo de vida del software se define como "*el proceso de dividir el trabajo de desarrollo del software en distintas fases para mejorar su diseño, la gestión del producto, y la gestión de proyecto*" (Sommerville, 2021). De hecho, Farley (2021) amplió dicha definición al afirmar que dicho proceso debe "*producir software de la mayor calidad, y con el menor costo y tiempo posible*". Las fases del ciclo de vida del software suelen definirse de la siguiente manera:

1. Análisis. En esta fase se determinan y analizan los deseos y necesidades expuestas por el o los futuros usuarios del software, de tal manera que éste tenga la capacidad de satisfacer tales expectativas. Como resultado se producirá una especificación precisa del software que habrá de desarrollarse (i.e., requisitos).
2. Diseño. En esta fase se elabora una arquitectura que considere cada uno de los componentes identificados para que el software satisfaga la especificación obtenida en la fase anterior. Es decir, se establece la estrategia que se seguirá para el desarrollo de una solución técnica (i.e. el software) que incluye a la documentación arquitectónica, la estructura de los componentes y la distribución de los elementos en particular.
3. Codificación. En esta fase se materializa aquello que hará funcional al software. Es decir, cada uno de los componentes definidos en la fase anterior será creado haciendo uso de herramientas y recursos como el lenguaje de programación, las bases de datos, cualquier otro sistema útil que proporcione información, y demás. Por lo tanto, en esta fase se realiza la construcción de los componentes necesarios para asegurar que el software funcione de forma correcta y, además, satisfaga todos los deseos y necesidades de los usuarios que fueron especificados en forma de requisitos. Teniendo este objetivo en mente, se lleva a cabo la realización de pruebas a nivel de componente para verificar que todo está funcionando correctamente.
4. Integración y pruebas. Dado que se asume que el software es desarrollado por más de una persona, en esta fase se realiza la unificación de los componentes codificados en la etapa anterior con el propósito de integrar un solo producto de software. Por otro lado, se realizan también pruebas exhaustivas para asegurar que los componentes, ya integrados, funcionan correctamente durante su inspección.
5. Mantenimiento. La comúnmente considerada como última fase del ciclo de vida indica que se ha concluido una iteración del proyecto y se procede a iniciar la siguiente o bien que se han concluido todas las iteraciones definidas. Además, es importante mencionar que se asume que en esta fase se debe realizar la corrección de los errores que no fueron detectados en las fases de codificación y de integración y pruebas o bien, para introducir nuevas mejoras en el código. En este contexto, la evolución y el crecimiento son fundamentales para introducir nuevos requisitos que mejoren considerablemente el funcionamiento del software

Cabe resaltar que, como se explicó en el Capítulo 1 esta tesis, se pretende definir una estrategia que facilite a las pequeñas organizaciones desarrolladoras de software la gestión de las pruebas de regresión sobre los productos de software que estén precisamente bajo desarrollo. En este sentido, Aniche (2022) establece que las pruebas que se realizan sobre el software se pueden clasificar de acuerdo con tres parámetros importantes: el tipo de pruebas, los métodos, y los niveles. El *tipo*, por ejemplo, hace referencia a la forma en que las pruebas pueden llevarse a cabo, es decir manualmente, donde el proceso es realizado por el *tester* de manera directa sin la intervención de una herramienta que lo automatice, o bien de forma automatizada, para la cual las pruebas ya fueron configuradas en

un sistema externo y por lo tanto pueden ejecutarse sin ninguna intervención. Por otro lado, con relación a los *métodos*, Garousi et al., (2020a) argumentan que tradicionalmente se emplean pruebas de caja negra, las cuales se basan en entradas y salidas para evaluar la funcionalidad del software sin necesidad de conocer su estructura interna (i.e., el *tester* no cuenta con acceso al código fuente o al diseño interno del software); las pruebas de caja blanca, que se enfocan en examinar la estructura interna del código fuente (i.e., el *tester* debe comprender el código y su lógica para diseñar casos de prueba[6] que cubran distintos caminos y condiciones durante su ejecución), y, finalmente, las pruebas de caja gris, que no son más que la fusión de los dos métodos anteriores, por lo que el *tester* requiere poseer cierto conocimiento sobre la estructura interna del software con el fin de diseñar casos de prueba más efectivos (Pintelas et al., 2020; Wang et al., 2022). Considerando el parámetro final, el *nivel* sobre el que se pueden realizar las pruebas, regularmente se habla de pruebas funcionales y no funcionales. Baumgartner et al., (2021) consideran que las pruebas funcionales garantizan que el software satisfaga tanto los requisitos funcionales especificados en la fase de análisis, como el comportamiento que se espera del software. Entre las más conocidas se encuentran las pruebas unitarias, pruebas de componentes, pruebas de humo, pruebas de integración, pruebas de regresión, pruebas de cordura y pruebas de aceptación. Por otra parte, Jarzębowicz y Weichbroth (2021) definen a las pruebas no funcionales como aquellas que evalúan los aspectos que no están directamente relacionados con las funcionalidades específicas del software, sino más bien con características como la usabilidad, el rendimiento, la seguridad, la estabilidad, entre otras. Cabe mencionar que regularmente las pruebas no funcionales son asociadas con aspectos de calidad del software; sin embargo, como se demostrará más adelante en esta tesis, las pruebas funcionales otorgan calidad objetiva al software, mientras que la calidad de las pruebas no funcionales es meramente subjetiva.

En conclusión, dado que como resultado de esta tesis se pretende generar una estrategia que mejore la gestión de las pruebas durante el ciclo de vida del desarrollo de software, se incluirá a las pruebas funcionales de caja negra y/o blanca, ya sea que se realicen manualmente o de forma automatizada, que se ejecuten dentro de una pequeña organización desarrolladora de software. En este sentido, con el fin de generar un resultado más específico, la tesis abordará la definición, análisis y gestión de las pruebas de regresión, las cuales serán descritas en la siguiente sección del documento.

2.2. Las pruebas de regresión en el desarrollo de software

En la actualidad, las organizaciones desarrolladoras de software están obligadas a crear productos de calidad que satisfagan las necesidades y expectativas cada vez más exigentes de los clientes. En este sentido, Srikanth y Murugan (2020) argumentan que las pruebas de software permiten la identificación y corrección de problemas que pudieran estar afectando negativamente la integridad, seguridad y calidad de cualquier producto o servicio de software y que, además, vulneren las necesidades y deseos establecidos por el o los usuarios. Sin embargo, a pesar de los avances que se han logrado hasta el día de hoy en el área específica de las pruebas que se hacen sobre el software, los métodos formales y las técnicas empleadas no son suficientes para lograr examinar por completo el software desarrollado, antes de proceder a su liberación.

Por lo tanto, las *pruebas de regresión* buscan garantizar que los cambios realizados sobre el código fuente, durante su desarrollo, no introduzcan nuevos defectos que conduzcan, o no, a fallos en

[6] En el contexto de la Ingeniería de Software, un "caso de prueba" es un conjunto de acciones que verifican si una aplicación o un sistema de software, o una característica o comportamiento de éstos, es aceptable o no. Itkonen y Mäntylä (2014) consideran que dado que las pruebas de software son comúnmente consideradas como un proceso que ejecuta tales casos de prueba, éstos deben ser cuidadosamente prediseñados a través de técnicas específicas.

el software. De acuerdo con diferentes especialistas, las pruebas de regresión han sido útiles para alcanzar un objetivo específico: "*probar el software después de que se le hayan realizado cambios para asegurar que no se hayan introducido errores o alterado su funcionamiento previo*". Los siguientes investigadores dan fe de este dicho. Por ejemplo, Zarrad (2015) argumentó que estas pruebas podrían verse como un proceso que permitía revelar el impacto causado en los diferentes módulos que conforman a un software después de haberle realizado modificaciones. De manera coincidente, Rosero et al., (2016) consideraron que las pruebas de regresión permitían verificar que los cambios realizados o las nuevas características añadidas al software no afecten negativamente su funcionalidad y que se comporte igual o más eficiente que en las versiones anteriores. La investigación realizada por Kazmi et al., (2017) argumentó también que las pruebas de regresión podían ayudar a que los *testers* se aseguraran que los errores detectados en una versión anterior de código no se repitieran en una versión posterior, puesto que, al realizarlas, se genera un informe sobre la solución de las fallas que incrementa y mejora el conjunto de pruebas al agregar el nuevo caso de prueba que resolvió la falla en el código original. Sin embargo, resaltaron también el hecho de que dicho caso de prueba podía volverse obsoleto si no se le realizaban modificaciones periódicas, ya que su rango de cobertura corresponderá con los patrones específicos detectados para cada fallo en particular. Años más tarde, Ali et al., (2019) coincidieron con las aseveraciones anteriores puesto que consideraron que las pruebas de regresión representaban prácticamente un recurso crucial para garantizar que los cambios realizados en el software no hayan introducido nuevos defectos.

Las pruebas de regresión son pues, una herramienta esencial para mantener la calidad y estabilidad del software a lo largo del tiempo, puesto que poseen características esenciales como la repetibilidad, automatización, granularidad, y cobertura que potencializan las habilidades de un *tester* para reducir la tasa de errores en los productos/servicios de software que son modificados y/o mejorados continuamente. Considerando lo anterior, la *repetibilidad* se refiere a la posibilidad de que los *testers* obtengan resultados que puedan ser replicados o reproducidos de manera consistente (Felderer y Fourneret, 2015). La repetibilidad es deseable en las pruebas de regresión, ya que proporciona confianza en la precisión de los resultados. Si una prueba de regresión produce resultados inconsistentes o variables en el tiempo, puede indicar problemas con la estabilidad, la calidad o la aplicación de la prueba misma. La *automatización* en las pruebas de regresión, por otro lado, implica el uso de herramientas y *scripts* para ejecutar pruebas repetitivas de manera eficiente (Engström et al., 2010). La *granularidad* especifica el nivel de detalle con el que se realizan las pruebas de regresión, es decir, en términos generales, se trata de cuán específicas y minuciosas deben ser para asegurar que los cambios en el código no introduzcan errores o regresiones en el software existente (Catal y Mishra, 2013). Finalmente, la *cobertura* proporciona una medida objetiva de la efectividad de las pruebas de regresión en términos de qué parte del código se ha probado y cuál no. Sin embargo, es importante también considerar que el tener una alta cobertura de pruebas no garantiza que no haya errores en el software, pero sí aumenta la confianza en su calidad (Qiu et al., 2014).

Sin embargo, a pesar de que estas pruebas han tomado una mayor relevancia en la industria especializada en el desarrollo de software, en la actualidad son consideradas todo un reto puesto que no es fácil gestionarlas de tal manera que se obtengan beneficios directos con su realización. Así pues, las pruebas de regresión a menudo involucran diferentes tipos de evaluaciones que aseguran que el software no disminuya su calidad a causa de su modificación y/o mejora, entre las más comunes se encuentran las siguientes:

- Pruebas de regresión correctiva. Estas pruebas garantizan que cualquier modificación o cambio que se realice sobre el código fuente no introduzca nuevos defectos ni que los ya existentes vuelvan a manifestarse. Por lo tanto, se requiere la ejecución de un conjunto de

pruebas[7] sobre el software con el fin de asegurar que todas las funcionalidades respondan adecuadamente, evitando así la presencia de nuevos problemas. Considerando lo anterior, este tipo de pruebas evita que los defectos encontrados en el código se transfieran al producto que se entregará el cliente (Braz et al., 2022).

- Pruebas de regresión *retest-all*. Estas pruebas suelen realizarse sobre un código fuente que ya había sido previamente probado con el fin de garantizar que éste no presente una regresión al realizar cambios o actualizaciones. Es decir, este tipo de pruebas no se enfoca en partes específicas que se hayan visto afectadas por los cambios, si no que realiza la verificación de todo el software garantizando así que ninguna de las funcionalidades existentes se vea afectada negativamente. En resumen, es necesario realizar reiteradamente la ejecución de todas las pruebas que ya se hayan realizado anticipadamente para confirmar que ningún cambio o actualización al código fuente haya provocado una regresión en las funcionalidades ya implementadas (Elsner et al., 2022).
- Pruebas de regresión selectiva. Estas pruebas facilitan la selección de casos de prueba en función del tipo de cambio que se haya realizado en el software. Por lo tanto, es común que se utilicen para identificar problemas e interacciones que pudieron ser causados por alguna modificación en sus componentes. De acuerdo con los criterios establecidos, la selección puede darse por la realización de cambios en el código fuente, el tipo de cambio, el riesgo que se tiene al realizar algún cambio, entre otros. Es decir, este tipo de pruebas también evita que se presenten problemas a causa de cambios que se hayan realizado en el código, pero a diferencia de las pruebas de regresión anteriores, utilizando solamente aquellos casos de prueba que hayan sido previamente escogidos (Chen, 2021).
- Pruebas de regresión progresiva. Estas pruebas realizan la estimación escalonada de la calidad de los productos/servicios de software basándose en la supervisión gradual y continua del código fuente existente versus los cambios en el software ya desarrollado. Por lo tanto, se asegura que el software sea compatible tanto con nuevas funcionalidades agregadas, como con la corrección de los errores entre otras las modificaciones, y otros cambios. En resumen, se involucra a un proceso constante de verificación que inicia con las funciones más relevantes y continúa a lo largo del tiempo para tratar cada uno de los componentes del software desarrollado (Kumar, 2023).
- Pruebas de regresión completa. Estas pruebas se realizan cuando se requieren actualizaciones del software a gran escala a causa de los constantes cambios en las necesidades de los clientes. Dado que es posible que se presenten modificaciones importantes en el software, es prioridad que este tipo de pruebas garanticen que las funcionalidades persistan inalterables, aún después de haber hecho cualquier modificación (Bizyukov et al., 2020).
- Pruebas de regresión parcial. Estas pruebas determinan que los cambios realizados sobre un componente o funcionalidad específica no hayan afectado de forma imprevista al software. Comúnmente este tipo de pruebas se utilizan cuando el software no se ha acondicionado del todo de forma conjunta, sino que solamente se han realizado pequeños cambios. Por lo tanto,

[7] De acuerdo con Fraser y Arcuri (2012), los "conjuntos de pruebas" son agrupaciones lógicas de casos de pruebas cuyo propósito es definir qué pruebas deben ejecutarse juntas en un plazo específico de tiempo para demostrar que un software cumple o no con un conjunto específico de comportamientos. Además, un conjunto de pruebas regularmente incluye instrucciones u objetivos detallados para los casos de prueba e información sobre la configuración del software que se utilizará durante la ejecución de las pruebas.

las pruebas de regresión parcial proporcionan la certeza de que el software sigue funcionando como se prevé, reduciendo el riesgo de propiciar nuevos errores (Govil y Sharma., 2021).

- Pruebas de regresión unitaria. Estas pruebas se enfocan en evaluar unidades individuales de código fuente para determinar si son aptas para su uso o no. En este sentido, una unidad puede ser una sola línea de código, función, clase, método, procedimiento, módulo u objeto. Por lo tanto, este tipo de pruebas se aprovechará mejor en evaluaciones pequeñas, ya que brindan una vista granular de cómo funciona el código y se asegurarán de que éste siga funcionando de forma aislada (Ji et al., 2022).

Considerando la clasificación anterior, se podrá entender que las pruebas de regresión representan un aspecto fundamental para garantizar la calidad del software puesto que se vislumbran como el punto de partida para el crecimiento evolutivo, sin grietas, de un producto de software. Por esta razón es que, en el desarrollo de sistemas complejos de gran tamaño, las pruebas de regresión suelen ser repetitivas aumentando así la precisión sobre su alcance. Sin embargo, para llevar a cabo esta tarea con éxito se requiere de recursos y esfuerzo adicional para ejecutar los casos de prueba una o las veces que sean necesarias. Por lo tanto, Alí et al., (2019) argumentaron que se han propuesto distintas técnicas que buscan disminuir la cantidad de casos de prueba, los recursos y/o el esfuerzo requerido para realizar las pruebas de regresión. Entre las más reconocidas en la actualidad se encuentran las siguientes:

- Técnica de selección de los casos de prueba: La técnica procura disminuir el tiempo utilizado en la ejecución de las pruebas seleccionando solamente un grupo de casos de prueba del conjunto previamente definido. En este sentido, la investigación de Singhal et al., (2021) identificó una categorización que permite elegir el método más apropiado de selección. La categoría de *inclusividad*, por ejemplo, es utilizada para medir el grado de selección de la técnica mediante la evaluación de los resultados obtenidos al realizar pruebas, tanto en el software modificado como en el original, lo cual permite identificar los fallos provocados por los cambios. Por otro lado, la categoría de *precisión* mide la capacidad de la técnica para eludir las pruebas que provocarán que el software modificado produzca el mismo resultado que el software original. La categoría de *eficacia* mide el costo computacional y la viabilidad de la técnica de selección. Por último, la categoría de *generalidad* mide la capacidad de la técnica para manejar los cambios en el código durante la ejecución de pruebas reales. La técnica de selección de los casos de prueba se puede realizar considerando las siguientes aproximaciones:
    - *Enfoque basado en los grafos, las relaciones de dependencia y el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés)*. Es común que este enfoque se utilice para mejorar la representación del flujo de control de los datos y programas cuando se pretende realizar pruebas de regresión sobre el software embebido. Se argumenta que el enfoque es ideal para programas pequeños y que puede combinarse con algunas heurísticas para maximizar la selección de los casos de prueba.
    - *Enfoque basado en los modelos*. Este enfoque utiliza la abstracción de los modelos aplicados al desarrollo de software para representar a los programas, tomando en cuenta la priorización de las tareas, los temporizadores, la administración de las prioridades y el manejo de errores. Sin embargo, en la literatura solamente existe evidencia de que este enfoque ha sido utilizado para programas de tamaño pequeño.
    - *Enfoque basado en las heurísticas*. El enfoque se orienta a múltiples objetivos que mezclan las heurísticas con la programación lineal, considerando los criterios de cobertura y tiempo. A pesar de que existe poca evidencia de su uso en entornos

industriales reales, la mezcla de excepciones, diagramas de transición y diagramas de estado permite la representación del flujo de control de datos, el manejo de excepciones y la relación que existe entre los objetos. Por otra parte, se argumenta que este enfoque puede combinarse con diferentes algoritmos de partición y algoritmos genéticos para optimizar los resultados.

- *Enfoque basado en los datos históricos*. Este enfoque se anticipa a los fallos considerando los resultados y la información de proyectos anteriores, tales como el registro de los cambios, los casos de prueba, las ejecuciones de las pruebas, entre otros datos. Por lo tanto, la granularidad y la precisión del enfoque es de suma importancia, pero dependerá directamente de los datos históricos registrados.
- *Enfoque basado en la detección de fallos*. El enfoque identifica los fallos mediante el análisis del comportamiento de los programas en cada ciclo de desarrollo, y registrando los cambios que modifiquen el comportamiento esperado de los modelos.
- *Enfoque basado en las modificaciones*. Este enfoque se encuentra directamente relacionado con el TDD, comúnmente a nivel de método, por lo que muestra mayor eficiencia dada su capacidad de reducción del esfuerzo, favorable para la regresión.

- Técnica de priorización de los casos de prueba: La técnica busca garantizar que los recursos sean asignados de manera efectiva durante las pruebas de regresión y que, además, los aspectos críticos del software sean probados exhaustivamente. Mukherjee y Patnaik (2021) argumentan que la priorización de los casos de prueba puede basarse en los riesgos, las dependencias o los requisitos críticos. La priorización basada en los riesgos establece que los casos de prueba sean priorizados de acuerdo con el nivel de riesgo asociado con las funcionalidades o características del software. Por lo tanto, aquellas áreas que tienen un mayor impacto o son más propensas a errores críticos, son las que se probarán primero. Por otro lado, la priorización basada en las dependencias considera que los casos de prueba sean priorizados tomando en cuenta las dependencias que se tiene con otras partes del software. Garantizando así que las funcionalidades críticas que afectan a múltiples partes del software sean probadas primero para evitar impactos negativos en otras. Finalmente, la priorización basada en los requisitos críticos argumenta que los casos de prueba sean priorizados de acuerdo con la importancia de los requisitos del cliente o del negocio. Es decir, los requisitos críticos o fundamentales para cubrir las funcionalidades del software que serán probados primero. Esta técnica de priorización a menudo se realiza bajo las siguientes aproximaciones:
  - *Enfoque basado en detección de fallos*. El enfoque hace uso de algoritmos de ordenamiento sobre los casos de prueba, tomando como parámetro principal el instante en el tiempo en el que el fallo es detectado. Sin embargo, se argumenta que el enfoque aún no está listo para usarse de forma general, ya que los resultados obtenidos al probarlo sobre programas de tamaño mediano no han sido suficientemente certeros.
  - *Enfoque probabilístico basado en los datos históricos*. Este enfoque plantea el uso del ordenamiento de los casos de prueba tomando en consideración la probabilidad para la identificación de los fallos. Aunado a esto, el enfoque utiliza los datos registrados en los repositorios, los cuales son la base para llevar a cabo un enfoque probabilístico.
  - *Enfoque basado en los modelos*. El enfoque se basa en la priorización de los casos de prueba para programas escritos bajo el paradigma de la Programación Orientada a Objetos (OOP, por sus siglas en inglés). De esta manera, se utiliza la comparación de código para identificar el momento en el que haya ocurrido algún cambio con el fin de etiquetarlo y realizar una revisión que dictamine el/los elementos afectados por dicho

cambio. Finalmente, sin tomar en consideración la información dinámica del programa, se identifican los casos de prueba con la relevancia más alta.

- *Enfoque basado en el análisis dinámico y el lenguaje natural*. Este enfoque hace uso del análisis dinámico, el código y el análisis del lenguaje natural sobre el conjunto de pruebas con el fin de priorizarlos. La literatura muestra evidencia de resultados favorables para el TDD; sin embargo, tales resultados se limitan únicamente a proyectos de software de tamaño mediano.
- E*nfoque basado en los grafos y el análisis de dependencia*. Este enfoque considera el valor de las dependencias entre las distintas clases de un modelo en OOP para construir esquemas de priorización. El enfoque ha sido utilizado en entornos industriales y ha demostrado que reduce el esfuerzo durante la ejecución de las pruebas; sin embargo, también se ha sugerido la disminución del tamaño de los *stubs* (i.e., tipo de clases que emulan el comportamiento de otras) para el caso de las clases independientes, puesto que el enfoque no es útil cuando se pretende realizar una comprobación conjunta.
- *Enfoque basado en las heurísticas*. Este enfoque hace uso de un Procedimiento Adaptativo y Codicioso de Búsqueda (GRASP, por sus siglas en inglés) y métricas de cobertura para comparar en eficiencia cada caso de prueba a través de algoritmos genéticos y codiciosos.

- Técnica de reducción de casos de prueba: La técnica se enfoca en la identificación de casos de prueba que sean redundantes con el fin de eliminarlos para minimizar así el tamaño del conjunto de pruebas. En este sentido, a esto se le conoce como el problema de conjunto de impacto mínimo que especifica que cada requisito de prueba sea satisfecho por un solo caso de prueba. Sin embargo, en la práctica, el probar un requisito puede requerir más de un caso de prueba, lo que provoca ajustes en la granularidad de los casos de prueba. Yoo y Harnan, (2012) argumentaron que diversas heurísticas podrían abordar la complejidad del problema de reducir el conjunto de pruebas. Estas heurísticas suelen incluir enfoques basados en, por ejemplo, la programación lineal, los algoritmos voraces y las técnicas de análisis de conceptos formales. La técnica de reducción a menudo se realiza considerando una única aproximación:
  - *Enfoque basado en heurísticas*. Antes de la aplicación del enfoque, los casos de prueba son refinados para evitar casos redundantes, ambiguos y obsoletos, obteniendo así un conjunto minimizado de casos de prueba para las pruebas de regresión. Sin embargo, solamente existe evidencia de su uso experimental en entornos industriales, donde se emplearon algunas técnicas sobre programas de tamaño pequeño, tales como los algoritmos genéticos de clasificación no dominados y la evolución diferencial híbrida basada en un multiobjetivo cuántico.
- Técnica de optimización de los casos de prueba: Esta técnica se enfoca en la identificación y ejecución de un conjunto de pruebas que maximice la cobertura de código y los posibles escenarios de uso, minimizando con esto el tiempo y los recursos requeridos. De acuerdo con Do (2016), la optimización efectiva de los casos de prueba se basa en el seguimiento de pasos fundamentales, como: realizar un análisis exhaustivo de los requisitos del software y de las funcionalidades que se desean probar; diseñar un conjunto de casos de prueba que cubra todas las situaciones relevantes, prestando atención en la priorización de escenarios críticos y aquellos propensos a errores o fallos; realizar pruebas de ejecución y análisis de resultados para validar la efectividad del conjunto de pruebas diseñado. Este proceso puede implicar ajustes y refinamientos adicionales en la selección de pruebas para mejorar su cobertura y

eficiencia. En la actualidad, la técnica de optimización considera las siguientes aproximaciones:

- *Enfoque basado en lógica difusa*. Este enfoque tiene dos vertientes, la primera es la identificación de incertidumbre en los casos de prueba, mientras que la segunda es el uso de técnicas de Inteligencia Artificial (IA) sobre los casos de prueba. Por lo tanto, el enfoque utiliza criterios de ajuste y adaptabilidad para la clasificación de los casos de prueba. Aunado a esto, la lógica difusa puede complementarse con la programación multiobjetivo para ponderar y optimizar factores con la finalidad de obtener datos relevantes al momento de la evaluación.
- *Enfoque basado en heurísticas*. El enfoque persigue la optimización multiobjetivo mediante la incorporación de algoritmos genéticos, la ley de Pareto optimizada, así como también los algoritmos codiciosos. Sin embargo, es importante también resaltar la falta de experimentación en el entorno industrial y su orientación a programas de tamaño pequeño.

- Uso de la IA para mejorar las pruebas de regresión: De acuerdo con Maulud y Abdulazeez (2020), la IA permite mejorar la *eficiencia* de las pruebas de regresión puesto que facilita la automatización de gran parte del proceso, lo que ahorra tiempo y recursos en comparación con las pruebas manuales. Aunado a esto, se mejora la *cobertura* de las pruebas, ya que se pueden analizar grandes conjuntos de datos y escenarios de prueba para identificar patrones y casos que podrían pasarse por alto con las pruebas manuales. De manera similar, la IA facilita la *detección temprana de errores*, ya que la identificación de anomalías y problemas en el software durante las pruebas de regresión permite a los equipos abordar las anomalías de manera proactiva antes de que se conviertan en problemas graves al llegar a la producción. Por otro lado, el uso de la IA permite agregar la capacidad de *adaptabilidad* a las pruebas de regresión, puesto que los algoritmos pueden aprender con el tiempo para mejorar continuamente la eficacia de las pruebas de regresión a medida que se afrontan nuevos desafíos y cambios en el software. Finalmente, la IA promueve la *optimización de recursos* puesto que permite la priorización y ejecución de las pruebas de regresión de manera inteligente, así los equipos pueden optimizar el uso de recursos y centrarse en las áreas del software que son más susceptibles a cambios y errores. Considerando lo anterior, existen diversos estudios que evidencian los beneficios de combinar la IA con las técnicas descritas anteriormente:
  - *Selección de los casos de prueba*. De acuerdo con Pan et al. (2022), el uso de la IA en las técnicas de selección de casos de prueba es cada vez más común en el campo de la Ingeniería de Software, puesto que ofrece una serie de ventajas significativas en comparación con los métodos tradicionales. Es decir, la IA puede identificar patrones de comportamiento inesperados o errores durante la ejecución de los casos de prueba. Así, estos datos pueden realimentar el proceso de selección de casos de prueba para mejorar la cobertura y la efectividad de las pruebas. Por ejemplo, la investigación de Sutar et al., (2020) introdujo una propuesta que utiliza el Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) para encontrar casos de prueba de alto potencial del conjunto original de pruebas y seleccionar un caso de prueba en función de su intención por coincidir con los defectos. De acuerdo con los resultados reportados, se logró reducir el ciclo de regresión y se mejoró la productividad de los *testers*.
  - *Priorización de los casos de prueba*. La investigación de Sharif et al. (2021), por ejemplo, presentó *DeepOrder* como un modelo basado en el aprendizaje profundo para abordar la priorización de los casos de prueba en el contexto de la integración continua.

Es decir, se reconoce la importancia de seleccionar casos de prueba efectivos para detectar fallas tempranas en el ciclo de desarrollo. A través del aprendizaje computacional sobre la regresión, *DeepOrder* clasifica los casos de prueba considerando diversos factores, como la duración y el estado de ejecución. Los resultados de la experimentación mostraron que este enfoque supera tanto a las prácticas de la industria como a los enfoques modernos en términos de efectividad y eficiencia en la detección de fallas.

- *Reducción de los casos de prueba*. La investigación de Saifan et al. (2016) argumentó que los resultados de la técnica de reducción de los casos de prueba podrían mejorarse considerablemente al utilizar algoritmos de minería de datos, árboles de decisión, regresión logística, y/o *clustering* para analizar los datos y descubrir patrones y relaciones entre las características y los resultados de las pruebas. En este sentido, la generación de modelos predictivos basados en los patrones identificados puede ayudar en la predicción de resultados de pruebas futuras o a identificar áreas de riesgo en el software. La utilización de modelos predictivos y los patrones identificados, por otro lado, posibilitan la reducción en el número de casos de prueba necesarios para cubrir eficazmente los escenarios de prueba, puesto se pueden identificar casos de prueba redundantes o agrupar casos de prueba similares en categorías representativas. (Maulud y Abdulazeez, 2020).
- *Optimización de los casos de prueba*. Finalmente, la investigación de Anwar et al., (2019) destacó los resultados de usar un Sistema Neuro Difuso (NFS, por sus siglas en inglés) en la optimización de los casos de prueba. Dicho sistema inteligente se compuso de Redes Neuronales Artificiales (ANN, por sus siglas en inglés) y un sistema difuso de inferencia. Aunado a esto, se utilizó también un Sistema de Inferencia Neuro Difuso Adaptativo (ANFIS, por sus siglas en inglés) debido a su precisión, ya que ha demostrado producir menos errores (e.g., error cuadrático medio) en comparación con otros sistemas neuro difusos.

Como se observa, la realización de las pruebas de regresión no es algo trivial, puesto que se pretende que las organizaciones desarrolladoras de software las puedan llevar a cabo con la eficacia suficiente para obtener beneficios directos. De hecho, la investigación de Engström et al., (2012) lo había resaltado ya hace más de una década al afirmar que "*a medida que la complejidad en el software crece, el costo que implica realizar las pruebas de regresión también aumentará*". Es decir, no se trata de hacer o no las pruebas de regresión, sino de gestionarlas correctamente para asegurar tanto un beneficio para el proceso y producto de software, como las ganancias monetarias de la organización. De lo contrario, se generarán cuellos de botella importantes o bien simplemente las pruebas de regresión serán consideradas irrelevantes con el paso del tiempo. Así, es innegable la especial importancia que tiene la realización de pruebas eficaces de regresión para todas las organizaciones que buscan la actualización constante de sus servicios/productos de software. Es por ello por lo que continuamente surgen nuevos enfoques para el desarrollo de software que basan su eficiencia en la frecuencia, efectividad, y gestión de las pruebas (e.g., los métodos de desarrollo ágil en conjunto con la integración continua para actualizar los servicios/productos de software). Bajo esta premisa, las pruebas de regresión se realizan considerando el principio de validar y verificar que los cambios hechos en el código no impacten de forma negativa las funcionalidades que, antes de tales modificaciones, se ejecutaban correctamente.

A pesar de que la realización de las pruebas de regresión sobre el software a menudo representa un incremento sustancial en los costos de desarrollo, este tipo de pruebas se considera una parte fundamental e ineludible durante su desarrollo, puesto que garantizan la realización de cambios correctos y no negativos en el producto de software. De acuerdo con Wang et al. (2023), por ejemplo, una estrategia que ha funcionado correctamente durante poco más de 25 años en el contexto de la industria de software es la selección de la(s) técnica(s) de prueba(s) de regresión que mejor se acople(n) al contexto y al producto, considerando un conjunto de pruebas previamente definido. Es decir, una especie de gestión y reutilización del conocimiento acumulado. De esta manera, al considerar una o varias técnicas de prueba que hayan dado buenos resultados, se podrían validar eficientemente las modificaciones que se hacen sobre el software. Es decir, la idea es crear un algoritmo o herramienta automatizada que sea capaz de seleccionar de un conjunto de pruebas previamente establecido, el cual ya ha sido madurado a lo largo del tiempo, aquellas pruebas que deben ejecutarse sobre el código que ha sido modificado. Además, es importante mencionar que dicho algoritmo debe considerar todas las condiciones definidas por las características del proyecto y del producto, el entorno de desarrollo, el tamaño del equipo de trabajo, el costo del producto y demás, de tal manera que la(s) técnica(s) seleccionada(s) exhiban la mayor cantidad de fallos agregados con las nuevas modificaciones. Es importante mencionar que, además de la complejidad que implica la creación de un algoritmo de este tipo, una organización desarrolladora de software debe asegurarse de que el personal esté registrando, documentando, y utilizando toda la información generada en cada proyecto, de lo contrario no se obtendrá beneficio alguno. Zhong et al. (2019), por lo tanto, consideraban que el objetivo de dicha selección es reducir sustancialmente la repetición de casos de prueba puesto que se ejecutarían solamente aquellos que tengan efecto en el código modificado, siempre y cuando se tuviera plenamente conocimiento sobre el producto y el historial de cambios. De hecho, empresas importantes como *Google* han adoptado esta estrategia obteniendo resultados importantes como la reducción del 34% del tiempo destinado a las pruebas.

Con la finalidad de aumentar la eficacia al seleccionar la(s) técnica(s) más adecuada(s) para realizar las pruebas de regresión, la literatura también sugiere el uso de técnicas de priorización. Singhal et al. (2021), por ejemplo, argumentan que se debe dar mayor prioridad a técnicas que permitan incrementar la tasa de detección de fallos, es decir, la rapidez con la que los fallos son detectados a lo largo del proceso de pruebas. De esta manera, sería posible obtener oportunamente información valiosa del software que está siendo sometido a prueba, con el objetivo de que los *testers* corrijan mucho antes los fallos más importantes. Por lo tanto, el uso de técnicas de priorización en las pruebas de regresión facilita la reutilización de la información, ya que el probar después de haber realizado algún cambio en el código facilita el aprovechamiento de la información obtenida en la ejecución anterior y, como consecuencia, se podría determinar el orden de los casos de prueba a ejecutar.

Por otro lado, el uso de los conjuntos de prueba (o bien, colecciones de casos de prueba) se ha mantenido a lo largo del tiempo como una herramienta poderosa para detectar fallos a medida que el software evoluciona. Sin embargo, una de las principales desventajas de este enfoque se relaciona con los altos costos derivados de ejecutar grandes conjuntos de prueba puesto que, regularmente, se requiere de herramientas poderosas de automatización. En este sentido, la investigación de Greca et al., (2023) identifica tres enfoques que ayudan a los especialistas a incrementar el valor de un conjunto de pruebas: minimización, selección, y priorización. La minimización procura la eliminación de los casos de prueba repetidos para decrementar el tamaño del conjunto de pruebas a ejecutar. Por otra parte, la selección de los casos de prueba identifica aquellos casos de prueba con mayor relevancia para agregarlos al conjunto que será ejecutado sobre los cambios recientes en el código. Finalmente, la priorización de los casos de prueba tiende a ordenar los conjuntos de prueba con la finalidad de

maximizar la identificación anticipada de fallos. Desafortunadamente, Myers et al., (2023) consideran que el avance en la especialización o madurez total de la IS no ha sido del todo constante, incluso para las fases del ciclo de vida que involucran tanto a las pruebas funcionales como a las no funcionales. Considerando precisamente a este tipo de pruebas, es verdad que los principales avances están ampliamente fundamentados en la literatura, lo que ha permitido la introducción de, por ejemplo, nuevos enfoques como la priorización de los conjuntos de pruebas de regresión para su posterior selección; sin embargo, su uso dentro de la industria desarrolladora de software es poco común, ya que las técnicas y métodos propuestos son poco o nada escalables o, en el peor de los casos, las organizaciones no poseen la suficiente madurez (e.g., recursos humanos, recursos económicos, infraestructura) como para aplicarlos correctamente. A continuación, se presenta un análisis al respecto.

2.3. Las pruebas de regresión en la industria de software

Como se podría entender, por la naturaleza de su significado, las pruebas de regresión implican que los *testers* se vean en la necesidad de repetir las pruebas que fueron realizadas previamente, cuando el software se estaba desarrollando, con el fin de garantizar que no se hayan introducido nuevos errores o retrocesos con las posteriores modificaciones que se realizan sobre el código. Sin embargo, esto no es del todo cierto puesto que, a medida que se avanza en el desarrollo del software, el conjunto de pruebas tiende a aumentar considerablemente en tamaño, lo que hace prácticamente inasequible el volver a ejecutar todos los casos de prueba, bajo un enfoque *retest-all*, ya que los costos excesivos que se generen pueden consumir una parte significativa del presupuesto asignado al proyecto (Labuschagne, 2017). En este sentido, la investigación de Minhas (2022) argumentó que, a lo largo del tiempo, tres de las técnicas descritas en la sección anterior han sido las más exploradas en la industria del desarrollo de software al considerar la realización de las pruebas de regresión. La primera, por mayor importancia, es la *Reducción de los Casos de Prueba* (TSR, por sus siglas en inglés), la cual, como se mencionó anteriormente, se enfoca en la identificación y eliminación de todos aquellos casos de prueba redundantes y arcaicos que se encuentran en el conjunto de pruebas. La segunda técnica se basa en la *Selección de los Casos de Prueba* (TCS, por sus siglas en inglés) para elegir un subconjunto del conjunto original que es utilizado para verificar aquellas partes del código que hayan sufrido un cambio. La tercera técnica es la *Priorización de los Casos de Prueba* (TCP, por sus siglas en inglés), que promueve la identificación ordenada del conjunto de pruebas que maximiza los atributos deseables (e.g., detección temprana de fallos). En este sentido, Coviello et al., (2018) consideran que la literatura existente evidencia que la implementación de estas tres técnicas resulta rentable para las organizaciones la mayoría de las veces. Pero ¿estas técnicas son realmente utilizadas en la industrial real de software? A continuación, se intentará dar una respuesta a esta interrogante.

De acuerdo con Shi et al. (2019), la investigación sobre la definición y ejecución de actividades que permitan establecer un proceso eficiente para las pruebas de regresión ha estado vigente desde los años 80, pero, desafortunadamente, al parecer no se ha logrado causar el impacto deseado en la industria de software puesto que aún existen deficiencias al ponerlo en práctica. Por ejemplo, la investigación de Engström et al. (2010) reveló la existencia de discrepancias entre la investigación y la práctica relacionadas con las pruebas de regresión, y, ocho años más tarde, la investigación realizada por Khan et al. (2018) demostró que este panorama se había agravado aún más puesto que menos del 4% de las publicaciones se relacionaban con experiencias de la industria. ¿Qué ocurrió entre 2010 y 2018 que no permitió que el panorama inicial mejorara? La revisión sistemática de literatura conducida por Greca et al., (2023) aporta evidencia interesante que contribuye a dar una posible respuesta. En dicha investigación se examinaron alrededor de 1,320 estudios candidatos relacionados

con enfoques para realizar las pruebas de regresión que fueran relevantes y aplicables a la industria, y solamente se seleccionaron 79 como estudios primarios puesto que demostraron cierta relevancia industrial. Aunado a lo anterior, se aplicaron encuestas a 23 especialistas de la industria de Brasil, Italia, Finlandia, Hungría, Portugal y Suecia con el fin de comprender mejor el impacto de estos enfoques en la práctica dentro del entorno industrial. Tanto los resultados de la revisión de literatura como la realimentación proporcionada por los especialistas permitieron la obtención de información que proporcionó una comprensión más profunda de cómo estos enfoques abordaban las preocupaciones de aplicabilidad y si habían tenido impacto, o no, en la industria. En este sentido, la información recogida proporcionó evidencia, en primera instancia, de que los métodos empleados por las técnicas para realizar las pruebas de regresión en la industria se han enfocado en cumplir principalmente dos propósitos diferentes: considerar la fuente de la información utilizada como entrada para una técnica (véase Figura 2.1) y considerar el algoritmo utilizado para abordar el problema a resolver (véase Figura 2.2).

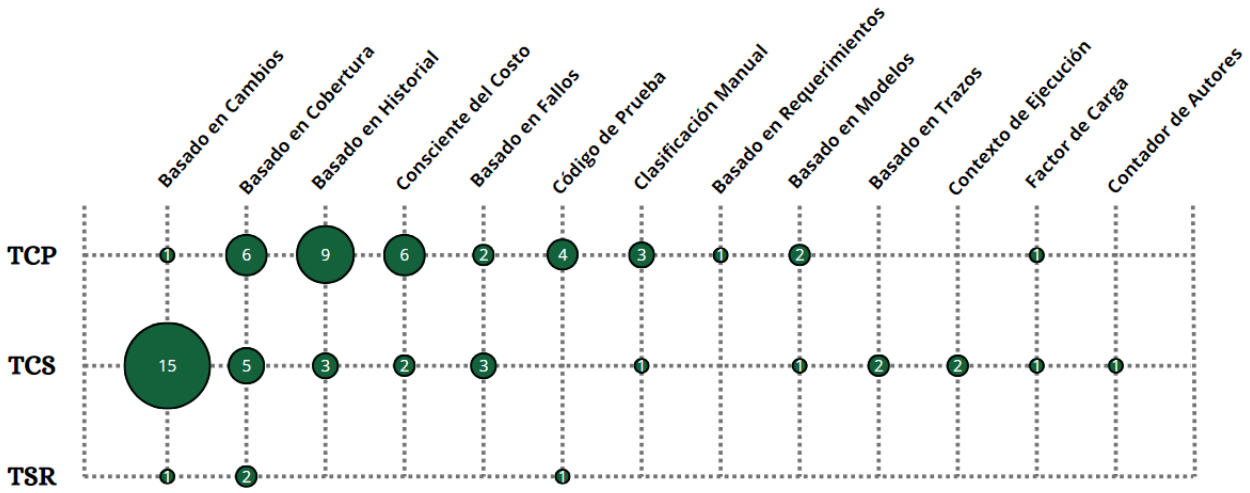


Figura 2.1. Distribución de enfoques de la información. Nota: Traducida de Greca et al., (2023)

Figura 2.2. Distribución de enfoques algorítmicos. Nota: Traducida de Greca et al., (2023)

Con relación a los *enfoques en la información* destacan los que se basan en los cambios, en la cobertura, en el historial y los que son conscientes de los costos; mientras que los *enfoques algorítmicos* más populares en la actualidad son el aprendizaje computacional, los que se basan en las búsquedas, los basados en las similitudes y los que se basan en los gráficos. Por otro lado, la investigación también identificó las principales medidas que se emplean en la industria para evaluar, principalmente, la efectividad (i.e., la capacidad de una técnica para cumplir con su tarea) o la eficiencia (i.e., el tiempo y costo asociados con el uso de dicha técnica) de las pruebas de regresión. Las Figuras 2.3 y 2.4 muestran tales medidas agrupadas de acuerdo con su objetivo principal. De acuerdo con la información recogida, se logró identificar a dos medidas adicionales que no pertenecían a ninguna de las categorías definidas: la aplicabilidad/generalidad y la capacidad de diagnóstico.

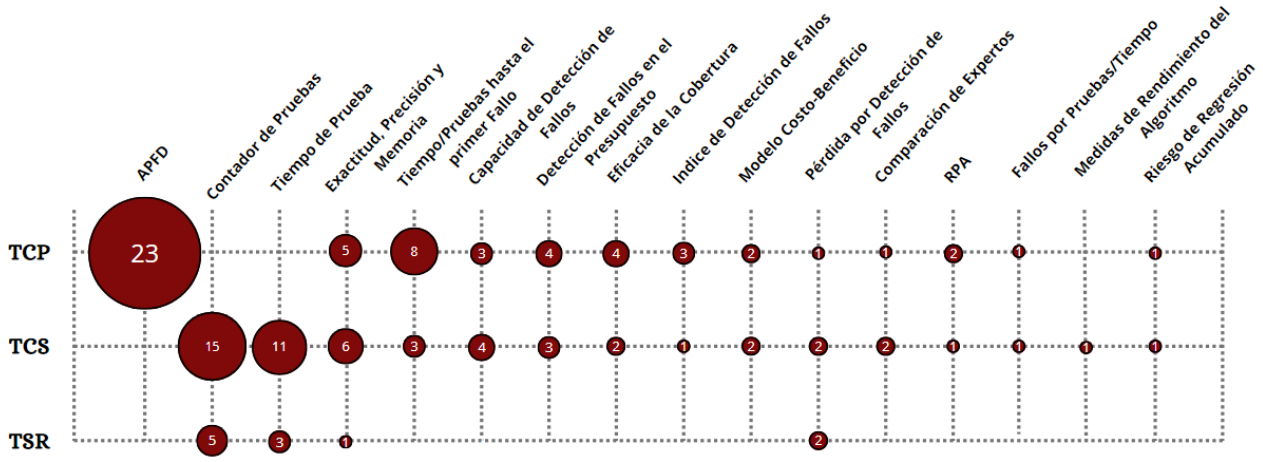


Figura 2.3. Distribución de medidas de efectividad. Nota: Traducida de Greca et al., (2023)

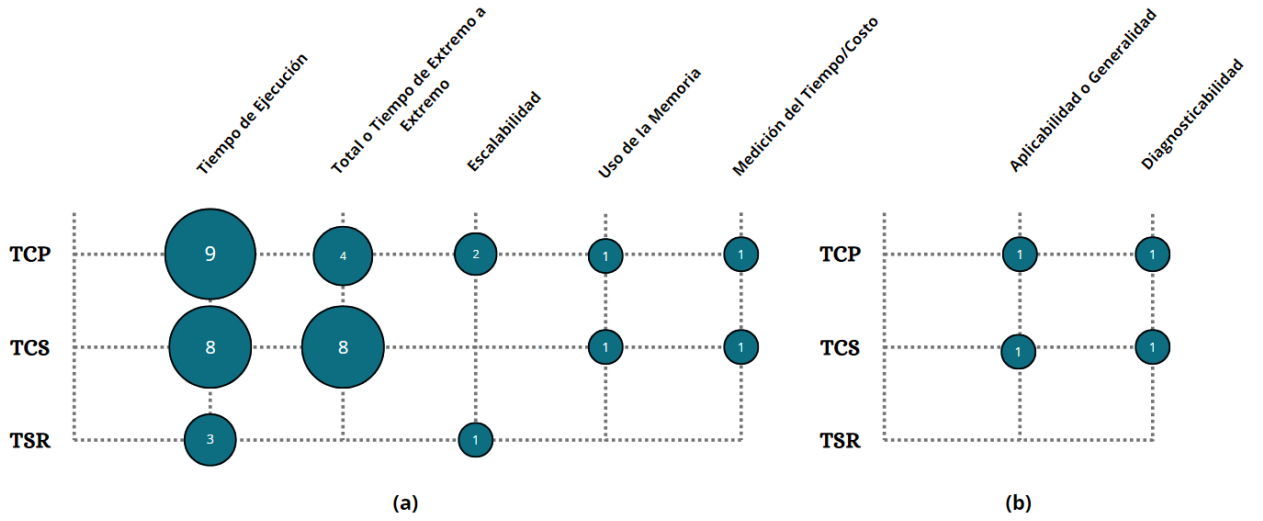


Figura 2.4. Distribución de medidas de (a) eficiencia y (b) otras. Nota: Traducida de Greca et al., (2023)

El *Porcentaje Promedio de Fallos Detectadas* (APFD, por sus siglas en inglés) es utilizada como la medida principal para evaluar los enfoques utilizados para la técnica de TCP (véase Figura 2.3). Para el caso de las técnicas de TCS y TSR, puesto que ambas buscan reducir la cantidad de pruebas respecto al conjunto original, destacan medidas como el *tiempo de prueba*, el *recuento de*

selección y la *pérdida en la detección de fallas*. Cabe resaltar que la medida de *exactitud, precisión y/o recuperación* parece abordar la mayoría de las técnicas que se presentan en la actualidad. Para garantizar la eficiencia se considera ampliamente la medida de *tiempo de ejecución*, además de que aplica para cualquier tipo de técnica. Sin embargo, algunos expertos indicaron que no creían que la selección de las medidas fuera el problema, sino que era más importante resaltar que el tamaño del conjunto de datos utilizado en los experimentos debía ser mayor para tener la posibilidad de ofrecer evidencia significativa en el contexto de la industria. Por lo tanto, en cuanto a la aplicación práctica de las técnicas para realizar las pruebas de regresión en las organizaciones, el 90% de los especialistas de la industria argumentó estar satisfecho con las medidas que suelen seleccionar. Aunado a esto, todos los especialistas consideran que las medidas influyen directamente en la toma de decisiones relacionadas con la elección de socios industriales, que su uso es regularmente percibido como necesario por los *testers* para desarrollar eficientemente su trabajo, y que son fundamentales para la adopción del proceso de pruebas y medir y ejecutar mejoras continuas sobre el mismo.

Por otro lado, durante la realización del estudio surgieron otras observaciones interesantes de los investigadores. Por ejemplo, se reflexionó sobre que, aunque las medidas utilizadas eran relevantes justo en el momento en que se aplicaron las encuestas, era posible que otras medidas pertinentes hubieran sido excluidas dado el tamaño de la muestra que se utilizó como especialistas de la industria (i.e., 23 participantes). Esta observación destacó la importancia de realizar un seguimiento sobre la adopción de los enfoques descritos anteriormente en entornos industriales reales. Curiosamente, también se observó que, si bien algunos estudios primarios describían colaboraciones industriales, los experimentos realizados no incluían herramientas computacionales que facilitaran la definición, ejecución y seguimiento de las pruebas de regresión. En este sentido, la observación principal que se hizo fue sobre la evidente existencia de investigación sobre herramientas computacionales que faciliten la aplicabilidad de las técnicas, pero no sobre el desarrollo de otras que permitan definir y diseminar el conocimiento sobre las pruebas de regresión a través de las organizaciones. Es decir, de los 79 estudios primarios que fueron analizados, solamente cinco no mostraron una motivación clara para realizar actividades de investigación sobre la relevancia y aplicabilidad industrial de las técnicas empleadas en las pruebas de regresión. Además, considerando la experimentación formal, 50 de los estudios proporcionaron evidencia de realizar experimentos en contextos donde se involucra al desarrollo de software a gran escala.

Con esta información se determinó que los especialistas han estado motivados por desarrollar nuevas técnicas, medidas, y/o desarrollos tecnológicos que faciliten la realización de las pruebas de regresión, al mismo tiempo que realizan una adecuada experimentación y evaluación de sus propuestas. Es importante mencionar que 44 de los estudios que proporcionaron información relevante sobre una evaluación real, también proporcionaron evidencia de realizar experimentos con la colaboración directa de un socio estratégico (i.e., una organización real de desarrollo de software), lo que demuestra la importancia de tales colaboraciones. Sin embargo, cabe resaltar que algunos estudios no proporcionaron información suficiente que reflejara la colaboración de socios industriales en los experimentos realizados, por lo que puede existir una amenaza importante sobre esta conclusión. No obstante, sí se destacó la importancia de la experiencia industrial de los investigadores, puesto que los estudios cuyos autores están directamente relacionados con la industria tienden a ser más relevantes en la práctica, ya que están diseñados con aplicaciones específicas en mente y a menudo proporcionan información valiosa sobre el flujo de trabajo de las pruebas en organizaciones desarrolladoras de software de un tamaño considerable. Mientras que los estudios que combinan investigadores industriales y académicos mostraron mejores resultados. En resumen, aunque los investigadores argumentaron haber observado un progreso en varios aspectos relacionados con la aplicación práctica

de las técnicas y medidas, todavía existen desafíos significativos que deben abordarse para mejorar la aplicabilidad de las pruebas de regresión en entornos reales de la industria.

Las investigaciones realizadas por Rosero et al., (2016) y Rosero et al., (2022) proporcionan información que complementa los hallazgos de Greca et al., (2023) para dar un panorama más completo sobre la realización de las pruebas de regresión en la industria de software. En este sentido, se afirma que las estrategias más utilizadas para conducir la implementación de las técnicas para realizar las pruebas de regresión, las cuales fueron descritas anteriormente, se basan en el análisis del código fuente, en los modelos del programa, en la información histórica de los casos de prueba, en los casos de prueba en sí, en heurísticas, y en enfoques de IA. Todas estas estrategias tienen sus pros y contras. Por ejemplo, las *estrategias basadas en modelos* generalmente intentan generar pruebas de regresión para la selección basada en abstracciones y, por ende, su principal desventaja es el detalle de granularidad, que no permite la detección de fallos, especialmente cuando los fallos están demasiado cerca de suceder. La *estrategia basada en información histórica* considera la perspectiva de la prevención o predicción y por eso utiliza la información histórica procedente de registros de defectos, correcciones o incidencias. Sin embargo, su desventaja es que el tratamiento dado a los fallos de versiones anteriores rara vez afecta al desarrollo incremental iterativo. Por otro lado, la *estrategia basada en el análisis del código fuente* resulta ser la más utilizada; pero su desventaja más fuerte es el costo en el que se incurre para realizar el proceso de análisis, puesto que a medida que crecen el código fuente y los casos de prueba también crece el esfuerzo computacional. Cabe mencionar que, de acuerdo con la evidencia recogida en las investigaciones, se observa el auge de las estrategias que incorporan enfoques de IA y minería de datos, principalmente, así como el campo de las heurísticas donde destacan la optimización multiobjetivo de los casos de prueba para la prevención y detección de fallos.

Considerando lo anteriormente expuesto, las herramientas computacionales fueron identificadas como un factor crucial para facilitar la aplicación de las técnicas en la industria. Sin embargo, solamente el 31% de los estudios primarios analizados proporcionó información sobre el uso de éstas, lo que sugiere la necesidad de hacer mayor énfasis en la disponibilidad de herramientas que faciliten tanto la realización de las pruebas de regresión en las organizaciones como la mejora de la replicabilidad de los resultados. El hallazgo más importante se relacionó con las herramientas computacionales que están disponibles en línea y que fueron utilizadas en los estudios para promover su replicabilidad y facilidad de acceso. Así pues, con el fin de facilitar las comparaciones entre las investigaciones y simplificar la experimentación en la industria, los investigadores consideran que es fundamental que se comparta la manera en que la técnica es implementada, ya sea en formato binario o código fuente. A pesar de lo lógico de esta observación, solamente el 15% de los especialistas encuestados proporcionó evidencia, a través de un repositorio de código fuente, de las herramientas que empleaban para automatizar las pruebas de regresión. Desafortunadamente, el acceso a esta información era privado y no fue posible replicar su uso, ni siquiera contar con un pseudocódigo, sin previa autorización de sus desarrolladores. No obstante, de acuerdo con los investigadores, se observó un cambio entre 2016 y 2020, puesto que 14 estudios ya contaban con herramientas que se podían replicar. Entre 2021 y julio de 2022 se encontraron ocho estudios que también ya satisfacían este criterio. La explicación que se atribuye a esto fue que, durante los últimos años, en diferentes conferencias de relevancia internacional sobre la Ingeniería de Software se ha destacado el valor de las investigaciones fácilmente replicables y esto ha provocado que diversos investigadores las conviertan ahora en una prioridad. Esta nueva tendencia ha llevado a que ya existan algunos estudios para los cuales el código fuente esté disponible, aunque se cuente con poca o ninguna documentación o explicación de cómo funciona, o bien que ya se proporcione en otros estudios algunos ejemplos con pasos claros y detallados sobre cómo utilizar el código y replicar los experimentos. De hecho, los

investigadores recibieron por correo electrónico el acceso a cuatro repositorios de códigos fuente, de los cuales solamente dos proporcionaron instrucciones precisas de uso para proyectos de software a gran escala. Estas herramientas están disponibles como *plug-ins* para Eclipse IDE y el sistema de compilación Maven, respectivamente. Los otros dos estudios proporcionaron el código fuente de sus herramientas, principalmente para la replicación de estudios, no necesariamente para su uso en otras organizaciones, lo que podría significar que dichas herramientas probablemente no sean lo suficientemente sólidas como para su uso práctico más allá de simples experimentos. Es verdad que, con mucha frecuencia, las herramientas que son desarrolladas en colaboración con un socio industrial terminen convirtiéndose en software propietario. Esto puede explicar porque los autores de 14 estudios respondieran que el código o la herramienta no se podían compartir, ya que el software resultante era total o parcialmente propietario o de carácter confidencial.

Otro problema que se identificó con la investigación tiene que ver con los lenguajes usados para programar el software que se pretendía probar y que se empleó en los experimentos. La Figura 2.5 muestra que 23 estudios involucraron a software escrito en Java, demostrando así la existencia de un fuerte sesgo hacia este lenguaje. Los investigadores consideran que, en la mayoría de los estudios enfocados en un lenguaje específico, no está claro si el mismo enfoque sería fácil de adaptar y si produciría resultados equivalentes en software que es desarrollado con otros lenguajes de programación. Sin embargo, 12 estudios involucraban a programas escritos en varios lenguajes o bien afirmaron explícitamente que el enfoque propuesto era independiente del lenguaje, lo que aumenta considerablemente su aplicabilidad. Desafortunadamente, los investigadores no pudieron identificar el lenguaje de programación empleado en 21 artículos, lo que crea un desafío sustancial para promover tanto la replicabilidad de los experimentos como de la técnica implementada en éstos.

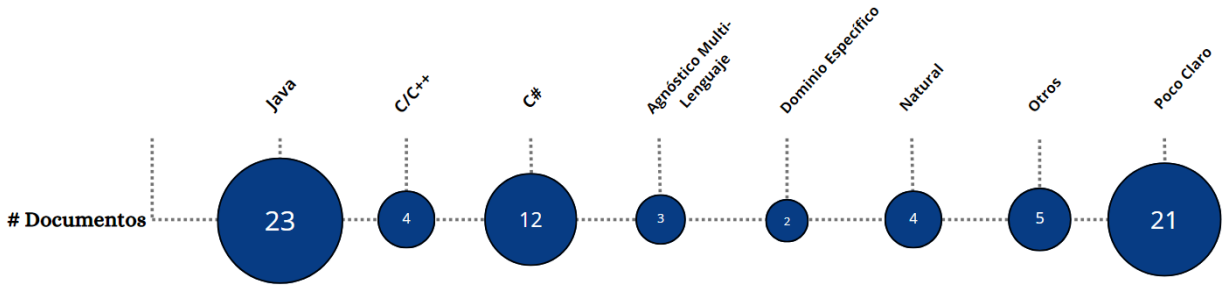


Figura 2.5. Distribución de los lenguajes de programación utilizados. Nota: Traducida de Greca et al., (2023)

En este sentido, la investigación sobre cómo lograr la realización efectiva de las pruebas de regresión en la industria moderna muestra la inquietud generalizada sobre la falta de tecnologías prometedoras desarrolladas en el ámbito académico. Es decir, la disparidad que existe entre las innovadoras propuestas académicas y su implementación efectiva en la industria real de software es la verdadera brecha que sigue retrasando la consolidación de la transferencia tecnológica entre, precisamente, la academia y la industria. Los investigadores y especialistas de la industria consideran que, si bien las preocupaciones sobre la adopción e implementación de propuestas de investigación resultan cruciales para reducir esta brecha, es digno de reconocer que este paso por sí solo no resolverá el desafío de realizar las pruebas de regresión de manera efectiva en todos los tipos y tamaños de organizaciones desarrolladoras de software alrededor del mundo. En términos reales, todas las organizaciones desarrolladoras de software son diferentes y es importante entender que lo que funciona en una organización de tamaño grande no necesariamente debe funcionar igual en otra de tamaño pequeño, y viceversa. Así pues, la misma investigación determinó la existencia de evidencia sobre el grado de adopción de propuestas que fueron desarrolladas en el ámbito académico y que

fueron evaluadas en organizaciones reales. Los hallazgos mostraron al respecto que, de los 79 estudios seleccionados como conjunto primario, 16 abordaron explícitamente la aplicación de propuestas académicas con socios industriales o bien insinuaron que la implementación se encontraba en curso al momento de realizar la publicación de la investigación. Sin embargo, después de una entrevista con los investigadores participantes de cada estudio, se comprobó también que, de este conjunto primario, solamente seis propuestas se seguían utilizando en la industria y cuatro habían dejado de usarse de manera definitiva. Otros ocho investigadores afirmaron que sus enfoques fueron implementados después de que la investigación fuera divulgada en la comunidad científica y argumentaron que el hecho de tener a un especialista de la industria, como coautor del estudio, facilitó la conexión entre la teoría y la aplicación práctica. Desafortunadamente, solamente ocho de los 16 estudios que evidenciaron el uso de propuestas académicas incluyeron comentarios del personal de la organización que utilizó la herramienta desarrollada. Esto indicó que, aunque las herramientas fueron integradas efectivamente al flujo de trabajo de la organización, tanto los beneficios a largo plazo como el nivel de aceptación por parte de los usuarios no fueron documentados. En última instancia, los investigadores también proporcionaron información relevante sobre el uso continuo de las herramientas computacionales y las razones de su desuso. De acuerdo con la información recogida, 12 propuestas nunca fueron implementadas, aunque en algunos casos se discutió la posibilidad de hacerlo en otro momento. Además, siete estudios indicaron que las herramientas computacionales dejaron de usarse después de unos años, lo que resaltó la necesidad de que los investigadores académicos conduzcan un seguimiento continuo para garantizar que los enfoques sean viables a largo plazo, incluso después de su implementación inicial.

De manera similar, en 10 estudios (de los cuales cuatro fueron llevados hasta la etapa de implementación) se proporcionó evidencia de que las herramientas creadas para mejorar la implementación práctica de las pruebas de regresión sí habían sido utilizadas, pero que dejaron de usarse después de un tiempo. En otros tres estudios se indicó la intención de llegar a la implementación, pero su estado actual al momento de publicar la investigación fue desconocido. De acuerdo con los investigadores, esto sugiere que, incluso cuando una técnica sea implementada en una herramienta computacional para facilitar su uso en la industria, aún existen desafíos significativos para asegurar su viabilidad a lo largo del tiempo. Algunos de los problemas señalados en los estudios primarios para justificar esta situación se relacionan con la obsolescencia de las herramientas y la falta de su actualización. Después de analizar los datos, se consideró que este escenario podría ser atribuible a diversos problemas técnicos, como el diseño de la herramienta para una versión anterior de un lenguaje de programación o sistema operativo, lo que requeriría esfuerzos por la parte de los especialistas, sin ayuda de los académicos, para actualizarla y adaptarla a un contexto más reciente, o bien porque la herramienta ya no se ajusta para abarcar nuevas necesidades de los usuarios. Los investigadores concluyeron al respecto que, por ende, la adaptación de un prototipo académico a un contexto industrial demanda más tiempo y recursos financieros de los que se establecen en un simple esfuerzo colaborativo. Además, se argumentó que una técnica algunas veces podría parecer prometedora en experimentos iniciales, pero que se requiere de una cantidad considerable de trabajo para que logre su incorporación real en un flujo de trabajo. Es decir, la técnica podría requerir de datos con los que no se cuenta o bien requerir la automatización de algún proceso manual. Por lo tanto, es necesario verificar la precisión y robustez de dicha herramienta antes de su uso práctico.

La *relación costo-beneficio* también fue un factor importante que destacó en la revisión de literatura. Por ejemplo, algunos especialistas afirmaron haber utilizado las propuestas creadas en la academia y que éstas al principio funcionaron bien, pero que desafortunadamente el costo asociado con, por ejemplo, un 1% de errores omitidos era demasiado alto. En otras palabras, aunque la herramienta computacional creada pudiera detectar el 99% de los errores, algunos especialistas son

escépticos acerca de utilizarla si el costo de los errores restantes es alto. Algo que, desde un punto de vista particular, pudiera sonar ridículo. Finalmente, un último factor que destacó entre las respuestas de los especialistas para justificar la falta de uso de las herramientas computacionales creadas en la academia fue la falta de familiarización con éstas. En este sentido, de los 23 especialistas que participaron en el estudio, el 60% reveló no haber estado familiarizado con las herramientas que fueron creadas como resultado de una investigación en la academia, lo que resaltó la falta de comunicación entre la academia y la industria. Aunado a esto, el 35% confirmó haber usado en paralelo otras herramientas computacionales para ayudarse durante la experimentación, pero desafortunadamente no se proporcionó información al respecto.

Como parte final de la investigación, la Figura 2.6 resume la relación que los investigadores encontraron entre los criterios de aplicabilidad y los enfoques que se han utilizado en la industria real. La figura muestra enfoques que fueron puestos en práctica en al menos dos estudios.

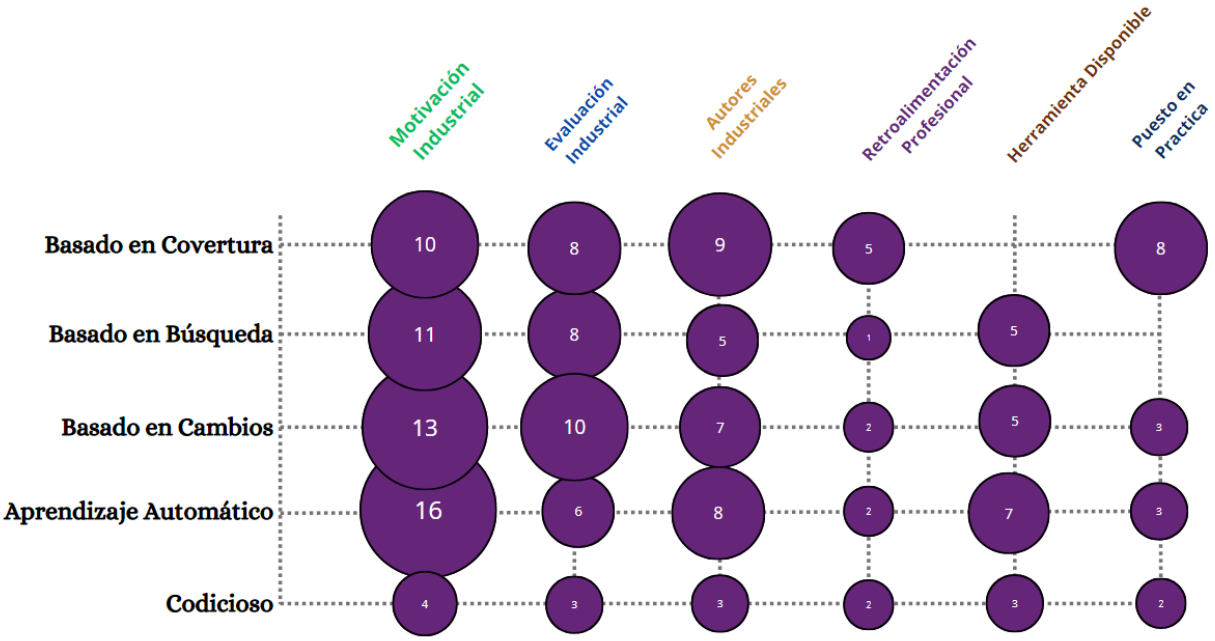


Figura 2.6. Mapeo de enfoques y técnicas que demostraron aplicación práctica en al menos dos artículos. Nota: Traducida de Greca et al., (2023)

Como era de esperar, los enfoques de tipo información y algorítmico más comunes son los más usados en el mundo real. Los *enfoques basados en la cobertura* dominan la implementación de las técnicas, a pesar de las preocupaciones sobre el costo de medir la cobertura, puesto que, aunque consumen un tiempo considerable, las mediciones de cobertura son fáciles de obtener para la mayoría de los lenguajes de programación. Por el contrario, los investigadores identificaron 16 estudios que proponen *enfoques de aprendizaje computacional*, de los cuales solamente tres fueron implementados, probablemente porque los modelos de aprendizaje computacional son tan buenos como los datos que reciben y a menudo la obtención de datos de suficiente volumen y calidad es más difícil que implementar el método en sí.

Los hallazgos presentados en esta revisión sistemática de literatura aportan un panorama interesante sobre la actualidad de las pruebas de regresión en la industria. Pero además de toda la información resumida anteriormente, uno de los descubrimientos más importantes, para el contexto de esta tesis, es que **todas las investigaciones que fueron analizadas en la revisión involucran a**

organizaciones desarrolladoras de software de tamaño grande en donde se desarrolla exclusivamente software a gran escala[8]. De hecho, otras investigaciones similares realizadas por Ali et al. (2019), Minhas et al., (2020), Mendonça et al. (2022), y Meyer et al. (2023) confirman que **existe un marcado interés en la creación de propuestas que mejoren la ejecución de las pruebas de regresión para el software a gran escala**. Por lo tanto, con el objetivo de desarrollar una solución que se ajuste al contexto de las pequeñas organizaciones desarrolladoras de software, se presentan a continuación los resultados de un estudio cualitativo/cuantitativo que se llevó a cabo con el fin de obtener un panorama más específico sobre la práctica habitual para realizar las pruebas de regresión en este tipo de organizaciones. Esta información fue crucial para desarrollar una solución que abordará los objetivos establecidos al inicio de esta tesis.

2.4. Diagnóstico del proceso de pruebas de regresión en pequeñas organizaciones

La investigación de Minhas (2022) argumenta que la mayoría de las organizaciones desarrolladoras de software de tamaño pequeño prefieren omitir la realización de pruebas de regresión en sus procesos de desarrollo de software, debido a que menudo las consideran un proceso complejo y desafiante. Dado que se sabe que este tipo de pruebas implica que el mínimo cambio en el código fuente puede suponer modificaciones funcionales en el resto del software que afecten negativamente la satisfacción del cliente, la lógica indicaría que su realización es crucial para evitar problemas más graves y costosos a futuro. Sin embargo, dentro de estas pequeñas organizaciones regularmente se tiene la falsa creencia de que la realización de las pruebas de regresión requiere de la contratación de personal altamente cualificado; además de que retrasan la entrega del software, introducen el continuo retrabajo al intentar corregir los fallos detectados, y establecen una serie de condiciones que estas organizaciones no pueden satisfacer. Nada más lejano de la realidad, puesto que al pensar fielmente que la calidad del software puede generarse por arte de magia se está estableciendo el escenario ideal para el fracaso. En este contexto, resulta importante mencionar que una buena parte de las pequeñas organizaciones desarrolladoras de software enfrenta problemas bastante particulares que a menudo le dificultan la realización efectiva de los procesos de software.

La investigación de Ali et al. (2020), por ejemplo, estableció que a este tipo de organizaciones se le dificulta la realización de las pruebas de regresión puesto que sus recursos económicos y humanos son escasos, además de que las limitaciones y/o restricciones de cada proyecto (e.g., costo, tiempo, calidad) obligan a la rápida entrega del software. Aunado a esto, Argüello (2020), Salgado (2020) y Cieza y Tantajulca (2023) consideran que estas organizaciones enfrentan problemas que afectan la realización apropiada de las pruebas sobre el software, entre los que destacan la comunicación deficiente entre los desarrolladores y *testers*, o entre los desarrolladores y los usuarios; las limitaciones de personal y presupuesto; la ausencia de las competencias adecuadas sobre las pruebas; la carencia de un monitoreo adecuado; la falta de herramientas y marcos metodológicos; el hábito incorrecto de no guardar información histórica de los proyectos; la falta de involucramiento del personal; y la resistencia al cambio. Por otro lado, el estudio realizado por Garousi et al., (2020) concluyó que en las pequeñas organizaciones desarrolladoras de software existe, desafortunadamente,

[8] De acuerdo con Kasauli et al., (2021), el desarrollo de software a gran escala se caracteriza por presentar un mayor nivel de dificultad en comparación con el desarrollo de software convencional. Es decir, cuando se habla de software a gran escala se refiere a aquel tipo de software que exhibe los siguientes aspectos: funcionalidad extensa y compleja; dependencia de una importante variedad de requisitos no funcionales (i.e., restricciones); procesamiento de una vasta cantidad de información; alta modularización y desacoplamiento entre vistas-interacción, lógica y modelos de datos; tanto el procesamiento, la información y el control se encuentran distribuidos; e integración de diferentes sistemas de software, hardware y comunicaciones.

una total desconexión entre la investigación y la práctica sobre las pruebas de software. Es decir, se argumenta que la investigación sobre esta área se ha centrado en gran medida en hacer que las pruebas sean "mejores" en términos técnicos, lo que incluye el mejorar el diseño de las pruebas, diseñar herramientas computacionales para respaldar las pruebas, medir la cobertura de las pruebas, etc. Desde la perspectiva de estos investigadores, esto no ayuda en la práctica a las pequeñas organizaciones, ya que la cuestión clave es ayudarlas a planear, diseñar y controlar las pruebas que son más efectivas para satisfacer las necesidades organizacionales, de tal manera que se logre minimizar el esfuerzo y el tiempo requerido para demostrar que el software es "suficientemente bueno". Por lo tanto, las investigaciones deben extenderse para abordar la relación que existe entre las pequeñas organizaciones y los procesos con los que cuentan para realizar las pruebas de regresión. Esta afirmación de los investigadores es importante, puesto que actualmente se considera que una inmensa mayoría de las organizaciones desarrolladoras de software en el mundo son de tamaño micro, pequeño o mediano y, obviamente, una buena parte de sus ingresos depende de las continuas modificaciones que realizan a sus productos de software.

Por lo tanto, con el objetivo de determinar una visión más actual y aterrizada al contexto de nuestro país con relación a la definición y realización de las pruebas de regresión en este tipo de organizaciones, se presentan a continuación los resultados de realizar un diagnóstico sobre un conjunto de 34 pequeñas organizaciones desarrolladoras de software distribuidas entre siete entidades federativas de México.

2.4.1. Participantes

Se invitó a 52 micro y/o pequeñas organizaciones desarrolladas de software a participar en un estudio corto que implicó el responder un cuestionario diseñado para obtener información relevante sobre su comprensión y realización de las pruebas de regresión sobre el software. Estas organizaciones fueron ubicadas de dos formas: (1) a través de una búsqueda en internet y (2) mediante el contacto con estudiantes egresados a nivel licenciatura y posgrado que trabajan en éstas. Del conjunto inicial de organizaciones, solamente 34 (2 micro y 32 pequeñas) aceptaron participar en el estudio (véase Tabla 2). En este sentido, a cada una de estas organizaciones se les pidió que participaran en el diagnóstico con al menos cinco empleados que estuvieran involucrados en la definición y/o ejecución de las pruebas sobre el software. Además, se dieron indicaciones para que la selección de estos empleados se basara en tres aspectos fundamentales: su involucración directa en el proceso de pruebas, una antigüedad de por lo menos dos años en el rol desempeñado, y la disposición del empleado por participar en el estudio. Es importante mencionar que cuatro organizaciones argumentaron no realizar pruebas de regresión a sus productos, dos organizaciones no aceptaron la invitación para participar, y 12 no respondieron a la invitación.

Considerando lo anterior, un total de 135 participantes, entre líderes de proyectos, programadores y *testers*, respondieron el cuestionario. De manera adicional, además del tesista, dos investigadores con 20 años de experiencia en el área de pruebas contribuyeron tanto en el diseño del cuestionario de diagnóstico como en el análisis e interpretación de las respuestas.

Tabla 2. Características de las organizaciones participantes en el diagnóstico

#	Tamaño	Giro	Entidad federal	Personal de software	Participantes en la encuesta	Roles
1	Pequeño	Software a la medida, desarrollo web	Baja California Norte	42	5	Líder de proyecto, *testers*

#	Tamaño	Giro	Entidad federal	Personal de software	Participantes en la encuesta	Roles
2	Pequeño	Software especializado (servicios financieros)	Baja California Norte	27	4	Programadores, *testers*
3	Micro	Software para plataformas de gestión del ciclo de los datos	Baja California Norte	9	2	*Testers*
4	Pequeño	Software a la medida, soporte	Baja California Norte	23	3	Líder de proyecto, *testers*
5	Pequeño	Software a la medida, automatizaciones y orquestación de servicios de nube	Baja California Norte	31	6	Líder de proyecto, programadores, *testers*
6	Pequeño	Software a la medida, consulta en mejora	Baja California Sur	23	4	Programadores, *testers*
7	Pequeño	Software RPA (*Robot Process Automatization*)	Baja California Sur	36	5	Líder de proyecto, *testers*
8	Pequeño	Software especializado (industria de la construcción)	Baja California Sur	25	3	Programadores, *testers*
9	Pequeño	Software a la medida, consultoría en mejora	Ciudad de México	40	5	Líder de proyecto, programadores, *testers*
10	Pequeño	Software especializado (servicios financieros)	Ciudad de México	39	5	*Testers*
11	Pequeño	Software a la medida	Ciudad de México	27	5	Líder de proyecto, programadores, *testers*
12	Pequeño	Software y plataformas de ciberseguridad	Ciudad de México	16	4	*Testers*
13	Pequeño	Software a la medida	Ciudad de México	23	5	Programadores, *testers*
14	Pequeño	Software especializado (servicios financieros)	Ciudad de México	20	3	*Testers*
15	Pequeño	Software a la medida, consultoría en mejora	Ciudad de México	29	4	Programadores, *testers*
16	Pequeño	Software para desarrollo de aplicaciones en microservicios	Ciudad de México	35	5	Líder de proyecto, programadores, *testers*
17	Pequeño	Software a la medida, desarrollo web	Ciudad de México	18	4	Programadores, *testers*
18	Pequeño	Software especializado (servicios financieros)	Ciudad de México	30	2	*Testers*

#	Tamaño	Giro	Entidad federal	Personal de software	Participantes en la encuesta	Roles
19	Micro	Software para uso específico de la empresa	Oaxaca	11	3	Programadores, *testers*
20	Pequeño	Software especializado (servicios financieros)	Oaxaca	13	4	Líder de proyecto, programadores
21	Pequeño	Software a la medida, desarrollo web	Oaxaca	16	5	Programadores, *testers*
22	Pequeño	Software a la medida	Oaxaca	10	3	Programadores, *testers*
23	Pequeño	Software a la medida	Oaxaca	11	4	Programadores, *testers*
24	Pequeño	Software a la medida, desarrollo web	Oaxaca	10	2	*Testers*
25	Pequeño	Software a la medida, consultoría en mejora	Puebla	20	4	Líder de proyecto, *testers*
26	Pequeño	Software a la medida, desarrollo web	Puebla	16	5	Líder de proyecto, programadores, *testers*
27	Pequeño	Software a la medida, desarrollo web	Puebla	23	5	Líder de proyecto, programadores, *testers*
28	Pequeño	Software a la medida, desarrollo web	Puebla	25	5	Líder de proyecto, programadores, *testers*
29	Pequeño	Software como servicio (SaaS), integraciones y automatizaciones	Querétaro	37	3	*Testers*
30	Pequeño	Software a la medida, desarrollo web	Querétaro	19	2	*Testers*
31	Pequeño	Software especializado (servicios financieros)	Querétaro	26	5	*Programadores, testers*
32	Pequeño	Software a la medida, desarrollo web	Querétaro	16	4	Líder de proyecto, *testers*
33	Pequeño	Software a la medida, desarrollo web	Tlaxcala	14	3	*Testers*
34	Pequeño	Software a la medida, desarrollo web, consultoría en mejora	Tlaxcala	19	4	*Programadores, testers*
				Total:	135	

2.4.2. Instrumentos

Se diseñó un cuestionario de 24 preguntas para recoger información mediante respuestas de opción múltiple y abiertas que abordan los aspectos fundamentales de las pruebas de regresión que

fueron presentados en secciones anteriores de este capítulo (véase Tabla 3). En resumen, se buscó recoger la percepción de los participantes sobre los siguientes seis puntos:

- Formación académica del personal involucrado en la realización de las pruebas.
- Correspondencia entre la formación y el rol desempeñado (principalmente de *tester*).
- Tipos de pruebas de regresión realizadas.
- Técnicas usadas para realizar las pruebas.
- Estrategia seguida para aplicar la técnica.
- Herramientas de soporte a las pruebas.

El cuestionario se diseñó de tal forma que fuera posible recoger información tanto de organizaciones donde las pruebas de regresión fueran parte del proceso utilizado para desarrollar los productos de software, como de organizaciones donde no se tenía la seguridad de realizar las pruebas de regresión, pero sí otro tipo de prueba sobre el software. Finalmente, se utilizó una hoja de cálculo de Excel© para reunir, analizar e interpretar la información recogida por los cuestionarios.

Tabla 3. Cuestionario diseñado para realizar el diagnóstico sobre la comprensión de las pruebas de regresión en las pequeñas organizaciones desarrolladoras de software

Cuestionario de diagnóstico para organizaciones desarrolladoras de software
Estimado(a) participante, el presente cuestionario tiene por finalidad recoger información sobre la práctica habitual para diseñar y conducir pruebas de regresión sobre el software que es producido en tu organización. Dicha información será completamente anónima y por ello te agradeceremos que respondas las preguntas del cuestionario con sinceridad, seriedad y desde la perspectiva del rol que tu desempeñas en la organización. Agradecemos tu apoyo y generosidad para participar en esta actividad.
Instrucciones: Por favor escoge la respuesta correcta y/o introduce la información que se requiera. Si consideras que tu respuesta a alguna pregunta requiere de la selección de más de una respuesta, por favor hazlo
1. Con relación a tu formación académica, ¿cuál es la licenciatura/ingeniería que estudiaste?
2. Considerando las siguientes actividades, ¿con cuál de éstas se relaciona mejor la organización en la que laboras actualmente? • Desarrollo de software a la medida. • Diseño y desarrollo web. • Desarrollo de software empotrado. • Desarrollo de paquetería contable. • Desarrollo de software educativo. • Otro (especifica):
3. ¿Cuál de los siguientes roles tiene más relación con la actividad que desempeñas en la organización? • Analista de requisitos. • Arquitecto o diseñador de software. • Programador. • *Tester*. • Líder de proyectos, Scrum Master o función gerencial. • Otro (especifica):
4. ¿Cuánto tiempo (años) tienes trabajando en la organización y desempeñando dicho rol desde tu incorporación a la industria? Años en la organización _____ Años desempeñando el rol _____

<table>
<tr><th colspan="2">Cuestionario de diagnóstico para organizaciones desarrolladoras de software</th></tr>
<tr><td colspan="2">5. ¿Cuánto tiempo (años) tienes trabajando en la industria de desarrollo de software?</td></tr>
<tr><td colspan="2">6. ¿Consideras que tu formación académica es la apropiada para desempeñar este rol?
• Si.
• No.
• No lo sé.</td></tr>
<tr><td colspan="2">7. ¿Estás satisfecho con tu desempeño en este rol?
• Si.
• No (por favor indica las razones):</td></tr>
<tr><td colspan="2">Las pruebas de regresión suelen ser un tipo de prueba de software que se utiliza para confirmar que los cambios más recientes sobre un programa o código no hayan afectado las funcionalidades existentes de manera adversa. Por lo tanto, durante las pruebas de regresión se determina que el software o la aplicación funcione bien con respecto a los nuevos cambios y correcciones de errores. Así, las pruebas de regresión consisten principalmente en volver a ejecutar los casos de prueba ya ejecutados con el fin de confirmar si el software funciona correctamente.</td></tr>
<tr><td colspan="2">8. Considerando la definición anterior, ¿las actividades relacionadas con el rol que desempeñas en la organización se relacionan de alguna manera con este tipo de pruebas?
• Si.
• No.</td></tr>
<tr><td>Respuesta Si</td><td>Respuesta No</td></tr>
<tr><td>9. ¿En qué fase del ciclo de vida del software participas para realizar las pruebas de regresión?
• Programación.
• Pruebas.
• Mantenimiento.
• Otra (especifica):</td><td>9. ¿Que otro tipo de prueba utilizas para validar o verificar la calidad del producto que estás generando con tu trabajo?
• Revisión con listas de comprobación para los requisitos y elementos del diseño.
• Pruebas unitarias.
• Pruebas de integración.
• Pruebas de aceptación.
• Pruebas de usabilidad.
• Otra (especifica):</td></tr>
<tr><td>10. ¿Qué tipo de pruebas de regresión se realizan en la fase que indicaste?
• Correctivas.
• Retest-all.
• Selectivas.
• Progresivas.
• Completas.
• Parciales.
• Unitarias.</td><td>10. ¿Cuál consideras que es la razón principal por la que no se promueve la realización de pruebas de regresión en tu organización?</td></tr>
<tr><td>11. ¿Qué técnica utilizas para realizar las pruebas de regresión?
• Selección de los casos de prueba.
• Priorización de los casos de prueba.
• Reducción de los casos de prueba.
• Optimización de los casos de prueba.</td><td>11. Tomando en cuenta el objetivo que persiguen las pruebas de regresión ¿consideras que deberían entrenar al personal de la organización para introducirlas en la organización y mejorar así la calidad del software?
• Si.
• No.
• No sé.</td></tr>
</table>

<table>
<tr><th colspan="2">Cuestionario de diagnóstico para organizaciones desarrolladoras de software</th></tr>
<tr><td>• Otra (especificar):
• ¿Alguna combinación? (especificar):</td><td></td></tr>
<tr><td>12. ¿En qué se basa la estrategia utilizada en la organización para aplicar la(s) técnica(s) anterior(es) en los proyectos?
• Código fuente.
• Modelos generados (por ejemplo, diagramas de clases, diagramas de secuencia, etc.).
• Datos históricos.
• Casos de prueba.
• Inteligencia Artificial.
• Heurísticas.
• Otra (especificar):
• ¿Alguna combinación? (especificar):</td><td>12. En el caso hipotético de que tu organización aceptará capacitarte para que adquirieras la habilidad de realizar las pruebas de regresión ¿te comprometerías a modificar tus hábitos de trabajo?
• Si.
• No.
• No sé.

Nota: Recuerda que el entrenamiento a menudo requiere más dedicación, tiempo, compromiso, y nuevas responsabilidades y no necesariamente involucra una mejora salarial en la organización donde se realice.</td></tr>
<tr><td colspan="2">13. ¿En tu organización se practica el almacenamiento y reutilización de la información generada en cada proyecto que se realiza? Esta información histórica suele incluir artefactos como planes de proyectos, requisitos, diseño de alto y bajo nivel, códigos documentados, casos de prueba, etc.
• Si
• No</td></tr>
<tr><td>Respuesta Si</td><td>Respuesta No</td></tr>
<tr><td>14. ¿Con qué frecuencia de tiempo consideras que se consulta y/o utiliza la información histórica para probar cada nuevo proyecto?
• Entre el 100% y el 75% de las veces.
• Entre el %74 y el 50% de las veces.
• Entre el 49% y 25% de las veces.
• Entre el 24% y el 1% de las veces.
• Nunca.</td><td>14. ¿Con qué frecuencia de tiempo consideras que tu trabajo se ve afectado negativamente por no utilizar la información generada en los proyectos anteriores?
• Entre el 100% y el 75% de las veces.
• Entre el %74 y el 50% de las veces.
• Entre el 49% y 25% de las veces.
• Entre el 24% y el 1% de las veces.
• Nunca.</td></tr>
<tr><td>15. ¿En tu organización se gestiona un conjunto maestro de casos de prueba que facilita la ejecución de las pruebas de regresión?
• Si.
• No (en caso de escoger esta respuesta, por favor especifique de dónde obtiene los casos de prueba).</td><td>15. Considerando tus respuestas para las preguntas 6 y 10 ¿cómo obtienes los casos de prueba que utilizas en las pruebas que realizas?</td></tr>
<tr><td colspan="2">16. ¿A través de qué medio síncrono o asíncrono sueles interactuar con tus compañero(a)s de equipo durante la realización de las pruebas?
• Reuniones virtuales (Meet, Zoom, Facetime, WhatsApp, etc.).
• Correos electrónicos.
• Reuniones presenciales.
• Informes impresos.
• Otro (especifica):</td></tr>
</table>

<table>
<tr><th colspan="2">Cuestionario de diagnóstico para organizaciones desarrolladoras de software</th></tr>
<tr><td colspan="2">17. ¿Qué herramienta computacional utilizas para intercambiar información con tu equipo de trabajo sobre las pruebas realizadas sobre el software?
• Fracttal.
• JIRA.
• GitLab.
• Zephyr.
• Ninguna.
• Otra (especifica):</td></tr>
<tr><td colspan="2">18. ¿En tu organización se cuenta con una herramienta computacional, ya sea propietaria o de código abierto, que te facilite la gestión de las pruebas que realizas sobre el software?
• Si.
• No.</td></tr>
<tr><td>Respuesta Si</td><td>Respuesta No</td></tr>
<tr><td>19. ¿Esta herramienta fue el producto que se obtuvo de algún proyecto vinculado con la academia?
• Si.
• No.
• No lo sé.</td><td>19. ¿Consideras que de tenerla se podrían mejorar los resultados de las pruebas?
• Si.
• No.
• No lo sé.</td></tr>
<tr><td colspan="2">20. ¿Consideras que es apropiado el número de defectos y/o fallos con que se libera un software que fue creado y modificado en tu organización al agregar nuevas funcionalidades?
• Si.
• No (justifica porqué):
• No aplica.</td></tr>
<tr><td colspan="2">21. ¿Consideras que la realización de pruebas sobre un producto que está siendo modificado constantemente solamente incrementan el presupuesto y tiempo asignado al proyecto?
• Si.
• No.
• No lo sé.</td></tr>
<tr><td colspan="2">22. ¿Qué indicador o medida se utiliza en la organización para evaluar si las pruebas sobre el software fueron ejecutadas correctamente?
• Número de defectos encontrados.
• Tasa de errores.
• Tiempo consumido.
• Esfuerzo dedicado.
• Otro (especifica):</td></tr>
<tr><td colspan="2">23. Si tu respuesta a la pregunta 8 fue afirmativa, indica secuencialmente las actividades que se llevan a cabo para realizar las pruebas de regresión.</td></tr>
<tr><td colspan="2">24. Finalmente, en términos generales, ¿cuáles consideras que serían las funcionalidades mínimas que una herramienta para gestionar la información generada antes, durante y después de la realización de las pruebas debería cubrir?</td></tr>
</table>

2.4.3. Proceso y resultados

Dado que las empresas que participaron en el diagnóstico son de siete diferentes estados de la República Mexicana se decidió que el cuestionario propuesto fuera diseñado con *Google Forms* a través de la suite gratuita de *Google Docs Editors* (véase Figura 2.7). La decisión de utilizar esta herramienta gratuita es que permite obtener información precisa de forma rápida, facilita el análisis de las respuestas, genera gráficas de resultados en tiempo real, permite la exportación de los datos a hojas de cálculo, y maneja diferentes tipos de preguntas. El proceso que se siguió se resume a continuación:

- Se envío un enlace de invitación al correo de todos los participantes con el fin de que accedieran y respondieran el cuestionario. El correo enviado definía el objetivo del cuestionario en el contexto de la investigación que se estaba realizando, agradecía el apoyo brindado, establecía un tiempo límite para proporcionar respuestas, y daba algunas indicaciones generales.
- Posteriormente, una vez que se agotó el plazo definido para responder el cuestionario, se procedió a analizar la información con las mismas herramientas proporcionadas por *Google Forms*, y se realizó un análisis paralelo con la exportación de los datos a una hoja de cálculo de Excel©.
- Por último, se redactaron conclusiones a partir de los datos recopilados. Es importante recalcar que esta información fue de suma importancia para diseñar la solución que se presentan en el Capítulo 3 de esta tesis.

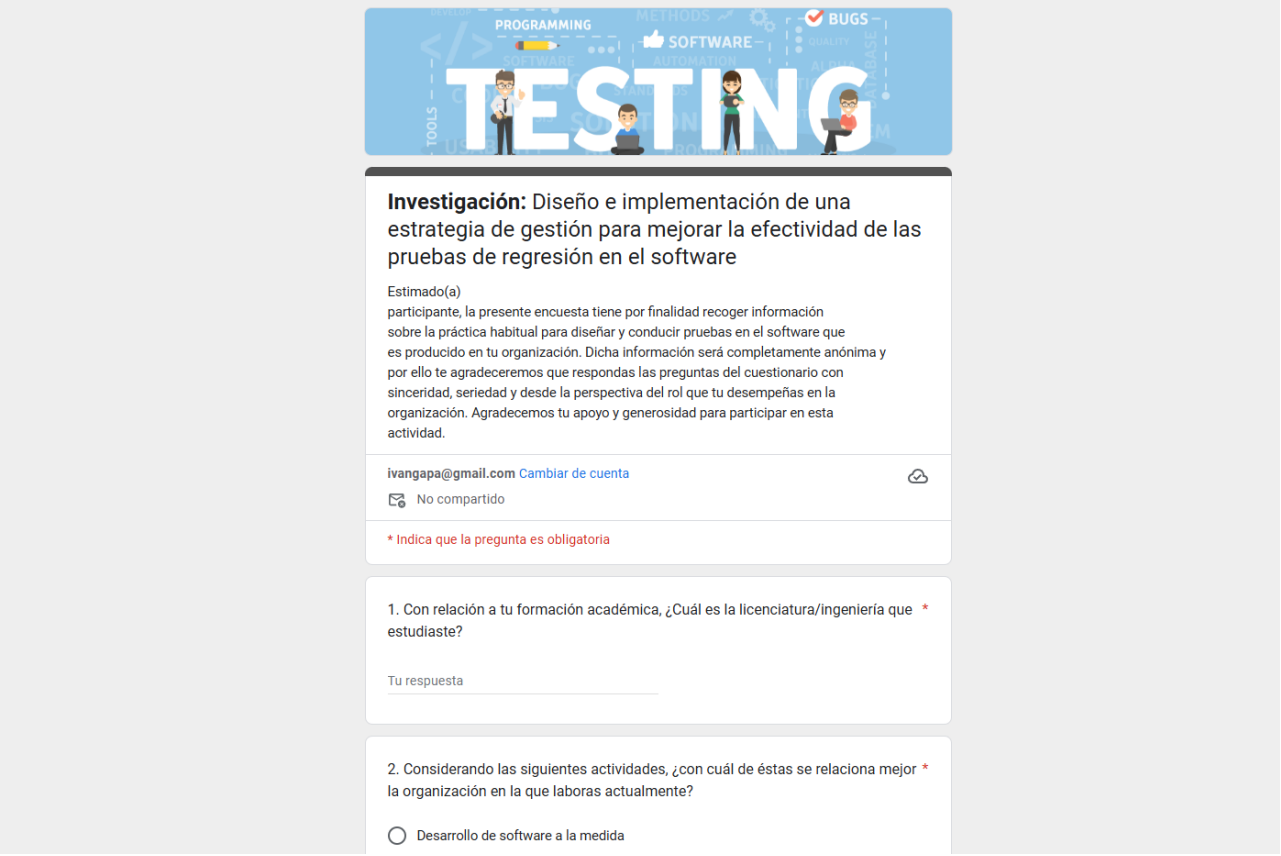


Figura 2.7. Ejemplo del cuestionario diseñado en Google Forms para el estudio diagnóstico

La aplicación del cuestionario reveló información relevante sobre los conocimientos que poseen las personas que participan en el desarrollo de software y la realización de las pruebas que se hacen sobre el software dentro de las pequeñas organizaciones que participaron en el diagnóstico. Las respuestas a la pregunta sobre los estudios cursados, por ejemplo, evidenció que el 65% de los encuestados son ingenieros en computación o licenciados informáticos, lo que corresponde al perfil esperado para el desarrollo de software, ya que esta formación proporciona las bases técnicas fundamentales para la creación de programas y/o aplicaciones de software. Por otro lado, el 20% de los participantes indicó haber egresado de las carreras de Ingeniería en Electrónica o Mecatrónica, disciplinas que, aunque en algunos casos se pueden relacionar con el desarrollo de software empotrado e interfaces, no están completamente alineadas con las competencias específicas necesarias para el desarrollo de software a nivel industria. Además, otro 10% de los participantes afirmó haber estudiado la carrera de Ingeniería Industrial, que, aunque es verdad que estos profesionistas pudieran contar con algunos fundamentos en cuanto a la definición formal de procesos y/o metodologías, no están entrenados en las metodologías propias del desarrollo de software. Finalmente, un 5% de los participantes indicó haber estudiado otras carreras que son completamente ajenas tanto al desarrollo de software como a la realización de pruebas. Estos resultados indican que una mayoría significativa de los participantes (65%) poseen el entrenamiento adecuado para desenvolverse profesionalmente en la industria de software, pero, desafortunadamente, **el 35% del personal encuestado no cuenta con los conocimientos requeridos para realizar eficientemente esta labor en las organizaciones desarrolladoras de software**. En este sentido, esta distribución de perfiles podría repercutir negativamente en la capacidad de las organizaciones participantes en el estudio para cumplir con los estándares de calidad requeridos en los productos de software.

Con relación a las actividades económicas que realizan las organizaciones involucradas en el diagnóstico, se determinó que el 80% se enfoca al desarrollo de software a la medida, lo que refleja una tendencia clara de este tipo de organizaciones por enfocarse al desarrollo de soluciones personalizadas en el ámbito organizacional. Es importante mencionar que otro 15% de las organizaciones está centrado en el diseño y desarrollo web, lo que concuerda con la relevancia de este tipo de sistemas en la era digital. En menor proporción, el desarrollo de software educativo es abordado por el 5% de las organizaciones. De esta manera, **la información recogida evidencia que las pequeñas organizaciones tienden a especializarse en áreas tecnológicas clave, especialmente en la creación de soluciones personalizadas y plataformas digitales**.

Por otro lado, considerando el rol desempeñado por los participantes en este diagnóstico, **el 75% afirmó desempeñar actividades que se relacionan con el rol de programador**. Este dato refleja la alta demanda de personal con conocimientos sólidos en programación y desarrollo de software que se presenta dentro de las organizaciones de este sector. Por otro lado, el *tester*, responsable de realizar las pruebas al software, fue escogido por el 23% de los participantes como el rol que mejor se ajusta a sus actividades diarias. Esta información indica que, si bien la programación es la actividad predominante, las pruebas de software ocupan un lugar significativo dentro de las actividades que desempeña el personal involucrado en los procesos de desarrollo del software. En contraste, solamente un 2% de los encuestados señaló que desempeña funciones gerenciales de líder de proyectos o Scrum Master.

Al determinar los años de experiencia de los participantes en la organización donde laboran actualmente, **se identificaron diferentes antigüedades que van desde quienes tienen menos de 2 años, hasta aquellos con trayectorias más amplias de entre 6 y 8 años**. Un grupo considerable de los participantes mencionó contar con entre 2 y 6 años de antigüedad; sin embargo, la mayoría de las respuestas se concentran en un rango intermedio lo que indica una diversidad en cuanto a las trayectorias y niveles de experiencia de los encuestados dentro de la compañía para la que trabajan.

Aunado a lo anterior, los encuestados con más tiempo en la organización suelen tener roles sólidos mientras que aquellos con menos tiempo destacan en roles iniciales o de reciente incorporación.

Por otra parte, en base a la experiencia global en la industria los resultados reflejan una amplia diversidad de tiempo entre los encuestados. Los datos muestran 2 años de trayectoria para aquellos profesionistas que se encuentran en etapas iniciales (i.e., reciente incorporación) hasta aquellos con más de una década de experiencia. En este sentido, **la mayoría de los participantes se encuentra en rangos intermedios, lo que puede indicar que gran parte del grupo tiene una trayectoria sólida en dicho sector**. De esta manera es posible mencionar que los encuestados con menor tiempo en la industria tienden a ocupar roles asociados al aprendizaje y adaptación de nuevas tecnologías, mientras que los encuestados con trayectorias significativas suelen estar en puestos tales como liderazgo o especialización técnica, lo que destaca la acumulación de experiencia como un factor clave en el desarrollo profesional.

Sobre los conocimientos del personal para desempeñar el rol asignado en la organización, **el 65% de los participantes que cuenta con estudios en Ingeniería en Computación o Licenciatura en Informática consideró que su formación académica es adecuada para desempeñar su rol** ya que, de acuerdo con su opinión, cuentan con las bases técnicas necesarias para el desarrollo de software. Sin embargo, el 20% que egresó de carreras como Ingeniería en Electrónica o Mecatrónica afirman que es necesario que continúen adquiriendo conocimientos adicionales sobre metodologías y prácticas específicas de la industria de software para desempeñar eficientemente su rol correspondiente. Para el caso del 10% de los participantes con formación en Ingeniería Industrial y el 5% con estudios en áreas completamente ajenas al desarrollo de software, los encuestados consideran completamente que, si bien su formación les aporta habilidades generales, no son suficientes ya que necesitan reforzar competencias técnicas y prácticas relacionadas directamente con el desarrollo y las pruebas de software para cumplir con los estándares requeridos por la industria.

Del 23% de los participantes que desempeñan el rol de *tester*, **un 90% de ellos indicó no estar satisfecho con su desempeño**. La principal razón mencionada de esta insatisfacción fue la falta de conocimiento sobre cómo realizar las pruebas adecuadamente. Este resultado evidencia una necesidad significativa de capacitación y formación específica para los *testers*, ya que la falta de habilidades y conocimientos técnicos en esta área podría impactar negativamente en la calidad del software desarrollado por las organizaciones participantes.

De acuerdo con los datos recogidos para la pregunta sobre si las actividades relacionadas con el rol que desempeñan los participantes en la organización están vinculadas con las pruebas de regresión, se encontró que el 87% de los participantes indicó que sí. Esto quiere decir que **la mayoría de los roles desempeñados en las organizaciones en las que se aplicó el cuestionario tienen alguna relación con las pruebas de regresión**, lo cual es una buena señal puesto que se considera importante el garantizar que los cambios recientes en el software no afecten negativamente las funcionalidades ya existentes. Este resultado destaca la relevancia de las pruebas de regresión como una práctica común y necesaria dentro del desarrollo de software, subrayando la necesidad de que los empleados estén capacitados para llevarlas a cabo de manera eficiente. Por otro lado, este resultado también podría reflejar que el 13% restante de los participantes realiza actividades que no están directamente relacionadas con este tipo de pruebas o carecen de claridad sobre su relación con ellas.

Al preguntarle a los participantes sobre la fase del ciclo de vida del software en la que realizan las pruebas de regresión, el 43% indicó que éstas se ejecutan en la fase de programación, lo que sugiere que una parte significativa de las pruebas de regresión se realiza de manera temprana, probablemente para validar que los cambios realizados al escribir o modificar el código no afecten las funcionalidades ya existentes. El 38% señaló que realiza las pruebas de regresión durante la fase de *testing*, lo cual es

esperado, ya que esta fase se enfoca específicamente en garantizar la calidad del software a través de diversas pruebas, incluidas las de regresión. El 19% realiza las pruebas de regresión en la fase de mantenimiento, lo que indica que estas pruebas también son relevantes para validar el correcto funcionamiento del software después de implementar actualizaciones o correcciones en sistemas ya terminados. **Estos resultados reflejan que las pruebas de regresión no están limitadas a una única fase del ciclo de vida del software, sino que se distribuyen a lo largo de todo el ciclo de vida del desarrollo de software, dependiendo de las necesidades y procesos de cada organización**.

Ahora bien, con relación al tipo de pruebas de regresión que se realizan en la fase indicada anteriormente por los participantes, se determinó lo siguiente: El 90% de los encuestados mencionó que realiza pruebas unitarias. Esto indica que la mayoría de estos participantes se enfoca en probar unidades específicas del código de forma aislada, para asegurarse de que los cambios recientes no afecten negativamente a las funcionalidades individuales. Por otro lado, el 10% restante indicó que realiza pruebas correctivas que se centran en verificar que los cambios realizados no hayan alterado funcionalidades previas del software, más allá de las pruebas unitarias. **Este resultado muestra que, en general, las pruebas unitarias predominan en las fases en las que se realizan las pruebas de regresión, lo que refleja una tendencia hacia la validación de componentes individuales antes de realizar pruebas más amplias del sistema desarrollado**.

De acuerdo con los datos recogidos para la pregunta relacionada con la técnica utilizada en las pruebas de regresión, se determinó que el 80% de los encuestados utiliza la selección de casos de prueba como técnica principal. Esto muestra que la mayoría de los participantes se enfocan en elegir de manera específica los casos de prueba más relevantes y críticos para asegurar que las funcionalidades existentes no se vean afectadas por los constantes cambios que se hayan realizado al software. Por otra parte, el 15% no supo cómo responder la pregunta, lo que podría indicar la falta de familiaridad con las técnicas de pruebas de regresión o claridad en sus responsabilidades diarias dentro de su entorno laboral. Así mismo, el 5% mencionó que no utiliza ninguna técnica para realizar las pruebas de regresión, lo que podría reflejar una brecha en los conocimientos o en los procesos establecidos dentro de las organizaciones para realizar este tipo de pruebas. Estos resultados muestran que **la técnica de selección de casos de prueba es la más comúnmente empleada**, pero también destacan áreas de mejora en cuanto a la capacitación y estandarización de las técnicas de pruebas de regresión dentro de estas pequeñas organizaciones.

Aunado a esto, la información recopilada para la pregunta sobre en qué se basa la estrategia utilizada en la organización para aplicar la o las técnicas de pruebas de regresión empleadas, se observó que el 97% de los encuestados mencionó a los casos de prueba. Esto indica que la mayoría de las organizaciones utiliza casos de prueba específicos para asegurar que las funcionalidades existentes no se vean afectadas por los cambios realizados en el software. En contraste, el 3% señaló que la estrategia se basa en modelos generados, como diagramas de clases, diagramas de secuencia u otros modelos de diseño. Así, una pequeña cantidad de organizaciones utiliza modelos visuales para guiar las pruebas de regresión, lo cual puede ser útil en proyectos más complejos o estructurados. En resumen, **la estrategia predominante en las pequeñas organizaciones para realizar las pruebas de regresión se enfoca en la selección y ejecución de los casos de prueba**, con un enfoque menor en el uso de modelos generados como base para las pruebas.

Con el objetivo de determinar si las pequeñas organizaciones que formaron parte del diagnóstico practicaban el almacenamiento y reutilización sistemática de la información generada en los proyectos (incluyendo elementos clave como planes de proyectos, requisitos, diseño de alto y bajo nivel, códigos documentados, casos de prueba, entre otros) que pueden servir como conocimiento histórico para futuros proyectos, se analizaron las respuestas proporcionadas por los participantes para llegar a las

siguientes conclusiones: **el 73% de los encuestados indicó que sí reutilizan la información generada en proyectos anteriores para mejorar la eficiencia en el rendimiento de las pruebas**. En este sentido, la frecuencia de reutilización se distribuye de la siguiente manera: el 30% de los participantes afirmó reutilizar entre el 74% y el 50% de las veces, el 50% entre el 49% y el 25% de las veces, y el 20% entre el 24% y el 1% de las veces. Por otro lado, el restante 27% de los encuestados indicó que no reutilizan la información histórica de los proyectos anteriores. Para este caso, el 70% de estos participantes afirmó que su desempeño se ve afectado entre el 100% y el 75% de las veces debido a esta falta de reutilización. Mientras que el otro 30% reportó enfrentar impactos negativos entre el 74% y el 50% de las veces debido a que ni siquiera se genera información histórica.

En un contexto diferente, **el 100% de los participantes coincidió en que es necesario gestionar un conjunto maestro de casos de prueba que facilite la realización de las pruebas de regresión**. Así, el 30% argumentó utilizar técnicas formales como la prueba del camino básico y la partición en clases de equivalencia para diseñar un conjunto de casos de prueba, el 14% afirmó que aprovecha los casos de prueba de proyectos anteriores, con ciertas modificaciones que incrementan el conjunto maestro, y el 56% depende de procesos *ad hoc*, lo que revela un enfoque menos estructurado y una posible área de mejora. Esta información deja en claro que, aunque todos los participantes reconocieron el valor de la reutilización de conocimiento, existe una variabilidad en la frecuencia de su aplicación y un impacto considerable en la productividad para quienes no la implementan.

Con relación a la información recogida, las interacciones entre los integrantes de los equipos durante la realización de las pruebas se dan mediante una combinación de medios síncronos y asíncronos. Es decir, el 55% de los encuestados indicó que utiliza reuniones presenciales como principal forma de comunicación. El 45% reportó que prefiere reuniones virtuales, empleando herramientas como *Google Meet*, *Zoom* y *WhatsApp* para coordinarse y colaborar. Cabe destacar que el 100% de los participantes señaló que, si bien no está establecido como una regla en su organización, es común combinar el uso reuniones presenciales y virtuales para garantizar una comunicación efectiva. **Estos resultados evidencian una tendencia a combinar medios de comunicación síncronos y asíncronos, que se adapta a las necesidades específicas del equipo y las condiciones del entorno de trabajo**. La preferencia por las reuniones presenciales sugiere la importancia del contacto directo para ciertos aspectos de las pruebas, mientras que las reuniones virtuales ofrecen flexibilidad y eficiencia, especialmente en contextos de trabajo remotos o híbridos.

Ampliando el análisis de las respuestas anteriores, en relación con la pregunta que se hizo a los participantes sobre las herramientas computacionales que utilizan para intercambiar información, se observó una clara preferencia por el uso de herramientas específicas para la gestión y el intercambio de información relacionada con las pruebas. De hecho, el 80% de los encuestados indicó que utiliza JIRA como su principal herramienta de comunicación y seguimiento. Esto refleja su amplia aceptación y eficacia para gestionar proyectos, realizar un seguimiento detallado de las incidencias, y colaborar en tareas relacionadas con las pruebas. Además, el 20% señaló que utiliza GitLab, una herramienta que, además de su funcionalidad para la gestión de repositorios, es valorada por sus capacidades para coordinar tareas y realizar seguimientos de proyectos integrados al ciclo de desarrollo y pruebas. La predominancia de JIRA destaca su popularidad y versatilidad en la gestión de pruebas de software, mientras que el uso de GitLab subraya su utilidad para equipos que trabajan en entornos DevOps o que buscan una integración directa con sus flujos de desarrollo. Este panorama sugiere que **las herramientas computacionales utilizadas de código abierto son clave para la organización y colaboración eficaz en los equipos de trabajo dentro de las pequeñas organizaciones**.

Considerando el uso de herramientas computacionales que soporten la correcta gestión de las pruebas de regresión, se les preguntó posteriormente a los participantes si contaban con alguna herramienta que facilitara esta tarea. En este sentido, **el 83% de los encuestados respondió que no se cuenta con una herramienta computacional específica para gestionar las pruebas** y solamente el 17% indicó que sí se disponía de una herramienta, y en estos casos se trata principalmente de soluciones de código abierto. Al cuestionamiento de que si dicha herramienta era el resultado de algún proyecto vinculado con la academia, el 100% de los encuestados que respondieron afirmativamente argumentaron desconocer su origen. Este dato refleja una clara desconexión entre las iniciativas académicas y las necesidades prácticas de la industria. Por otra parte, el 100% de los encuestados que actualmente no cuentan con una herramienta computacional señaló que ésta les ayudaría significativamente a mejorar los resultados de las pruebas. Por lo tanto, la información proporcionada por los participantes evidencia una falta generalizada de herramientas especializadas para la gestión de pruebas en las pequeñas organizaciones, lo cual representa una oportunidad para implementar soluciones que mejoren la gestión de las pruebas de regresión. Además, la desconexión entre la academia y la industria sugiere la necesidad de fomentar proyectos colaborativos que desarrollen herramientas computacionales alineadas con las demandas reales de los equipos de *testing*. Esto no solo podría mejorar el rendimiento de las pruebas, sino también fortalecer los vínculos entre la investigación académica y la práctica industrial.

La opinión de los participantes sobre si consideran que el número de defectos y/o fallos con que se libera un software en su organización era apropiado o no, reflejó una percepción mayoritariamente positiva. Es decir, **el 78% de los participantes consideró que el número de fallos presentes en el software al momento de su liberación es apropiado, incluso después de haber sido modificado para incluir nuevas funcionalidades**. Este resultado sugiere que, para la mayoría, el control de calidad aplicado es suficiente y cumple con las expectativas organizacionales. Por otro lado, un 22% de los encuestados opina que el número de defectos detectados al liberar el software no es adecuado, lo cual indica que, aunque minoritario, existe un grupo que percibe áreas de mejora en los procesos de desarrollo, pruebas o implementación del software. En conclusión, aunque la opinión favorable es predominante, la existencia de una proporción significativa de respuestas críticas resalta la importancia de revisar y fortalecer los procedimientos relacionados con el aseguramiento de calidad y la gestión de defectos durante el ciclo de vida del software.

Continuando con el análisis de las respuestas proporcionadas por los participantes, sobre la pregunta de que si se considera que la realización de pruebas sobre un producto que está siendo modificado constantemente incremente el presupuesto y tiempo asignados a un determinado proyecto, las opiniones reflejaron una visión dividida respecto al impacto de estas pruebas en términos de costos y tiempos en los proyectos de desarrollo de software. Es decir, el 53% de los encuestados consideró que, en efecto, la realización de pruebas en un código que se modifica constantemente únicamente aumenta el presupuesto y el tiempo asignados al proyecto y que, por ende, deberían evitarse. Esta percepción podría estar asociada a una subestimación de los beneficios que estas pruebas aportan en términos de calidad y estabilidad del producto. De manera similar, el 31% de los participantes opinó que la realización de pruebas sí incrementa los costos y tiempos de entrega, pero que éstas representan una actividad necesaria y valiosa para garantizar el desarrollo de un producto final robusto y confiable. El restante 16% de los encuestados expresó incertidumbre al no saber qué responder, lo cual pone de manifiesto una carencia de información o comprensión sobre la importancia que tienen las pruebas de regresión, especialmente en el contexto de las pequeñas organizaciones. En conclusión, **aunque existe una mayoría que considera que las pruebas generan incremento en los costos y tiempos del proyecto, los resultados también destacan la necesidad de fomentar la concientización y**

capacitación sobre las ventajas estratégicas de las pruebas de regresión. Esto es crucial para lograr un equilibrio entre calidad, eficiencia y competitividad en el desarrollo de software.

En relación con la utilización de algún indicador o medida para evaluar si las pruebas sobre el software fueron ejecutadas correctamente en el contexto de las pequeñas organizaciones, los datos recopilados mostraron que la mayoría de estas organizaciones emplean medidas centradas en los resultados directos de las pruebas. De esta forma, **el 60% de los encuestados indicó que utilizan la tasa de errores como el principal indicador para evaluar la ejecución de las pruebas**. Este enfoque resalta la importancia de medir la cantidad de fallos o problemas detectados en el software como una forma de evaluar la efectividad de las pruebas y, por lo tanto, la calidad del producto liberado. **Un 20% de los participantes mencionó que el indicador utilizado es el esfuerzo dedicado a la actividad**, sugiriendo que la cantidad de recursos, tiempo y trabajo invertido en las pruebas también se considera una medida importante de su éxito, aunque no necesariamente esté directamente vinculada a la calidad final del software. Finalmente, **otro 20% señaló que el número de defectos encontrados es la medida que emplean para evaluar el éxito de las pruebas**. Este enfoque se centra en la detección de problemas específicos, lo que puede proporcionar una visión clara de la efectividad de las pruebas en identificar áreas de mejora dentro del código. En resumen, aunque la mayoría de las pequeñas organizaciones participantes se enfocan en medidas relacionadas con la tasa de errores para evaluar las pruebas, también existen enfoques complementarios que consideran tanto el esfuerzo invertido como el número de defectos encontrados, lo que refleja una variedad de perspectivas y prácticas en la evaluación de la calidad del software.

Por otro lado, cuando se le pidió al 87% de los participantes que había respondido afirmativamente a la pregunta 8 que listaran las actividades que realizaban para llevar a cabo las pruebas de regresión, **el 93% indicó seguir un proceso estructurado que incluye tres actividades clave: selección de los casos de prueba, ejecución de las pruebas y reporte de los resultados**. Este enfoque refleja una metodología estándar orientada a asegurar que se cubren los casos relevantes de prueba y se documentan adecuadamente los resultados para su posterior análisis. El restante 7% mencionó un enfoque ligeramente diferente que incorpora una actividad adicional de análisis y detección de cambios en las variables, antes de proceder con la ejecución de los casos de prueba. Tras esta actividad, también incluyen a la obtención y reporte de resultados, lo que sugiere que, para este grupo de personas, el análisis de las variables antes de la ejecución es crucial para adaptar las pruebas a los cambios recientes en el software. En conclusión, la mayoría de las organizaciones siguen un proceso clásico de selección, ejecución y reporte de resultados para sus pruebas de regresión. Sin embargo, un pequeño grupo ha incorporado pasos adicionales para asegurar que los cambios en las variables sean debidamente analizados antes de ejecutar las pruebas, lo que refleja una atención más detallada a las modificaciones del software durante el proceso de regresión.

Por último, con relación a la última pregunta del cuestionario que indaga sobre las funcionalidades mínimas que una herramienta debería cubrir para gestionar la información generada antes, durante y después de la realización de las pruebas de regresión, las respuestas fueron ampliamente consensuadas. Los participantes encuestados destacaron una serie de funcionalidades mínimas necesarias para una gestión efectiva y completa del proceso de pruebas, las cuales se resumen a continuación:

- En primer lugar, se destacó la importancia de la planificación, que incluye al control de los cambios realizados al código y la creación del entorno de prueba. De acuerdo con las opiniones de los participantes, esto aseguraría que las modificaciones en el software se gestionen adecuadamente y que el entorno de pruebas esté correctamente configurado para realizar las pruebas de manera eficaz.

- La documentación de los casos de prueba fue identificada como otra funcionalidad fundamental por los participantes ya que facilitaría, de acuerdo con sus necesidades, la organización y el seguimiento de las pruebas realizadas.
- Por otro lado, los participantes argumentaron que el incluir la selección de los casos de prueba les facilitaría identificar qué pruebas ejecutar en función de los cambios realizados en el código y las áreas críticas del sistema. Si bien la ejecución de los casos de prueba es importante, los participantes consideran que una funcionalidad esencial sería la gestión de los resultados de dicha ejecución, puesto que les permitiría realizar una ejecución eficiente y ordenada de las pruebas.
- Además, los participantes destacaron la necesidad de una adecuada gestión de los casos de prueba, lo que implica monitorear aspectos como el tiempo de detección y corrección de los defectos encontrados, lo cual es crucial para mantener la calidad del software.
- Finalmente, la generación y divulgación de reportes fue considerada también una funcionalidad imprescindible por los participantes, ya que permitiría a los equipos de desarrollo y a los interesados en el proyecto recibir información clara y accesible sobre los resultados de las pruebas, facilitando así la toma de decisiones y el seguimiento de los avances.

En resumen, **las funcionalidades mínimas que los encuestados consideran necesarias para una herramienta de gestión de pruebas incluyen a la planificación, documentación y selección de casos de prueba, gestión de las pruebas, así como la generación de reportes**. Estos elementos son vistos como esenciales para asegurar una gestión eficiente y un seguimiento adecuado del proceso de pruebas de software.

2.5. Estado del arte

El garantizar la calidad y funcionalidad del software a lo largo de su ciclo de vida es un desafío constante. Como ya se ha mencionado a lo largo de este capítulo, una de las formas para lograr este objetivo es a través de la realización de las pruebas de regresión, con el fin de verificar que los cambios que se hayan realizado en el código no hayan introducido errores ni afectado negativamente las funcionalidades existentes. Esta práctica es especialmente vital en entornos de desarrollo ágil, donde las iteraciones rápidas y frecuentes son la pauta para los procesos. Por lo tanto, las pruebas de regresión son esenciales para mantener la integridad del software, ya que permiten identificar rápidamente los defectos introducidos por nuevas modificaciones. Sin embargo, estas pruebas pueden ser costosas y laboriosas, especialmente en sistemas complejos con una gran cantidad de casos de prueba y, como consecuencia, su definición, realización y gestión no son tareas fáciles.

En este sentido, a lo largo del tiempo se han desarrollado diversas investigaciones para mejorar y/u optimizar el proceso relacionado con las pruebas de regresión con el objetivo de mejorar su eficiencia y efectividad dentro de las organizaciones desarrolladoras de software. Así, con el fin de ofrecer un panorama actual sobre estas investigaciones, en las siguientes secciones se presenta un análisis de las propuestas actuales que han buscado introducir diferentes estrategias para darle mayor formalidad a las pruebas de regresión en el contexto de la industria de software. Este análisis pretende proporcionar información relevante sobre las mejores prácticas y áreas de oportunidad en el contexto de las pruebas de regresión que contribuya al avance de la investigación y mejora del proceso de aseguramiento de la calidad del software en general.

2.5.1. Un modelo temático y un método de recomendación de casos de prueba basada en el historial de pruebas para las pruebas de regresión

2.5.1.1. Objetivo

La investigación de Aman et al., (2018) argumentó que la evolución del software que se da a través de las modificaciones continuas, mejora positivamente las funcionalidades y/o corrige errores, pero que este efecto depende en gran medida de la realización de pruebas de regresión que eviten la aparición de nuevos fallos una vez que los cambios se hayan realizado. Sin embargo, se afirmó también que la ejecución de todos los casos de prueba tras cada modificación es en realidad algo impráctico y con frecuencia ineficaz, puesto que esto implica que tanto los costos como el tiempo de dedicación de los *testers* se incrementen considerablemente. En este sentido, dado que los *testers* no siempre cuentan con todos los detalles sobre las modificaciones del código, especialmente en desarrollos que son subcontratados, se sugiere optar por los métodos bajo los cuales se realiza la TCP, los cuales a menudo se dividen en tres categorías:

- Métodos que se basan en el análisis del código fuente, los cuales priorizan los casos de prueba considerando su vínculo con las partes modificadas del código fuente.
- Métodos que se basan en el historial de pruebas, los cuales utilizan los datos históricos de las pruebas previamente realizadas para determinar la prioridad de un caso de prueba en particular.
- Métodos que se basan en el contenido de los casos de prueba, los cuales examinan los casos de prueba mediante técnicas de NLP para evaluar similitudes y diferencias entre sí para después optar por la selección de los mejores casos de prueba que se acoplen o se apeguen mejor al sistema bajo prueba.

Considerando lo anterior, el objetivo de esta investigación fue mejorar la detección de errores en las pruebas de regresión mediante la creación de un método híbrido que combina los métodos que se basan en el historial de pruebas y el contenido de los casos de prueba para recomendar casos de prueba adicionales.

2.5.1.2. Descripción

En el contexto de la investigación, las pruebas de regresión se realizaron considerando un conjunto de casos de prueba escritos en el idioma japonés, los cuales sirvieron como guías para que los *testers* ejecutaran las pruebas, ya fuera de forma tradicional o manual. Además, se reconoció que el *tester* a cargo de la actividad conocía las funcionalidades que fueron actualizadas previamente y que se tenía conocimiento de los errores que fueron corregidos. Así, el *tester* realizó la selección de aquellos casos de prueba que parecían estar más relacionados con las modificaciones realizadas sobre el sistema bajo prueba. Para minimizar el riesgo de no detectar regresiones (i.e., fallos), se recomendó un esquema que permitiera agregar otros casos de prueba que fueran útiles para este propósito (véase Fig. 2.7).

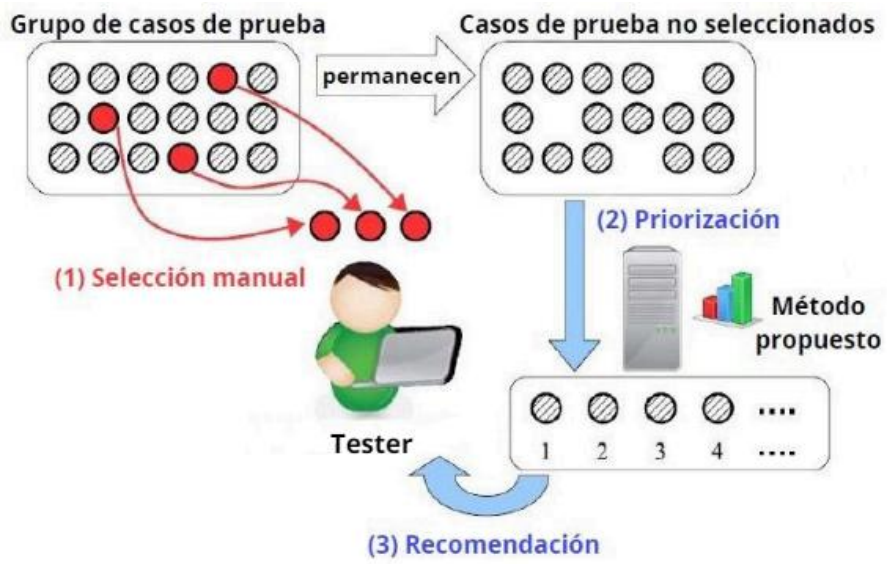


Figura 2.8. Esquema de recomendación de casos de prueba. Nota: Traducida de Aman et al., (2018)

Dado que posteriormente se decidió que las pruebas se realizaran de forma tradicional, el conjunto de recomendaciones fue lo más pequeño posible debido al costo asociado. Los datos que se utilizaron incluyen al historial de pruebas (ver Tabla 4) y el contenido de los casos de prueba (ver Fig. 2.8).

Tabla 4. Ejemplo de historial de pruebas. Nota: Traducida de Aman et al., (2018)

Caso de prueba	**Versión**				
	v1	*v2*	*v3*	*v4*	*v5*
t1	P	-	-	-	?
t2	F	P	P	F	✓
t3	-	P	-	-	?
t4	-	F	P	P	?
t5	P	-	-	-	✓
. . .	. . .	. . .	. . .	. . .	. . .

P: Aprobado, **F:** Fallido, **-:** No ejecutado, √**:** Seleccionado, **?:** Candidato

Pase el cursor del mouse sobre el ícono "?". Posteriormente, verifique si el mensaje de ayuda correspondiente se muestra en una ventana emergente.

Figura 2.9. Ejemplo de un caso de prueba. Nota: Traducida de Aman et al., (2018)

La Tabla 1 presenta un ejemplo del historial de pruebas con los símbolos "P", "F" y "-" que indican "aprobado", "fallido" y "no ejecutado", respectivamente, para cada versión del caso de prueba. Por ejemplo, el caso de prueba *t2* pasó en la versión *v2* y falló en la *v4*, mientras que el *t3* no fue ejecutado para la *v1*. Por otro lado, la versión *v5* es aquella sobre la que se realizaría una prueba de

regresión. El símbolo "✓" indicaría que los casos de prueba correspondientes habrían sido seleccionados por el *tester*; mientras que los casos de prueba restantes, que se muestran con el símbolo "?", serían los candidatos para la recomendación. Por lo tanto, el objetivo consistió en recomendar casos de prueba que fueran útiles a partir del conjunto de candidatos. Así, los investigadores argumentaron que, para obtener una mejor recomendación, se debían utilizar los casos de prueba con mayor prioridad puesto que son los que manejan la probabilidad más alta para la detección, lo cual implicaría el detectar más fallos con menos casos de prueba. Esto es lo que se define formalmente como "*el problema de recomendación de los casos de prueba*". Dicho problema establece que, dado un conjunto de casos de prueba denotado como *T*, el cual se encuentra conformado por *n*-casos de prueba $T = \{t_i\}_{i=1}^{n}$. Por ejemplo, un *tester* realiza la selección de *m*-casos de prueba $t_{1,j}$ de T (para $j = 1,\ldots,m$); sea $H = \{t_{1,j} | t_{1,j} \in T\}_{j=1}^{m}$, enseguida realiza la recomendación de los casos de prueba restantes después de la selección ordenándolos de mayor a menor en función de su prioridad (i.e., denotados como una secuencia) $(t_{2,r})_{r=1,\ldots,n-m}$ donde $(t_{2,r} \in T \wedge t_{2,r} \in H)$ y $t_{2,r}$ cuenta con una prioridad más alta que $t_{2,r'}$ sí $r < r'$ (para $r = 1,\ldots,n-m$). Por lo tanto, para determinar qué tan útil era una recomendación se analizó el área bajo la curva del diagrama de Alberg[9] (AUCA, por sus siglas en inglés). Así, un valor AUCA alto implicaría una mejor recomendación puesto que se entendería que más casos de prueba habrían logrado identificar fallos en una fase temprana de las pruebas de regresión. Por ejemplo, el mejor escenario se ilustra en la Figura 2.9 (a), donde se muestra que todos los fallos fueron identificados por los casos de prueba de mayor prioridad. Por lo contrario, si algunos de estos casos de prueba de alta prioridad no hubieran logrado detectar los fallos, el diagrama sería más parecido al mostrado en la Figura 2.9 (b). Se puede observar que el valor AUCA de la Figura 2.9 (b) es más bajo que el de la Figura 2.9 (a).

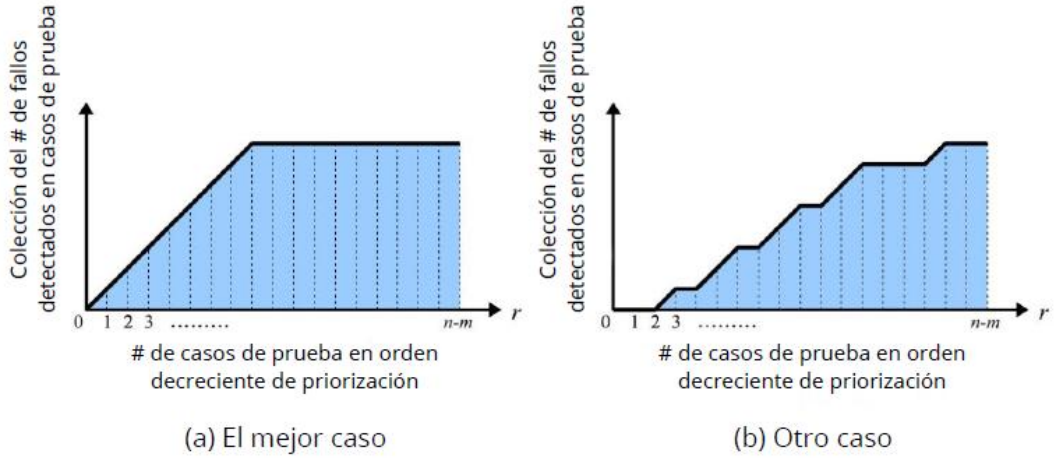


Figura 2.10. Ejemplos de diagramas de Alberg. Nota: Traducida de Aman et al., (2018)

Ahora bien, como se mencionó anteriormente, para realizar la recomendación (i.e., priorización) de los casos de prueba, los investigadores optaron por utilizar tanto el historial de pruebas como el

[9] De acuerdo con Runeson et al. (2001), un diagrama de Alberg facilita la evaluación de diferentes modelos de predicción de fallos puesto que ubica a los componentes de software en orden decreciente de acuerdo con el número de fallos, de tal manera que el número acumulado de fallos para diferentes porcentajes de tales componentes representa lo mejor que cualquiera de los modelos bajo evaluación podría lograr. Así, al comparar las curvas reales y las del modelo que se evalúa, es posible cuantificar la calidad de dicho modelo.

contenido de los casos de prueba. Para el caso de la priorización basada en el historial de pruebas, se consideraron especialmente dos casos:

- [*Tipo-1*] (SP1) Si un caso de prueba se ejecutó o no en una versión determinada.
- [*Tipo-2*] (SP2) Si un caso de prueba falló o no en una versión determinada.

Además, la aplicación de la *Media Móvil Ponderada Exponencialmente* (EWMA, por sus siglas en inglés) y el *Suavizado Exponencial* fueron utilizados con el objetivo de aprovechar sus características para la aplicación de un par de casos de prueba (t_i) y una versión (v_j), definiendo de esta forma la probabilidad de selección de casos de prueba como:

$$P(t_i, v_j) = \alpha * h_{ij} + (1 - \alpha)P(t_i, v_{j-1}) \quad (1)$$

donde h_{ij} pertenece al conjunto {0,1} que no es más que la observación del historial de pruebas para t_i y v_j; y α es la constante de suavización utilizada para ponderar las observaciones del historial de forma individual ($0 \leq \alpha \leq 1$): $P(t_i, v_0) = h_{i1}$. Por lo tanto, a mayor probabilidad de selección [$P(t_i, v_j)$] se determina una mayor priorización para el caso de prueba (t_i) en las pruebas de regresión de la versión posterior (v_{j+1}). Así, la propuesta de los investigadores se enfocó en utilizar la ecuación anterior para los casos [*Tipo-1*] y [*Tipo-2*]. Para el caso [*Tipo-1*], el valor de inicial de h_{ij} = 1 siempre y cuando el caso de prueba t_i no se halla ejecutado en la versión determinada por v_j; de lo contrario el valor de h_{ij} será igual a 0 ($h_{ij} = 0$). Tomando como referencia la ecuación (1) y un valor menor para α, como por ejemplo α = 0.1, se obtendría una probabilidad mayor ($P(t_i, v_j)$) si el caso de prueba t_i no se hubiera ejecutado en más versiones hasta el valor máximo especificado por v_j. Aplicando el mismo principio para el caso [*Tipo-2*], se tendría que h_{ij} = 1, si el caso de prueba t_i falló durante la ejecución en la versión indicada por v_j; de otra forma el valor de h_{ij} = 0. Utilizando la misma ecuación (1), pero ahora con un valor elevado para α, como por ejemplo α = 0.9, se determina que si el caso de prueba t_i falló de forma consecutiva durante todas las versiones hasta el límite dado por v_j, entonces la probabilidad $P(t_i, v_j)$ crecerá. Aunado a esto, la propuesta de los investigadores optó por la integración de los datos, provenientes de los dos casos mencionados con anterioridad, a través del método de *Mahalanobis-Taguchi* y, además, se obtuvo la información correspondiente mediante las medidas de *Diferencia entre la Última versión ejecutada y la Actual* (GLC, por sus siglas en inglés) y la *Tasa de Fallos* (FR, por sus siglas en inglés). Así, específicamente para un caso de prueba t_i, se tiene que GLC (t_i) denotaría el número de versiones consecuentes en donde el caso de prueba t_i no se hubiera ejecutado hasta la versión actual. Por otra parte, para el caso de prueba t_i, FR (t_i) denotaría la tasa de fallos para el t_i hasta llegar a la versión actual. En otras palabras, la tasa de fallos estaría determinada por el cociente entre el número de versiones fallidas para t_i y el número de versiones en las que se ejecutó t_i.

Tomando en consideración lo anterior, se analizó la posibilidad de que un caso de prueba que tuviera un valor alto para GLC y que además no se hubiera ejecutado para versiones posteriores a su última ejecución, pudiera introducir el riesgo mayor de omitir fallos. Es decir, a pesar de que habitualmente los *testers* seleccionan aquellos casos de prueba que manifiestan una relación con las funcionalidades que han sido actualizadas en su versión, algunas partes del código podrían contener fallos, por lo que valdría mejor la pena volver a ejecutar distintos casos de prueba que no tengan relación alguna con los cambios realizados. De manera similar, un caso de prueba con un valor FR

más elevado habría demostrado ser más efectivo para detectar fallos en el pasado. En este sentido, se consideró que este tipo de caso de prueba sería útil para evaluar una parte del producto que probablemente tuviera defectos, con el fin de asegurar una mayor probabilidad de encontrar nuevamente un fallo. Por lo tanto, sería preferible volver a ejecutar la mayor cantidad posible de casos de prueba. Además, se consideró que para ambas medidas un valor de 0 indicaría que el caso de prueba correspondiente tendría la menor probabilidad de detectar un fallo y que la distancia entre $x_i{}^T$ = (GLC (t_i), FR (t_i)) y 0^T = (0,0) podía expresar el valor de t_i para ser re-ejecutado en términos de estas medidas. De hecho, se pensó que la forma más sencilla de calcular esto sería utilizando la distancia Euclidiana que se encuentra determinada para este caso por:

$$d_E(x_i) = \sqrt{(x_i - 0)^T (x_i - 0)} \quad (2)$$

Sin embargo, se argumentó que la distancia Euclidiana no consideraba la dispersión de los datos. Es decir, dado que la métrica GLC abarca un rango más amplio que la FR, la GLC podría influir más en la distancia calculada. Por lo tanto, la Distancia de Mahalanobis (MD, por sus siglas en inglés) resultó ser la mejor opción para integrar de manera coherente los efectos de ambas medidas:

$$d_M(x_i) = \sqrt{(x_i - 0)^T \, S^{-1} (x_i - 0)} \quad (3)$$

donde S es la matriz de la covarianza para los valores de GLC y los valores de FR, y S^{-1} es la matriz inversa de la covarianza. Dado que la MD utiliza a la matriz de covarianza, no solamente sería capaz de considerar la dispersión en cada eje para una determinada medida, sino que también tomaría en cuenta la covarianza que existe en los datos, o mejor dicho las correlaciones que existen entre las medidas.

Por otro lado, los investigadores propusieron que para agrupar los casos de prueba basándose en su contenido y combinándolos con la MD con el fin de realizar las recomendaciones de casos de prueba, se siguieran dos pasos: (1) clasificar los casos de prueba en varios grupos de acuerdo con sus similitudes, basándose en palabras similares y (2) usar casos de prueba del mismo grupo que los casos seleccionados manualmente y utilizar el método de la MD, mencionado con anterioridad. Ahora bien, los investigadores plantearon la existencia de deficiencias en su enfoque, principalmente relacionadas con la existencia de palabras similares en los casos de prueba. Por ejemplo, cuando un *tester* selecciona un caso de prueba t_x y hay otro caso de prueba t_y similar pero no idéntico, este *tester* debería considerar ejecutar tanto t_x como t_y. Es decir, si desea probar dos funciones diferentes, A y B, bajo una condición determinada, t_x se ejecutaría en el orden (A, B) y t_y en el orden inverso (B, A). En este caso, los contenidos de t_x y t_y deben ser similares. Sin embargo, dado que no son idénticos, no se debe pasar por alto el omitir t_x ni tampoco t_y, es necesario ejecutar ambos. Sin embargo, algunos *testers* podrían ejecutar solo t_x y pasar por alto un fallo que podría haberse detectado con t_y. Por lo tanto, aunque el método explicado con anterioridad recomendó el uso de casos de prueba capaces de detectar fallos, también planteó un desafío: ¿cómo manejar la variación de palabras entre los casos de prueba?, dado que cada persona podría escribir frases diferentes utilizando distintas palabras para expresar un propósito idéntico o similar. En particular, el japonés, idioma en el que se escribieron los casos de prueba para realizar la evaluación empírica del enfoque, tiende a tener una alta vaguedad en las frases. Dado que inicialmente se consideró la similitud entre los casos de prueba mediante la

coincidencia exacta de palabras similares, esta vaguedad representaba una seria amenaza para la validez de la propuesta. Para resolver este problema, los investigadores propusieron la aplicación de un *modelo temático*[10]. En este modelo, las palabras se extraen de los documentos para analizar sus relaciones de concurrencia. A través de este análisis, cada palabra es asignada a uno o más temas que se consideran variables latentes (véase la Figura 2.10).

Este enfoque es similar al agrupamiento o *clustering* de palabras, pero no limita cada palabra a un solo clúster. Finalmente, cada documento se asocia con uno o varios temas. Aunque los temas no se mencionan explícitamente en un documento, las características de éste se expresan mediante un vector de características $(p_1, \ldots, p_k)$, donde p_i indica la probabilidad de que el documento trate el tema i-ésimo y k es el número de temas. El modelado de temas se realiza mediante la Asignación Latente de Dirichlet (LDA, por sus siglas en inglés), un método estadístico popular para estimar los temas latentes en los documentos.

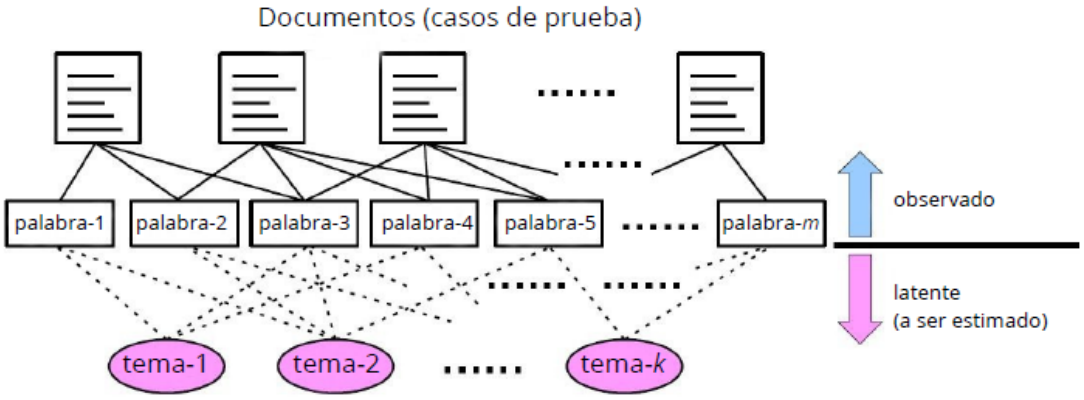


Figura 2.11. Ejemplo de temas latentes. Nota: Traducida de Aman et al., (2018)

Con este modelo temático se resolvió el problema de las palabras similares considerando ahora la similitud entre los casos de prueba y la semántica latente de las palabras. Dado que el vector de características $(p_1, \ldots, p_k)$ muestra los grados de pertenencia a los temas, la similitud semántica entre los casos de prueba se calculó utilizando estos vectores de características. Para esto, se definió la similitud entre los casos de prueba t_x y t_y como:

$$sim(t_x, t_y) = 1 - \frac{1}{k}\sum_{i=1}^{k}|p_{xi} - p_{yi}| \quad (4)$$

donde k es el número de temas, y p_{xi} y p_{yi} son las probabilidades de que t_x y t_y pertenezcan al tema i-ésimo, respectivamente. En este sentido, considerando la distancia de Manhattan se propuso un método para la recomendación de casos de prueba que constó de dos pasos:

- Paso 1. Suponiendo que el *tester* haya realizado una selección manual de m casos de prueba $\{t_1, i\} i = 1, \ldots, m$; la similitud de un caso de prueba restante t_x a partir del conjunto seleccionado se definió como:

[10] En estadística y NLP, un modelo temático es un tipo de modelo estadístico que se utiliza para descubrir "temas" abstractos que figuran en una colección de documentos. Así, el modelado de temas es una herramienta de extracción de textos de uso frecuente que es útil para descubrir estructuras semánticas ocultas en el cuerpo de un texto (Gui et al., 2019).

$$s(t_x) = \min_{i=1,\dots,m} [sim(t_x, t_1, i)] \quad (5)$$

Posteriormente se debe realizar la recomendación de los casos de prueba con altas similitudes a partir del conjunto seleccionado manualmente.

- Paso 2. Realizar la recomendación de los casos de prueba restantes a partir de cualquiera de los métodos SP1, SP2 o MD.

En este sentido, el primer paso se basa solo en la similitud del caso de prueba. Si un caso de prueba muy similar no es seleccionado, puede provocar una regresión excesiva. En el segundo paso, se utiliza el método convencional de priorización (i.e., SP1, SP2 o MD). Por lo tanto, esta propuesta es la principal contribución de los investigadores, ya que combina los métodos mencionados anteriormente. De esta manera, una vez que se haya seleccionado manualmente un subconjunto "semilla" de casos de prueba, el método propuesto prioriza automáticamente los casos de prueba restantes del grupo.

2.5.1.3. Resultados

Para examinar su propuesta, los investigadores obtuvieron datos de pruebas de regresión para un sistema basado en *web* que fue desarrollado y era mantenido por la corporación Toshiba. Así, se contó con 300 casos de prueba ($n = 300$) y un historial de pruebas para 13 versiones $v_{j=1}^{13}$del producto. Para el estudio realizado se ejecutaron manualmente los 300 casos de prueba en la última versión. Esto tomó 539 minutos y reveló que existían fallos que se pasaron por alto en las pruebas de regresión anteriores: 22 casos de prueba fallaron en la última versión. La supervisión ocurrió en la versión v_7, para la cual siete casos de prueba ($m = 7$) fueron seleccionados manualmente del grupo de casos de prueba, pero se debieron seleccionar más casos de prueba para detectar fallos anteriores.

Los resultados obtenidos fueron tomados a partir de la definición del número de temas ($k = 100$), posteriormente se realizó un análisis con los componentes primordiales obtenidos a partir del estudio empírico realizado. Considerando más del 90% de la información del espacio original, se decidió optar por 19 componentes del total de *k*. Por otra parte, se probaron distintos valores τ_x(x = 1, 2, 3, 4, 5) para definir el umbral de similitud denotado por la letra griega $\boldsymbol{\tau}$. Una vez que todos los parámetros necesarios fueron determinados, se realizó un recuento para concentrar la información mostrada en la Tabla 5.

Tabla 5. Valores AUCA obtenidos con la evaluación empírica. Nota: Traducida de Aman et al., (2018)

	Método basado en el historial		
	SP1	**SP2**	**MD**
Solamente historial	5,166	4,252	5,318
τ_1	**5,409** (+4.7%)	4,339 (+2.0%)	**5,386** (+1.3%)
τ_2	5,374 (+4.0%)	**4,390** (+3.2%)	5,380 (+1.2%)
τ_3	5,320 (+3.0%)	4,333 (+1.9%)	5,325 (+0.1%)
τ_4	5,265 (+1.9%)	4,295 (+1.0%)	5,269 (-0.9%)
τ_5	5,211 (+0.9%)	4,238 (-0.3%)	5,212 (-2.0%)

En la primer columna de esta tabla se muestran los valores de $\boldsymbol{\tau}$ para los valores del historial, el registro "Solamente historial" muestra los valores AUCA de las recomendaciones basadas en el historial de forma convencional, posteriormente en las tres columnas restantes se muestran los enfoques SP1, SP2 y MD, cada uno con los valores obtenidos a partir de cada $\boldsymbol{\tau}$ y enseguida se muestra la diferencia obtenida entre el valor aplicando el método basado en el historial (método propuesto) y el valor convencional dividido entre el valor convencional. El valor AUCA más alto está resaltado en la Tabla 2 para indicar que, independientemente del método basado en el historial, el método propuesto con τ=τ1 o τ2 muestra el mejor rendimiento al recomendar casos de prueba en nuestro conjunto de datos.

Para el estudio empírico el resultado obtenido a partir del método propuesto arrojó un valor de $\boldsymbol{\tau} = \boldsymbol{\tau_1}$ y $\boldsymbol{\tau} = \boldsymbol{\tau_2}$ como el mejor valor, lo que se traduce en que repetir la ejecución de los casos de prueba con mayor similitud en un 1% o 2% resulta favorable. En un caso ficticio donde se tenga un par de casos de prueba bastante similares, t_x y t_y, se podría dar el caso que el *tester* pase por alto a t_y dado que el resultado al ejecutar t_x es favorable y, erróneamente, se espera que t_y también lo sea. Sin embargo, se demostró que el valor AUCA aumenta al priorizar los casos de prueba con mayor similitud, por lo que resulta importante centrar los resultados en los casos de prueba similares. En este sentido, la Figura 2.11 (a) presenta los diagramas de Alberg que comparan al método convencional contra el propuesto, teniendo las 100 recomendaciones en el eje *x* y la $\boldsymbol{\tau} = \boldsymbol{\tau_1}$. De igual forma, la Figura 2.11 (b) muestra la misma información considerando al $\boldsymbol{\tau}$ que ahora es $\boldsymbol{\tau} = \boldsymbol{\tau_5}$.

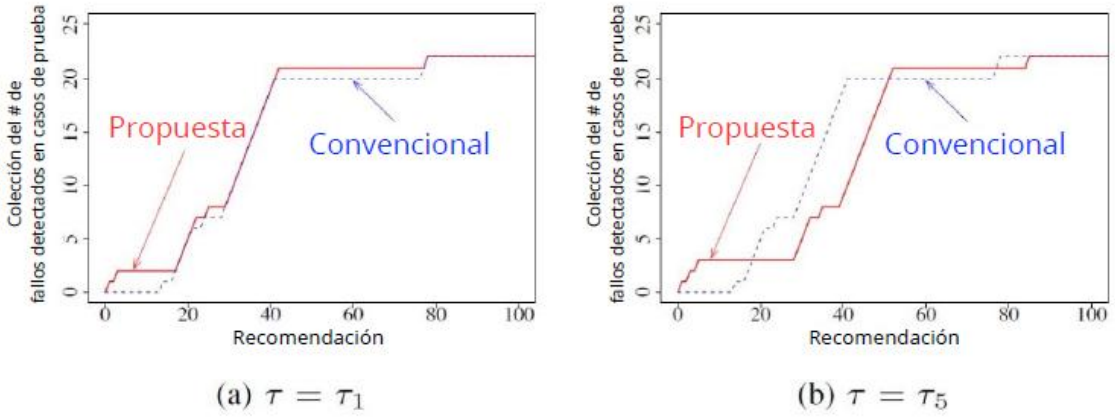


Figura 2.12. Esquema de recomendación de casos de prueba. Nota: Traducida de Aman et al., (2018)

En la gráfica se aprecia como, al inicio, los casos de prueba con mayor similitud fueron capaces de detectar de forma exitosa los fallos que los casos de prueba seleccionados de forma manual no detectaron. A diferencia del diagrama de Alberg de la Figura 2.11 (b), donde al recomendar aún más casos de prueba $\tau = \boldsymbol{\tau_5}$ se provoca un retraso en el crecimiento al momento de detectar los fallos.

A pesar de que en este estudio empírico las recomendaciones de casos de prueba con mayor similitud fueron efectivas para detectar regresiones, su éxito podría depender de las características del producto en general, del enfoque de las pruebas o incluso de los *testers* que llevan a cabo la/las prueba(s) de regresión.

2.5.2. Sistema de apoyo a la toma de decisiones para la elección de una estrategia de pruebas de regresión sobre el software

2.5.2.1. Objetivo

De acuerdo con Chernov et al. (2019), las actividades más relevantes para realizar un proceso eficiente de pruebas de regresión son: la elección de la estrategia óptima para conducir dichas pruebas, clasificar los enfoques de comprobación, investigar los factores que influyen en la elección del tipo de prueba, y mostrar cómo utilizarlos para tomar decisiones Sin embargo, debido a limitaciones de tiempo y recursos presentes en las organizaciones desarrolladoras de software, la elección de los métodos para realizar las pruebas de regresión más adecuados para cada proyecto se vuelve la actividad crucial. En este sentido, el objetivo de esta investigación fue desarrollar un algoritmo que facilitara la elección del tipo de prueba más adecuado y diseñar, además, un sistema de ayuda a la toma de decisiones, considerando indicadores de calidad del software ya especificados.

2.5.2.2. Descripción

La calidad del software está definida por un conjunto de características que determinan su capacidad para cumplir con necesidades específicas o anticipadas. De acuerdo con las principales fuentes especializadas de literatura, estas características pueden verse como un conjunto de propiedades (i.e., atributos) que se utilizan para evaluar y describir la calidad total del producto/servicio de software. En este sentido, se argumenta que un modelo de calidad del software consiste en un conjunto específico de características y relaciones que existen entre ellas, puesto que sirven como base para determinar los requisitos de calidad y realizar su posterior evaluación. El modelo de calidad del producto simplifica las propiedades de calidad de un software en ocho características: adecuación funcional, rendimiento, usabilidad, compatibilidad, seguridad, fiabilidad, portabilidad (movilidad) y soporte (véase Figura 2.12). Además, como se puede observar en la figura, cada una de estas características se desglosa en una serie de subcaracterísticas correspondientes.

Los investigadores consideran que algunos de los criterios de calidad mostrados en la figura son definidos al elaborar una especificación técnica, otros son evidentes, y algunos pueden ser establecidos por los *stakeholders*[11] durante el desarrollo de un producto/servicio de software. Sin embargo, en muchos casos, la *idoneidad funcionalidad* y la *fiabilidad* son criterios de calidad indispensables para un producto/servicio de software, y asegurar precisamente la *fiabilidad* es un interés constante a lo largo de todas las fases y procesos del desarrollo de software. Los demás criterios se aplican de acuerdo con los requisitos del producto/servicio y las necesidades de los usuarios. Es importante mencionar que todos estos indicadores de calidad pueden medirse o evaluarse mediante la realización de pruebas. Es decir, como se ha mencionado en repetidas ocasiones a lo largo de este capítulo, las pruebas son un método para evaluar la calidad del software puesto que permiten la identificación de defectos en los requisitos funcionales y no funcionales, así como en las características del software. Por lo tanto, al probar un producto/servicio de software, se puede asegurar que los proyectos cumplan con los estándares de calidad especificados para todo el ciclo de vida.

[11] De acuerdo con Robertson y Robertson (2012), los *stakeholders* son aquellos artefactos (e.g., libros, manuales, software heredado) y grupos o individuos (e.g., empleados, clientes, proveedores, accionistas, bancos, ambientalistas, gobierno, etc.) que están interesados en el software o que tienen conocimiento relacionado con éste y que pueden afectarlo directamente.

El objetivo principal de las pruebas es identificar errores en la implementación de los requisitos funcionales o bien, como se le reconoce regularmente, encontrar fallos en la ejecución de los programas y corregirlos lo más pronto posible con el fin de reducir los costos de desarrollo mediante la detección temprana de defectos. Sin embargo, como se ha mencionado también con anterioridad, en la práctica no es posible emplear todos los tipos de pruebas habidas y por haber dadas las restricciones de tiempo y costos que enfrentan un importante número de organizaciones desarrolladoras de software.

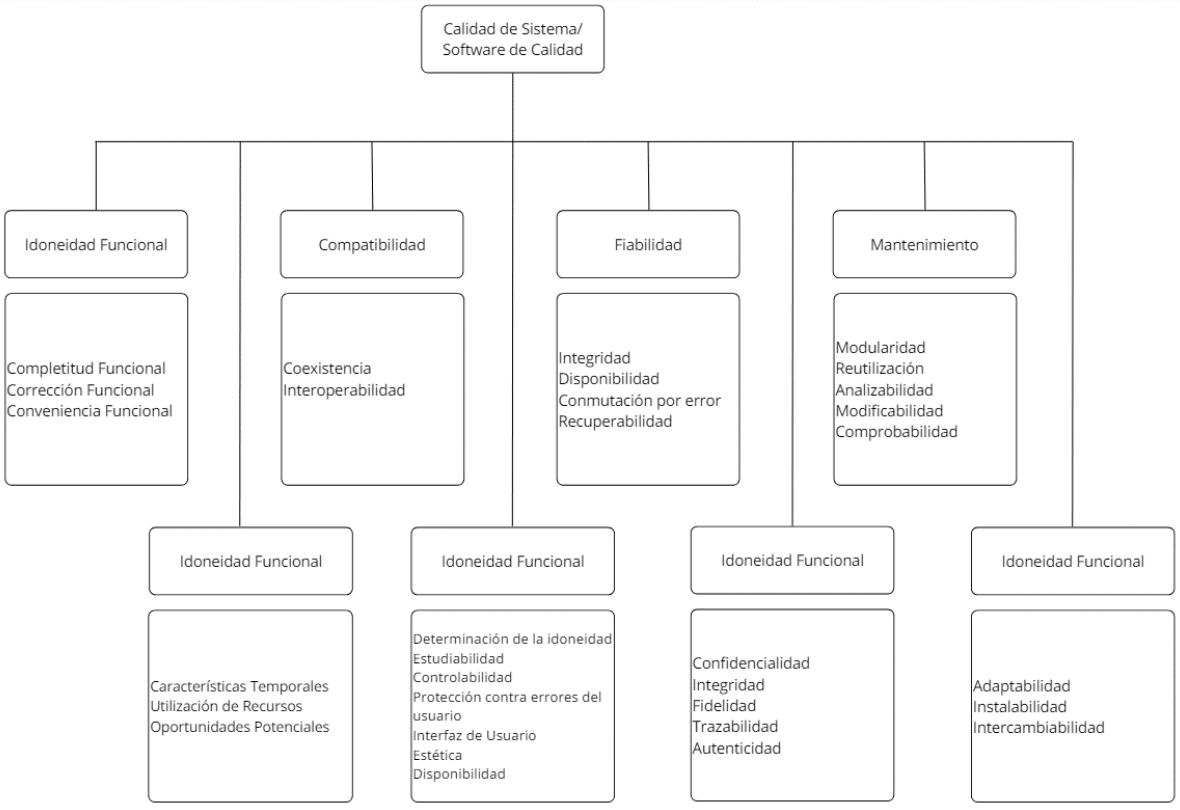


Figura 2.13. Modelo de calidad de los productos de software. Nota: Traducida de Chernov et al. (2019)

Así pues, la optimización del proceso de pruebas requiere del desarrollo de un plan que permita a los *testers* el elegir los métodos más apropiados para cada proyecto específico. Para abordar este desafío, la investigación identificó siete factores clave:

- Frecuencia de las versiones. El lanzamiento de versiones de un producto de software implica que se ha realizado un conjunto específico de cambios sobre éste, los cuales pueden estar relacionados ya sea con la corrección de errores detectados durante su uso o con la incorporación de funcionalidades adicionales solicitadas por el cliente. Es importante resaltar que estos cambios no deben alterar la filosofía general del producto para esa versión particular.
- Naturaleza del sistema. Los sistemas *web* y las aplicaciones para un solo usuario requieren de enfoques distintos para realizar las pruebas. En el caso de los sistemas *web*, es crucial enfocarse en las pruebas de carga y asegurar la disponibilidad continua del sistema. Por otro lado, para las aplicaciones de un solo usuario, la prioridad es evaluar el rendimiento y la estabilidad de dicha aplicación. Por lo tanto, es esencial utilizar métodos de prueba diferentes que correspondan con la naturaleza del sistema.

- Criticidad de los defectos. Un defecto es la consecuencia de un error en el software que puede, o no, causar un fallo. La criticidad de estos defectos se clasifica en tres niveles: alto, medio y bajo. Si los defectos en el sistema son críticos será necesario dedicar más tiempo a las pruebas, prestar atención a más detalles, realizar controles dobles, probar cada tarea y verificar la funcionalidad, fiabilidad y tolerancia a los fallos. Por lo contrario, si los defectos no son críticos será suficiente con probar las tareas individuales y realizar pruebas de regresión al final del proyecto.
- Complejidad del sistema. Es común que la etapa del ciclo de vida en la que se realizan las pruebas varíe de acuerdo con el modelo de desarrollo de software que se esté utilizando (e.g., cascada, en V, iterativo, espiral, modelo ágil). Por lo tanto, es importante considerar esto antes de elegir.
- Nivel y tipo de riesgo. El nivel de riesgo se define por la probabilidad de que ocurra un evento adverso y, además, por el impacto que éste tendrá en el proyecto. Además, los riesgos se emplean para decidir por dónde comenzar las pruebas y qué aspectos requieren mayor atención. Así, las pruebas ayudan a disminuir el riesgo de efectos adversos o mitigar sus consecuencias.
- Calidad. La elección de diferentes métodos de prueba dependerá del(os) criterio(s) de calidad que se pretenda evaluar (e.g., fiabilidad, seguridad, rendimiento).
- Complejidad del sistema. Esta complejidad se determina por el tamaño del código fuente y la complejidad matemática o algorítmica del mismo. Es decir, cuando un sistema realiza múltiples funciones y contiene numerosos módulos, resulta beneficioso dividirlo en tareas más pequeñas y aplicar un enfoque de prueba por tareas. Además, los criterios que influyen en el enfoque de las pruebas incluyen las regulaciones establecidas, los requisitos contractuales o del cliente, los objetivos de las pruebas, la disponibilidad de la documentación, la experiencia de los *testers*, el tiempo y el presupuesto disponibles, así como la experiencia previa en la detección de ciertos tipos de defectos.

Considerando estos factores, los investigadores propusieron un procedimiento para seleccionar el o los enfoques de pruebas más apropiados para un proyecto (véase Figura 2.13). De esta manera, los investigadores establecen que, dependiendo de la frecuencia de las actualizaciones realizadas sobre el código (i.e., semanal, mensual o anual), será necesario seleccionar enfoques diferentes y asignar tiempos distintos para las pruebas.

Además, argumentan que, si los lanzamientos ocurren semanalmente, será imposible realizar pruebas extensas y periódicas. Por ende, se indica que se debe tomar en cuenta la vasta experiencia que los *testers* han acumulado a lo largo de los años para realizar pruebas cada 1-3 días, seguidas de pruebas de regresión un día antes del lanzamiento. En el caso de lanzamientos mensuales, los investigadores consideran que es factible llevar a cabo pruebas cada 3-7 días antes del lanzamiento, con tiempo adicional para probar cada tarea individual. Finalmente, para lanzamientos anuales, los investigadores sugieren la realización de pruebas de regresión mensuales, junto con pruebas de funcionalidad y de tareas un mes antes del lanzamiento.

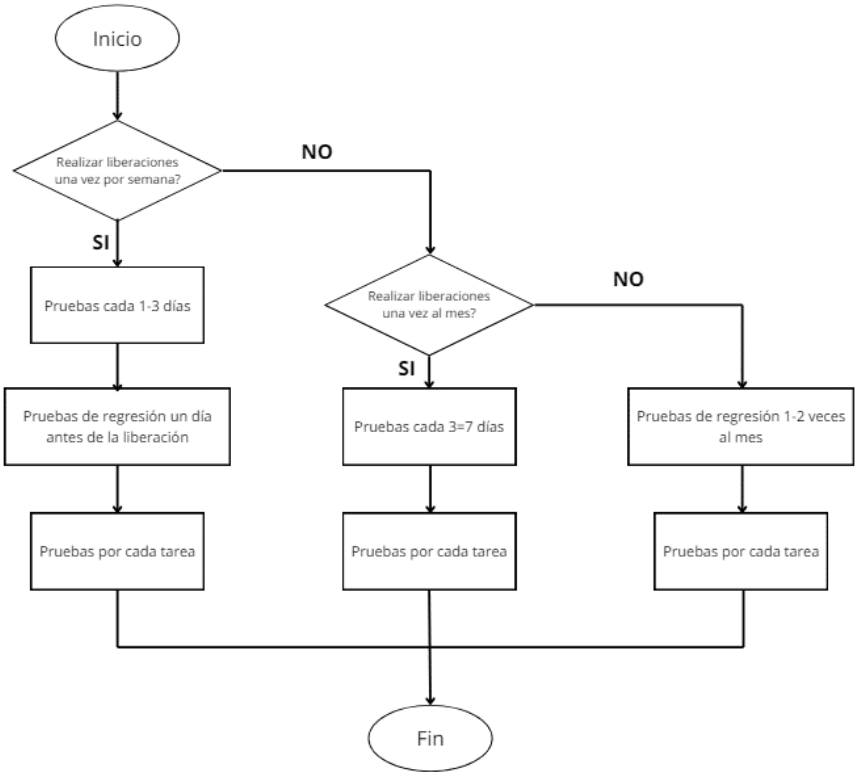


Figura 2.14. Algoritmo propuesto para elegir un enfoque de pruebas en función de la frecuencia de los lanzamientos. Nota: Traducida de Chernov et al. (2019)

Aunado a lo anterior, se sugiere que, para el caso de las aplicaciones móviles, se realicen pruebas en un emulador y pruebas de regresión en un dispositivo físico. En el caso de una aplicación *web* será necesario evaluar la carga, la funcionalidad y la usabilidad. Si se trata de un servicio local se deberán realizar pruebas de funcionalidad, tolerancia a fallos y carga. Para un servicio *web* también es crucial probar la interfaz de interacción. En el caso de una aplicación local se deberán realizar pruebas para verificar la funcionalidad de los algoritmos y validar el código. Para el caso de una aplicación LAN multiusuario, las pruebas deberán incluir evaluaciones de carga, fiabilidad y funcionalidad. Como resultado de estos razonamientos, la Figura 2.14 presenta el algoritmo creado por los investigadores para facilitar la elección de un enfoque de pruebas considerando la naturaleza del sistema.

Por otro lado, los investigadores también establecieron un proceso para facilitar la elección de los métodos de prueba que se basó en los aspectos de calidad que se identificaron como clave para el desarrollo de software. Este proceso estableció que, si se pretende evaluar el desempeño, será necesario realizar pruebas de carga, escalabilidad, volumen, estrés, competitividad y utilización de recursos. Al evaluar la compatibilidad, se deberán ejecutar pruebas de configuración y de compatibilidad entre distintos navegadores. Para evaluar la usabilidad, se deberán llevar a cabo pruebas de usabilidad, accesibilidad, interfaz, internacionalización y localización. Respecto a las pruebas de fiabilidad, será esencial verificar la tolerancia a fallos, la capacidad de recuperación y la fiabilidad del sistema. La seguridad deberá probarse mediante la realización de pruebas de seguridad; mientras que las pruebas de portabilidad deberán incluir a las pruebas de adaptabilidad, facilidad de instalación e intercambiabilidad. Una vez completada la evaluación de todos los aspectos de calidad pertinentes, se deberá probar la funcionalidad del sistema (véase Figura 2.15).

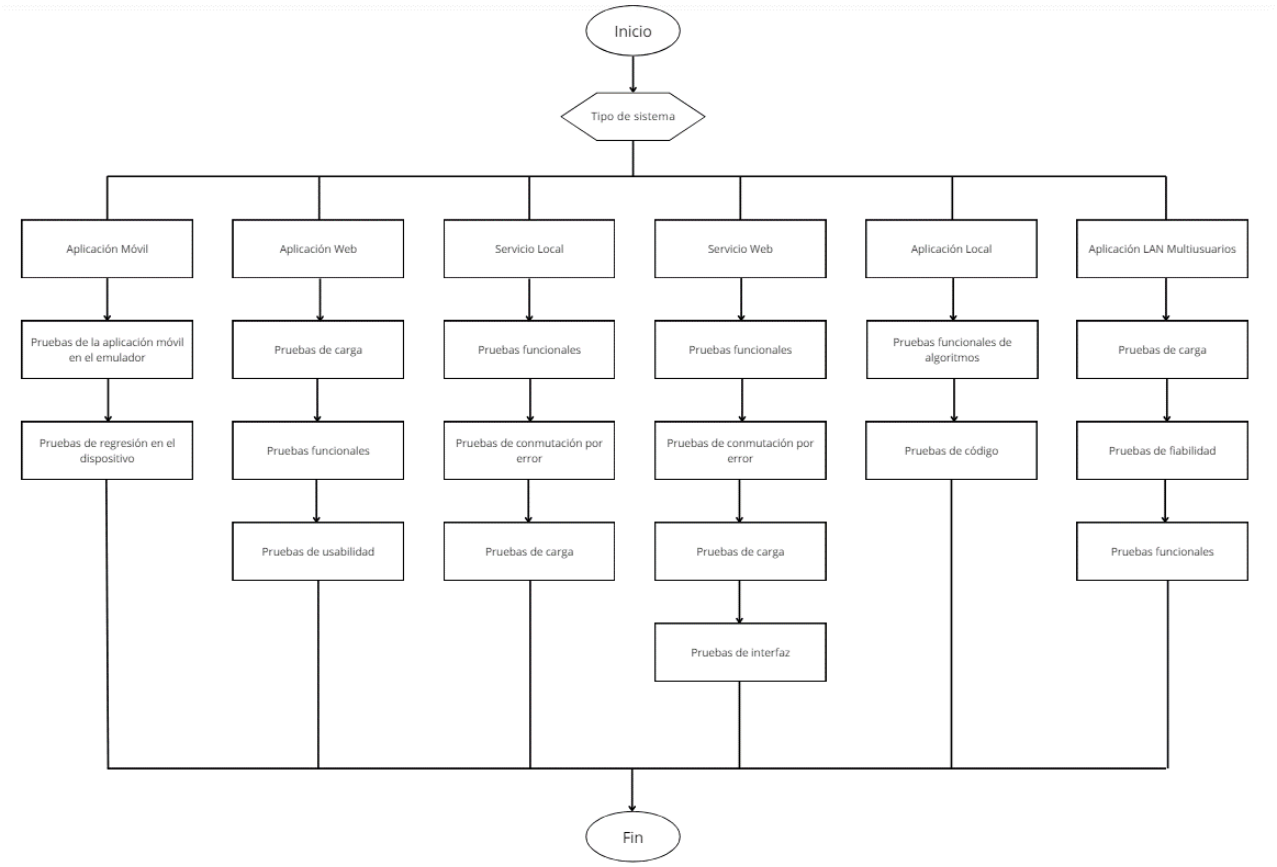


Figura 2.15. Algoritmo propuesto para elegir un enfoque de pruebas en función de la naturaleza del sistema.
Nota: Traducida de Chernov et al. (2019)

Considerando estos razonamientos, y los algoritmos que se fundamentaron en la lógica difusa y en metodologías de toma de decisiones multicriterio, los investigadores decidieron desarrollar una herramienta computacional que sirviera de apoyo a la toma de decisiones cuando se pretenda elegir los métodos de prueba más adecuados para un proyecto de software específico. El principal propósito que tenían los investigadores en mente al crear esta herramienta de apoyo fue apoyar a los *testers*, o a quien sea el responsable de tomar las decisiones con respecto al proceso de pruebas, a facilitar sus lecciones a través de la automatización del proceso de creación de una estrategia de pruebas de software. Las funcionalidades de esta herramienta incluyeron a la optimización del proceso de pruebas, la reducción del tiempo y los costos financieros asociados, y el establecimiento de una estrategia de pruebas basada en los criterios de calidad mostrados en la Figura 2.12. Desafortunadamente, la herramienta fue desarrollada como una aplicación de uso local para un solo usuario. Para funcionar correctamente, se requirió de la implementación de una base de datos que almacenara las recomendaciones de pruebas que corresponden directamente con los indicadores seleccionados por el usuario. Considerando los detalles técnicos de la implementación, los investigadores argumentan que la herramienta computacional fue programada en el lenguaje C *Sharp* y se utilizaron *Visual Studio* como entorno de desarrollo, SQL *Server* para la gestión de la base de datos, y *Crystal Reports* para la generación de informes y formularios. Las Figuras 2.16 y 2.17 ilustran algunos de los pasos del proceso de trabajo sobre fragmentos del programa. En cada etapa, el usuario deberá seleccionar un valor para el criterio y el resultado correspondiente se guardará en la base de datos.

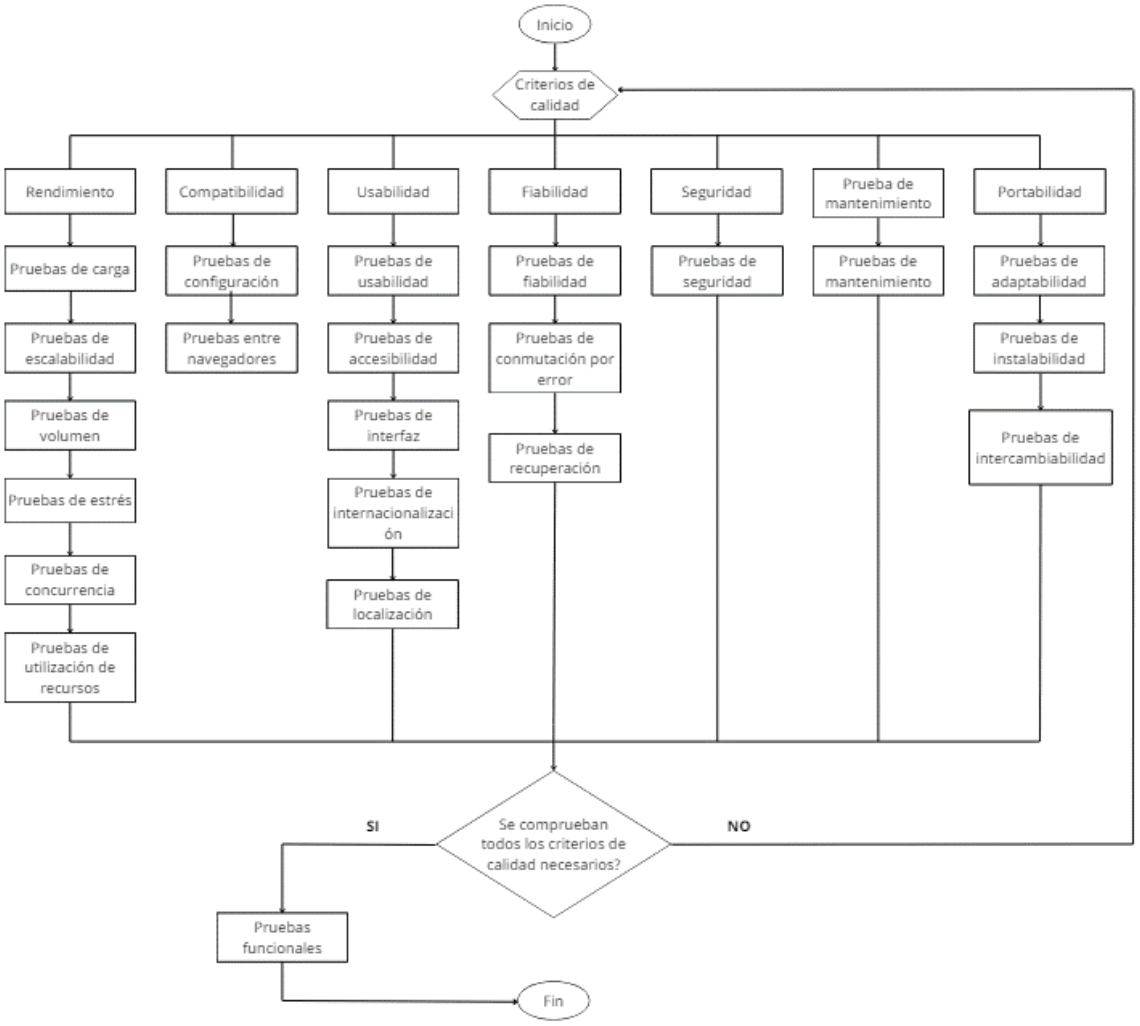
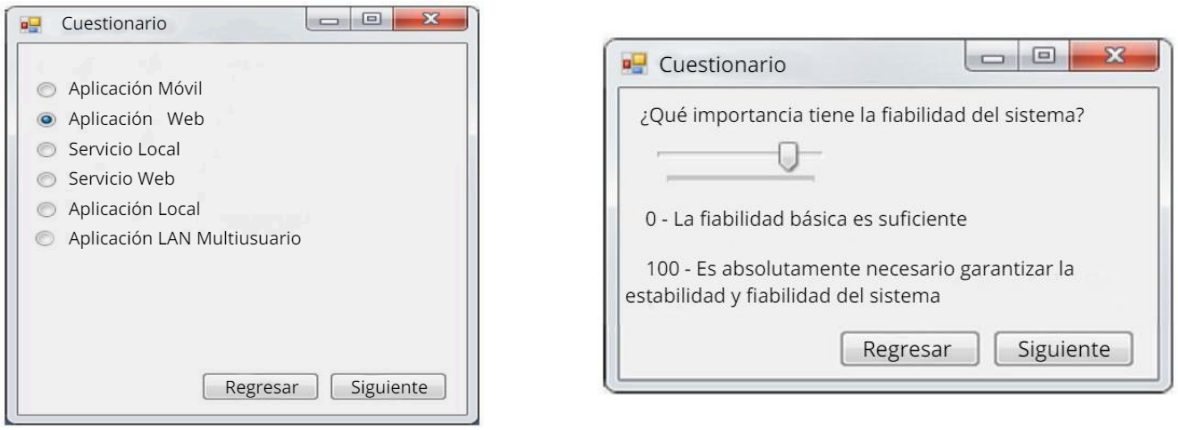


Figura 2.16. Algoritmo propuesto para elegir un enfoque de pruebas en función del factor de calidad. Nota: Traducida de Chernov et al. (2019)



(a) Elección del tipo de sistema (b) Evaluación del grado de importancia

Figura 2.17. Ejemplos de la interfaz de la herramienta computacional. Nota: Traducida de Chernov et al. (2019)

Una vez que el responsable de la toma de decisiones ha completado todos los pasos, la herramienta computacional generará y presentará las recomendaciones de pruebas apropiadas para el componente que se esté probando desde la base de datos (véase Figura 2.17). De acuerdo con los investigadores, la herramienta computacional creada genera información que las organizaciones pueden utilizar para definir y mejorar, a largo plazo, un proceso de pruebas que involucra la realización periódica de pruebas de regresión. Desafortunadamente, se argumenta que la herramienta se encuentra aún bajo evaluación para determinar su utilidad real.

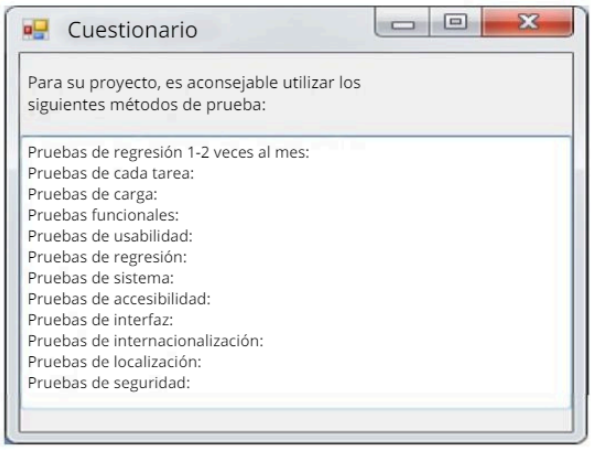


Figura 2.18. Resultado generado por la herramienta computacional. Nota: Traducida de Chernov et al. (2019)

2.5.2.3. Resultados

Como se indicó, los investigadores no proporcionan datos de una evaluación empírica sobre la herramienta, por lo que, en el contexto de esta tesis, es imposible determinar la efectividad de la propuesta. Considerando esto, los principales resultados de la investigación se limitan al análisis realizado para determinar los métodos actuales de prueba, la revisión de modelos e indicadores de calidad del software, y la identificación de los factores que pueden afectar la elección de los métodos de prueba para cada producto de software en particular. Aunado a esto, una contribución significativa de la investigación está representada por los algoritmos que fueron diseñados para facilitar la selección de una estrategia de pruebas basada en los indicadores de calidad definidos. Además, se creó una herramienta computacional de apoyo a la toma de decisiones con la finalidad de facilitar la identificación de la mejor opción para el proceso de pruebas, optimizando así tanto el tiempo como también los costos económicos de las pruebas.

2.5.3. Mejora del procedimiento de respuesta a consultas en pruebas de regresión mediante un esquema de recorrido de grafos optimizado y seguro

2.5.3.1. Objetivo

A lo largo de este capítulo se ha argumentado que la realización de pruebas sobre el software no es una tarea sencilla debido al enorme volumen de datos que complican la situación. En este sentido, la investigación realizada por Sivaji y Rao (2021) propuso el modelo llamado Réplica Optimizada del Recorrido de Hormiga León en Grafos Hash (OHGW-ALR, por sus siglas en inglés) para programar los casos de prueba de acuerdo con su prioridad y, como consecuencia, predecir los

fallos. Este enfoque organiza los casos de prueba minimizando el uso de los recursos y el tiempo de procesamiento.

2.5.3.2. Descripción

Con el objetivo de implementar un esquema de inspección que verifique las aplicaciones digitales que son creadas en función de su comportamiento, es necesario que las pruebas de regresión se repliquen en varias plataformas siguiendo un flujo de seis etapas concretas:

1. Selección, minimización o priorización basada en los casos de prueba.
2. Configuración de la prueba.
3. Ordenamiento de los casos de prueba.
4. Ejecución de los casos de prueba.
5. Evaluación del progreso.
6. Mitigación de fallos.

El factor crucial en los dispositivos digitales es la integridad de los datos, por lo que se utilizan modelos de réplica de sumas de comprobación para procesar con éxito dicha integridad. Durante el análisis de los casos de prueba, la programación basada en prioridades es esencial para facilitar las pruebas. Es decir, como se ha dicho anteriormente, el objetivo de las pruebas de regresión es detectar errores de software o de codificación en etapas tempranas para mejorar las funciones del sistema. Por lo tanto, el esquema de pruebas basado en la regresión ha sido efectivo con diversos tipos de modelos de pruebas, incluidas las pruebas de caja. Así pues, existen dos tipos de réplicas de pruebas: por prioridad y visión. La estrategia basada en la prioridad se enfoca en los casos más importantes y en los casos basados en el período, mientras que el enfoque basado en la visión incluye a los tipos de software, las versiones, etc. Considerando lo anterior, con la investigación se pretendió diseñar un modelo eficiente para mejorar la precisión en la predicción de los fallos y reducir el tiempo de procesamiento. Los principales pasos desarrollados con este modelo son los siguientes:

- Primero, se entrenan *n* casos de prueba para el sistema.
- Se desarrolla el OHGW-ALR para avanzar en las pruebas de regresión, ordenando los casos de prueba según su prioridad.
- Una vez recopilados los casos de prueba, se almacenan en la nube.
- Las estadísticas guardadas se aseguran mediante un enfoque binario.
- Se predicen los fallos presentes, donde la presencia de la “aptitud física” de la hormiga león mejora la precisión de la detección.
- Finalmente, la réplica desarrollada se evalúa en términos de la tasa de exactitud, la recuperación, la precisión, los casos de prueba procesados, etc., buscando la obtención de los mejores resultados.

El esquema de regresión es especialmente adecuado para diversos casos de prueba gracias a su función iterativa. Sin embargo, esta réplica dinámica requiere más tiempo para completar el trabajo. Cada módulo de software utiliza una plataforma de programación diferente, lo que hace que el código varíe según el marco específico del software. Esto genera una amplia gama de dificultades en las pruebas de regresión. Para abordar este problema, se diseñó una estrategia de perfiles horizontales que permite inspeccionar la ejecución del código y detectar fallos. Además, se adoptó un conjunto de datos

de referencia para realizar las pruebas. Este esquema de regresión es adaptable a todas las aplicaciones digitales y, para mejorar su eficacia, se desarrolló un sistema basado en prioridades con el fin de lograr una medida de error alta. Ahora bien, en el marco teórico se explicó que las pruebas de regresión se pueden realizar utilizando cuatro tipos de técnicas: minimización, selección, repetición y/o priorización de los casos de prueba. En este sentido, se consideró un funcionamiento básico para las pruebas de regresión (véase Figura 2.18).

En algunos casos, una prueba de regresión provoca un error de *bits*. Si este error es alto, la tasa de ejecución será muy baja y el tiempo de ejecución aumentará considerablemente. Por lo tanto, se propuso la creación de un esquema para las pruebas eficientes de regresión que se basó en la planificación por prioridades. Como consecuencia, el modelo OHGW-ALR facilita la planificación de los casos de prueba y reduce, al mismo tiempo, el índice de fallos (véase Figura 2.19). En primer lugar, los casos de prueba se protegen mediante un modelo de suma de comprobación *hash*[12]. Posteriormente, los casos son ordenados por prioridad para disminuir tanto el consumo de los recursos, como la congestión del tráfico.

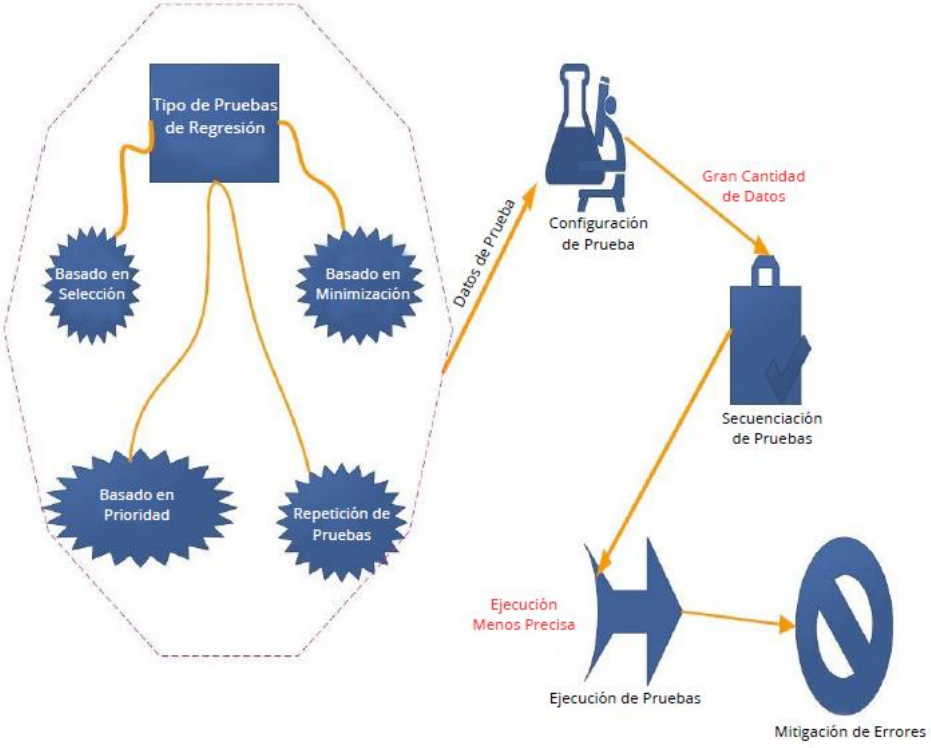


Figura 2.19. Modelo de sistema y definición de problemas. Nota: Traducida de Sivaji y Rao (2021)

[12] Un *"hash"* es el resultado de aplicar una función matemática que toma una entrada y la transforma en una cadena de caracteres, generalmente una representación alfanumérica de longitud fija de los datos de entrada (Levi y Sarimurat, 2017).

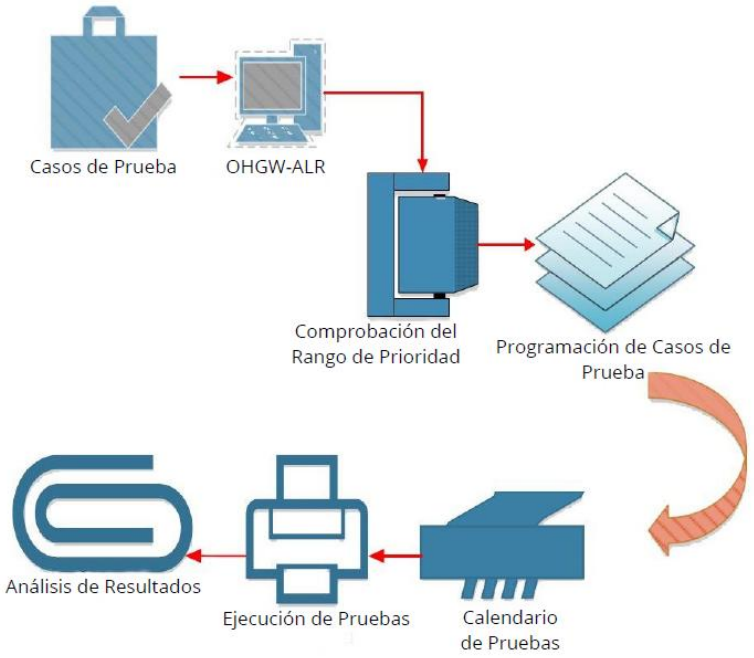


Figura 2.20. Modelo de OHGW-ALR. Nota: Traducida de Sivaji y Rao (2021)

Para la planificación de los casos de prueba se emplea el modelo de Hormiga León (AL, por sus siglas en inglés), el cual constituye un marco optimizado para organizar dichos casos. Asimismo, los casos de prueba se ejecutan siguiendo un orden de prioridad. La estimación del nodo de elevación se realiza utilizando la aptitud del recorrido aleatorio de AL mediante la ecuación (6).

$$n^{*g}_{h} = \frac{(n^{*g}_{h} - s)\, x\, (d - C^{g}_{h})}{(d - s)} \tag{6}$$

En este contexto, la jerarquización de los marcos de prueba se lleva a cabo de acuerdo con el intervalo de tiempo disponible, es decir, las fechas límite. Además, se asigna una duración específica requerida para cada caso de prueba. Durante la ejecución, si el tiempo necesario para completar un caso es menor que el intervalo medio, se considera de alta prioridad. Por lo tanto, la selección de los casos de prueba basada en la prioridad se realiza utilizando la ecuación (7).

$$caso\ de\ prueba = avg\ \ time > deadline \tag{7}$$

Por ejemplo, los seis nodos A1, A2, A3, A4, A5 y A6 de la Figura 2.20 representan seis casos de prueba distintos y la ejecución de cada caso dependerá de sus fechas límite, es decir 0.5, 0.6, 0.65, y 0.7 (línea roja). Asimismo, de acuerdo con la prioridad, el caso de prueba A1 se ejecutaría primero. Una vez que el fallo ha sido previsto, es necesario comprimirlo mediante la ecuación (8).

$$fault = \frac{f + f*}{2} \tag{8}$$

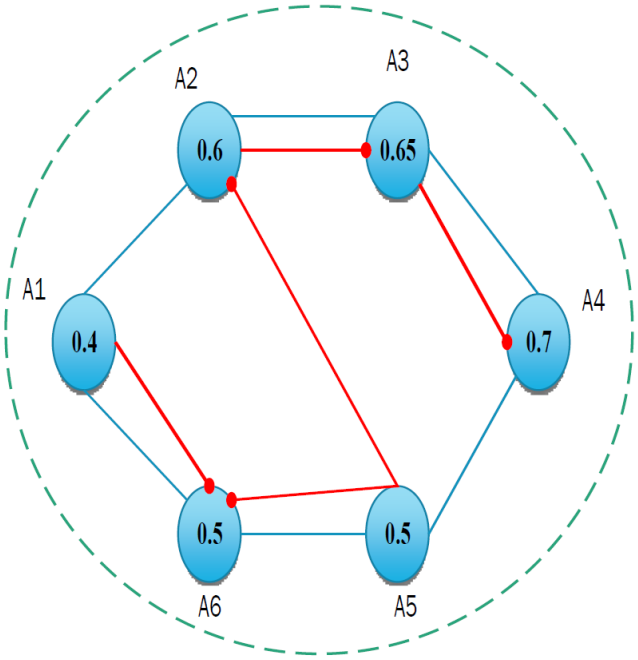


Figura 2.21. Seis distintos casos de prueba. Nota: Traducida de Sivaji y Rao (2021)

El sistema de funcionamiento de la réplica proyectada se describe con el siguiente algoritmo:

Algoritmo: pseudocódigo de OHGW-ALR

inicio

{

int: W, W, n**

//W es el grafo ponderado y W es el grafo no ponderado, n* es el centro actual en medio de la red*

int i, j, k

//el parámetro del caso de prueba está determinado por i, j, k

//i es el parámetro de mitigación de fallos

//j es la selección del caso de prueba basada en la prioridad

//k son los casos de prueba recopilados

//determinación del parámetro del análisis de regresión

k = número de casos de prueba

//todos los casos de prueba recopilados se planifican utilizando j

SumaDeComprobación( )

k = SumaDeComprobación

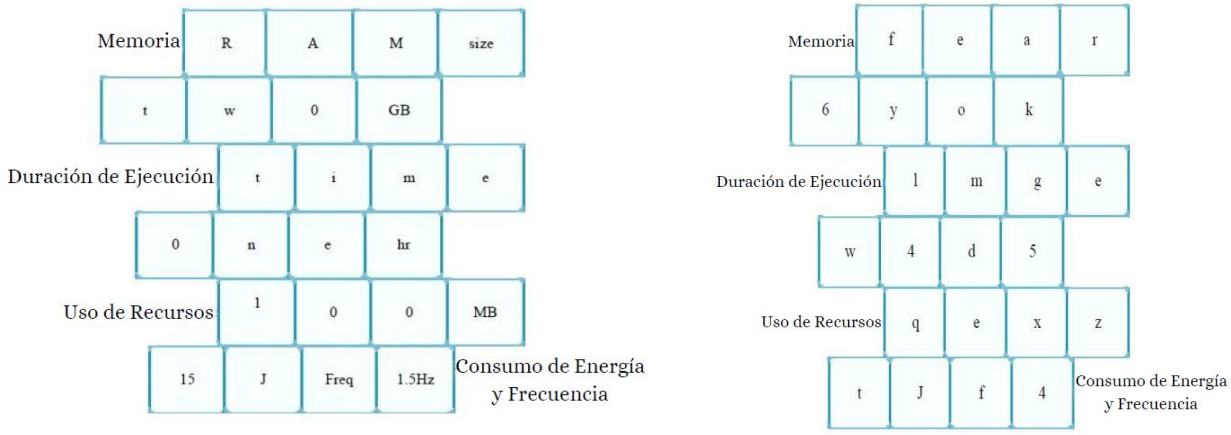
j = casos de prueba de corta duración

//Se actualiza el comportamiento de AL para seleccionar el caso principal de prueba
planificación de casos = plazo corto

EjecuciónDeLaPrueba( )
int t
//Ejecución del caso de prueba t

DetecciónDeFallos( )
i = fallo registrado
Registro optimizado de fallos utilizando la ecuación (8)
}
fin

De esta manera, el modelo basado en la regresión puede evaluar la eficiencia del sistema mediante la observación de sus comportamientos y considerar estadísticas que reflejen la capacidad de la memoria y la velocidad, entre otros aspectos. Para lograrlo, es esencial asegurar el conjunto de datos durante el proceso de verificación. Por lo tanto, una vez que se recopilan los casos de prueba de una plataforma de software específica, se protegen mediante una réplica de una suma particular de comprobación, conocida también como esquema binario (véase Figura 2.21 (a)).



(a) Casos de prueba recopilados (b) Casos de prueba después del *hashing*

Figura 2.22. Proceso de *hashing* sobre los casos de prueba. Nota: Traducida de Sivaji y Rao (2021)

De ahora en adelante, la información recopilada de la prueba es protegida mediante diversos enfoques de *hash*. Además, después de la función de suma de comprobación, los datos recopilados se modifican en forma de código que se expone en la Figura 2.21 (b). Por lo tanto, cuando el evento malicioso se localiza en el medio de la red mientras se realiza el análisis de regresión, no es posible registrar las estadísticas. La Figura 2.22 muestra la estrategia que sigue el modelo diseñado.

Figura 2.23. Modelo de flujo de la estrategia prevista. Nota: Traducida de Sivaji y Rao (2021)

2.5.3.3. Resultados

Para analizar el funcionamiento del sistema, es crucial replicar eficazmente los casos de prueba basados en la regresión. Por ello, la investigación propuso diseñar un nuevo modelo denominado OHGW-ALR para examinar y detectar posibles fallos. El principal objetivo de este modelo de casos de prueba es mejorar el sistema específico al identificar sus defectos en etapas tempranas. La efectividad del enfoque desarrollado se compara con otros estudios relevantes utilizando varios parámetros. Así, las métricas principales se validan con otros modelos asociados como el Modelo Basado en la Cobertura (CBM, por sus siglas en inglés), el modelo Adicional de Llamada Codiciosa (AGC, por sus siglas en inglés) y el Perfil Horizontal (HP, por sus siglas en inglés).

En este sentido, el propósito de desarrollar un modelo de casos de prueba basado en la regresión es anticipar los fallos en un software específico en una etapa temprana. Si se identifica un fallo en una fase inicial, se implementa un esquema de reducción adecuado para disminuir la tasa de fallos. En este caso, se realizó una evaluación que procesó hasta 55 casos de prueba. Al mismo tiempo, el modelo HP ejecutó 44 casos de prueba, AGC gestionó 40 muestras, y el enfoque CBM procesó 39 casos. La evaluación estadística obtenida se presenta en la Figura 2.23 y en la Tabla 6.

De acuerdo con los argumentos de los investigadores, la *precisión en la detección de los fallos* ha optimizado el desempeño del modelo. Es decir, la técnica que presentó una menor exactitud mostró

el peor rendimiento. De acuerdo con las estadísticas de evaluación, el modelo desarrollado logró una exactitud del 99% en la detección de los fallos, que es la tasa máxima alcanzada por este diseño.

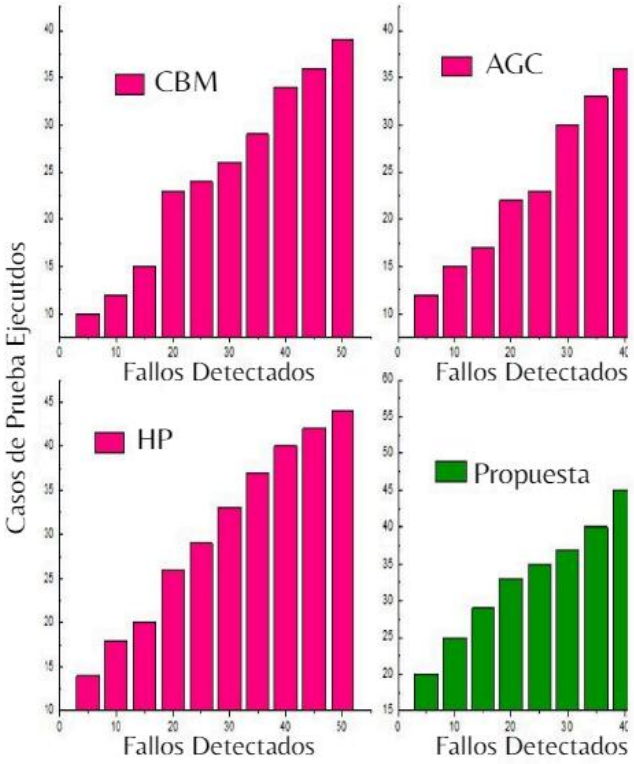


Figura 2.24. Tasa de errores detectados con casos de prueba ejecutados. Nota: Traducida de Sivaji y Rao (2021)

Tabla 6. Ejecución de pruebas contra detección de fallos. Nota: Traducida de Sivaji y Rao (2021)

Detección de fallos	**Ejecución de casos de prueba**			
	CBM	**AGC**	**HP**	**Propuesta**
5	10	12	14	20
10	12	15	18	25
15	15	17	20	29
20	23	22	26	33
25	24	23	29	35
30	26	30	33	37
35	29	33	37	40
40	34	36	40	45
45	36	38	42	50
50	39	40	44	55

Las estadísticas de exactitud están detalladas en la Tabla 7 y en la Figura 2.24. Además, la exactitud se calcula utilizando la ecuación (9).

$$Exactitud = \frac{detección\ exacta}{predicción\ total} \quad (9)$$

Tabla 7. Evaluación de la exactitud. Nota: Traducida de Sivaji y Rao (2021)

Detección de fallos	Exactitud			
	CBM	AGC	HP	Propuesta
5	93.8	95	96	99
10	93.3	94.7	95.5	98.9
15	93	94.2	95	98.6
20	92.8	93.8	94.8	98.3
25	92.5	93.3	94.3	98
30	92	93	94	97.9
35	91.6	92.8	93.6	97.6
40	91	92.5	93.3	97.3
45	90.6	92	93	97
50	90.3	91.6	92.6	96.8

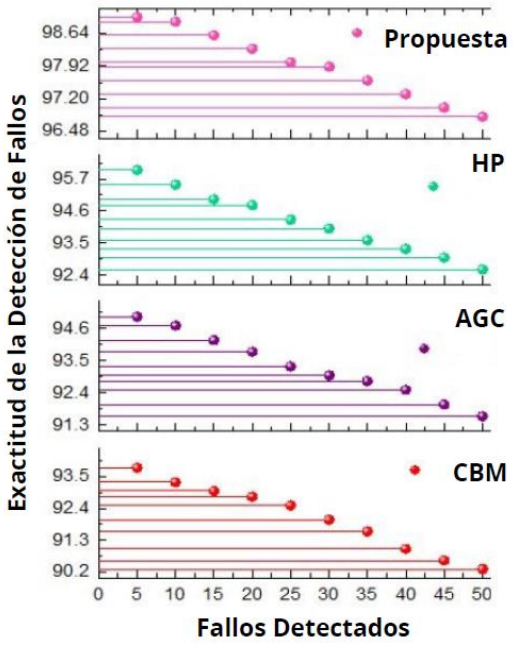


Figura 2.25. Comparación de medidas de exactitud. Nota: Traducida de Sivaji y Rao (2021)

Por otro lado, la medida utilizada para respaldar la estimación de la exactitud es la precisión. En este sentido, una réplica que logra una medida exacta superior muestra un alto índice de precisión. El avance de la tasa de precisión se presenta en la Tabla 4 y en la Figura 2.25. La *medida de precisión* se calcula utilizando la ecuación (10).

$$Precisión = \frac{detección\ correcta}{detección\ correcta + detección\ falsa} \quad (10)$$

Tabla 8. Evaluación de la precisión. Nota: Traducida de Sivaji y Rao (2021)

Detección de fallos	Precisión			
	CBM	AGC	HP	Propuesta
5	93.7	95	95.9	98.9
10	93.1	94.6	95	98.91
15	92.9	94.3	94.9	98.5
20	92.5	93.4	94.6	98.31
25	92.4	93.1	94.3	98.1
30	92.1	92.8	94.1	97.8
35	91.6	92.8	93.7	97.5
40	91.2	91.5	93.4	97.31
45	90.5	91	93.1	97.1
50	90.3	90.6	92.5	96.8

En este contexto, el modelo propuesto alcanzó una precisión máxima del 98.9%. En contraste, el modelo HP registró una medida de precisión del 95.9%, el método AGC obtuvo una tasa de precisión del 95%, y el enfoque CBM alcanzó el 93.7%.

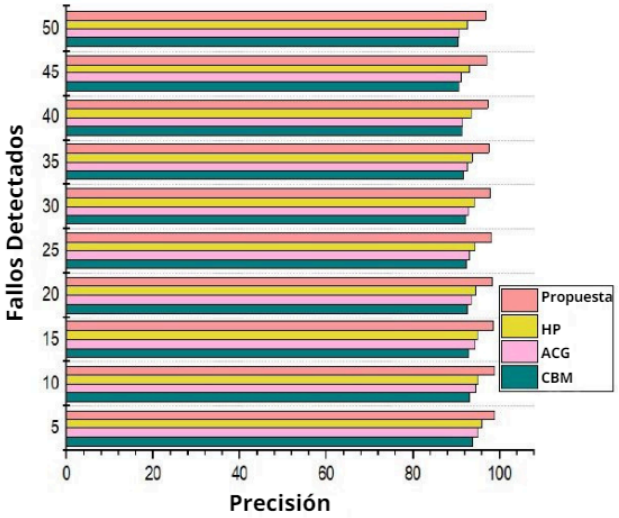


Figura 2.26. Evaluación de la precisión. Nota: Traducida de Sivaji y Rao (2021)

La predicción exacta y la falsa se utilizaron para evaluar *la recuperación del modelo*, y mediante esta evaluación, se determinó que la medida de sensibilidad obtenida fue casi idéntica a la medida de exactitud, lo cual, de acuerdo con los investigadores, confirmó la eficacia del modelo desarrollado. Para este caso, se logró la mayor recuperación con un 98.9%; mientras que el modelo HP alcanzó el 95.9%, el AGC un 95.0%, y el enfoque CBM un 93.6%. La evaluación de las estadísticas recogidas se detalla en la Tabla 9 y en la Figura 2.26.

Tabla 9. Medida de recuperación. Nota: Traducida de Sivaji y Rao (2021)

Detección de fallos	Recuperación			
	CBM	AGC	HP	Propuesta
5	93.6	95	95.9	98.9
10	93.1	94.6	95	98.91
15	92.6	94.32	94.91	98.51
20	92.5	93.4	94.61	98.32
25	92.4	93.2	94.3	98.2
30	92.1	92.7	94.11	97.8
35	91.5	92.5	93.7	97.51
40	91.3	91.5	93.41	97.31
45	90.5	91.1	93.1	97.1
50	90.3	90.6	92.5	96.8

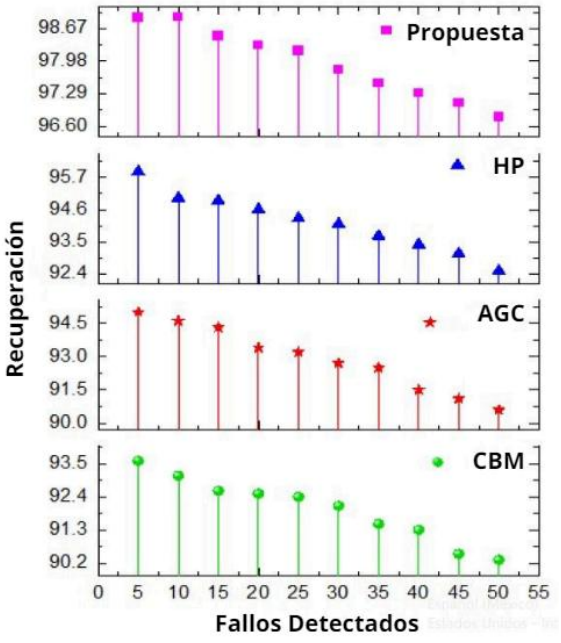


Figura 2.27. Evaluación de la recuperación. Nota: Traducida de Sivaji y Rao (2021)

Por último, el mantener un amplio margen de seguridad en el entorno de ejecución regularmente conlleva a alcanzar una alta tasa de confidencialidad. En este sentido, "confidencial" implica mantener el registro estadístico en el más alto grado de secreto. Por lo tanto, para asegurar los

casos de prueba que han sido almacenados se emplea cualquier tipo de réplica binaria de suma de comprobación al almacenarlos. La Tabla 10 y la Figura 2.27 muestra los valores obtenidos con la evaluación de la *medida de confidencialidad*.

Tabla 10. Medida de recuerdo. Nota: Traducida de Sivaji y Rao (2021)

Detección de fallos	CBM	AGC	HP	Propuesta
5	96.5	84.5	94.6	99
10	96.2	84.2	94.3	98.5
15	96	84	94	98
20	96.9	83.9	93.9	97.5
25	96.5	83.6	93.6	97
30	96.1	83.2	93.3	96.8
35	95	83	93	96.5
40	94.9	82.9	92.8	96.2
45	93	81	91	96
50	92	80	90	96.9

Como se puede observar en la Figura 2.27, la mayor medida de confidencialidad alcanzada por el modelo propuesto fue del 99%. Por otro lado, el modelo HP logró una medida de 94.6%, el AGC de 84.5%, y el enfoque CBM de 96.5%.

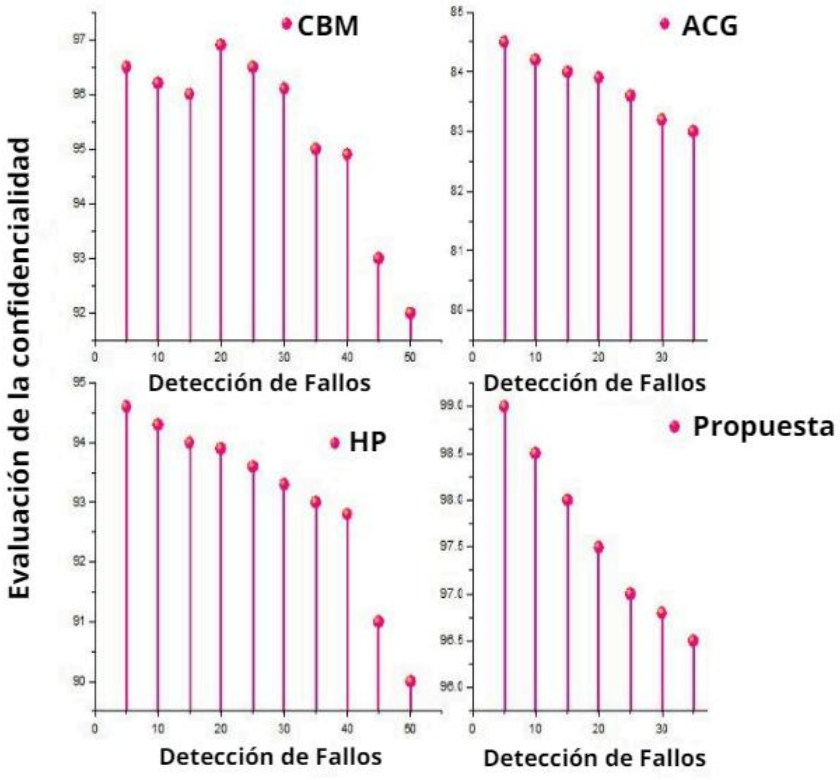


Figura 2.28. Evaluación de la confidencialidad. Nota: Traducida de Sivaji y Rao (2021)

2.5.4. Listas de comprobación para facilitar la toma de decisiones en las pruebas de regresión

2.5.4.1. Objetivo

A lo largo de este capítulo se ha argumentado que los profesionales que se dedican a la creación de software a gran escala se enfrentan a numerosos retos durante la realización de las pruebas de regresión. Una de las posibles causas de estos desafíos es la ausencia de un proceso que permita la correcta realización de estas pruebas. En este sentido, de acuerdo con la investigación de Minhas et al. (2023), las listas de comprobación pueden ser útiles para que los profesionales monitoricen las actividades importantes de las pruebas de regresión, proporcionando así un cierto grado de estructura al proceso. Por lo tanto, el objetivo de la investigación tuvo como finalidad la de proporcionar listas de comprobación sobre las pruebas de regresión para (1) determinar si los líderes y equipos de pruebas están preparados para iniciar las pruebas de regresión y (2) supervisar las actividades esenciales de las pruebas de regresión durante su planificación y ejecución.

2.5.4.2. Descripción

Como ya se ha mencionado, las pruebas de regresión pueden llevarse a cabo de dos formas: *volver a probar todo*, lo que implica ejecutar todos los casos de prueba en el conjunto de regresión, o *probar selectivamente*, que se enfoca en ejecutar solamente un subconjunto específico de casos de prueba. Sin embargo, una preocupación importante es el tamaño del conjunto de pruebas. En este sentido, en el desarrollo de software a gran escala, los profesionales preferir ejecutar Pruebas de Regresión con un Alcance Específico (SRT, por sus siglas en inglés). El principal reto radica en determinar el alcance de estas pruebas, es decir, decidir qué pruebas incluir en el conjunto de pruebas de regresión. Las técnicas empleadas para las SRT incluyen a las mencionadas a lo largo de este documento: selección de los casos de prueba, la priorización de los casos de prueba, y la minimización del conjunto de pruebas.

Por otro lado, los profesionales de diversas disciplinas emplean listas de comprobación como una herramienta cognitiva para asegurar la correcta ejecución de cualquier tarea. Una lista de comprobación es una herramienta estandarizada que detalla los criterios del proceso que son necesarios para que los profesionales lleven a cabo una actividad específica puesto que, además, les permite registrar la presencia o ausencia de tareas esenciales. Las listas de comprobación son comúnmente utilizadas como sistemas mnemotécnicos o como herramientas de evaluación. Como *sistemas mnemotécnicos*, sirven como recordatorios para que los profesionales no omitan tareas esenciales, asegurando que sigan el marco organizacional y apliquen las mejores prácticas. Estas listas ayudan a reducir errores humanos y a mejorar el rendimiento general. Por otro lado, las *listas de comprobación evaluativa* contribuyen a la estandarización de las evaluaciones al ofrecer pautas claras, lo que mejora la credibilidad del proceso evaluativo. Los ingenieros de software pueden emplear diferentes listas de comprobación a lo largo del ciclo de vida del desarrollo de software, como, por ejemplo, en la auditoría de las especificaciones de los requisitos y el diseño o en la inspección de códigos. Inicialmente, una lista de comprobación no necesita ser exhaustiva puesto que se pueden añadir elementos adicionales con el tiempo y, con su uso prolongado, garantizan que los procesos sean repetibles. En el contexto de la investigación, los autores realizaron una revisión sistemática de literatura con el fin de ayudar a los profesionales al proporcionarles información sobre la estructuración del proceso de pruebas de regresión a través de la implementación de listas de comprobación específicas. Se adoptó, por lo tanto, un enfoque iterativo para diseñar, desarrollar y evaluar estas listas de comprobación y, además, responder las siguientes tres Preguntas de Investigación (RQ, por sus siglas en inglés):

- RQ1: ¿Qué actividades consideran los profesionales al planificar, realizar y evaluar las pruebas de regresión?
- RQ2: ¿Qué listas de comprobación y elementos de estas listas pueden ser útiles para los profesionales al planificar, realizar y evaluar las pruebas de regresión?
- RQ3: ¿Cuál es la perspectiva de los profesionales sobre las listas de comprobación propuestas?

La Figura 2.28 muestra el enfoque que se empleó para la creación y desarrollo de las listas de comprobación específicas para las pruebas de regresión. De acuerdo con los investigadores, dicho enfoque incluyó a profesionales en actividades que van desde la identificación de las actividades hasta la validación de las listas de comprobación. Aunado a esto, se argumenta que se realizaron entrevistas individuales y grupales con expertos en el área de pruebas para determinar las actividades esenciales que se relacionan con las pruebas de regresión. Posteriormente, estas actividades fueron relacionadas con elementos cruciales para la toma de decisiones antes y después de las pruebas, como se ilustra en la Figura 2.29. Este proceso sirvió como punto de partida para la creación de las listas de comprobación que fueron presentadas a los profesionales como borradores iniciales con el fin de recibir comentarios de su evaluación. Finalmente, se argumenta que expertos de la industria evaluaron las listas propuestas en términos de alcance, utilidad y relevancia para su entorno específico.

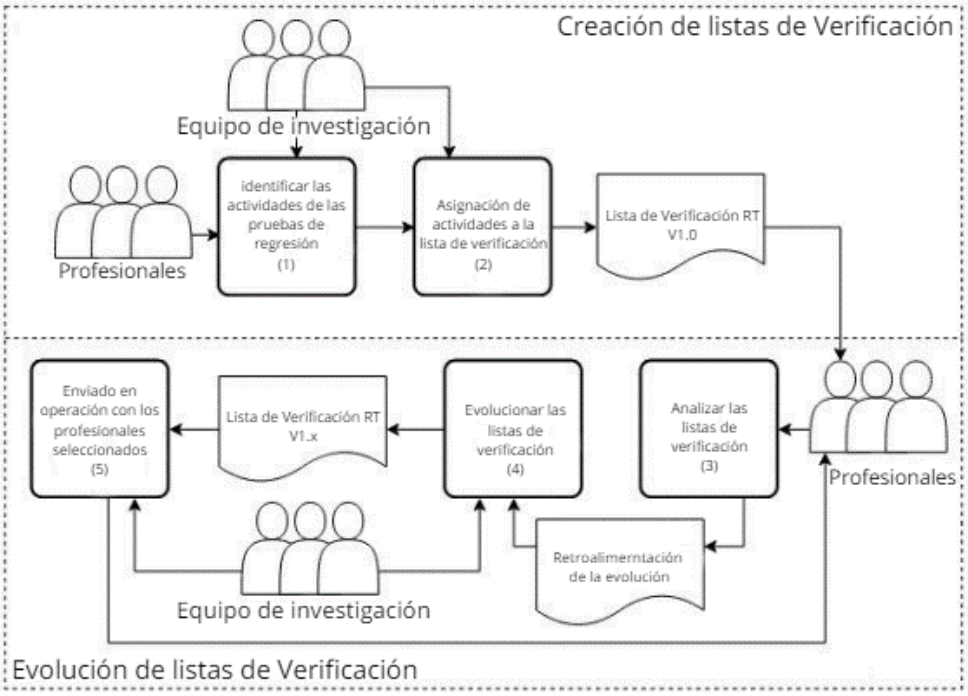


Figura 2.29. Descripción general del enfoque utilizado para diseñar y desarrollar las listas de comprobación. Nota: Traducida de Minhas et al., (2023)

Es importante mencionar que se adoptó un método de *muestreo de bola de nieve* para elegir a los participantes del estudio, quienes fueron profesionales de alto nivel con al menos cinco años de experiencia en las pruebas de software. Inicialmente se estableció comunicación con siete expertos de tres grandes organizaciones suecas, dos de los cuales ya habían colaborado en estudios previos sobre las pruebas de regresión. Cinco de ellos respondieron positivamente y participaron en talleres introductorios para comprender el alcance y los detalles operativos del estudio.

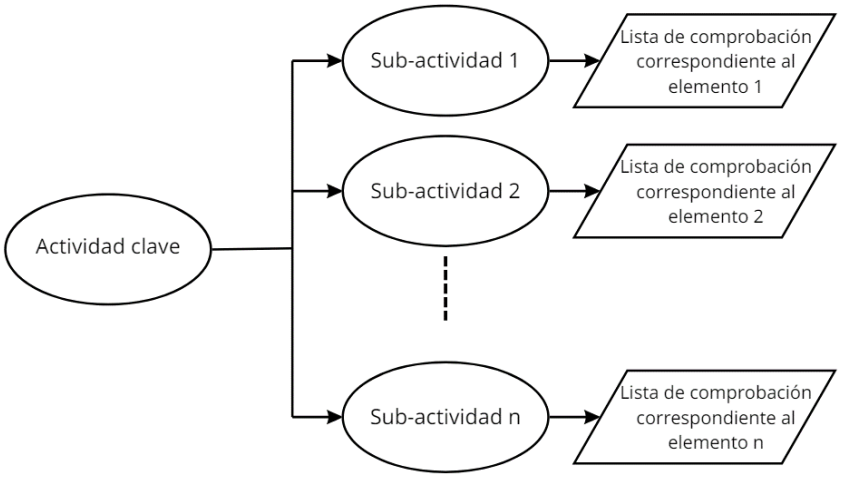


Figura 2.30. Mapeo de actividades de pruebas de regresión con elementos de la lista de comprobación. Nota: Traducida de Minhas et al., (2023)

Cabe resaltar que la búsqueda se extendió a través de *LinkedIn* y se organizaron talleres introductorios para los interesados, sin importar el número de participantes potenciales. Al final se logró el consentimiento de 25 profesionales de pruebas de 12 empresas diferentes, ampliando así la muestra de manera efectiva. La experiencia de estos participantes varió entre cinco y 23 años, con una media de doce años, y la inmensa mayoría trabajaba en entornos de desarrollo de software a gran escala. De las organizaciones participantes solamente dos contaban con menos de 250 empleados, por lo que se consideran de tamaño grande. En estas organizaciones se utilizaban metodologías ágiles, incluyendo Scrum, CI/CD y DevOps. Finalmente, estas organizaciones desarrollan software para dominios como finanzas, banca, salud, transporte, vigilancia y seguridad, telecomunicaciones, soluciones de inteligencia artificial y sistemas de seguridad.

La recopilación de los datos se realizó con cuatro técnicas con el fin de obtener información sobre la realización de las pruebas de regresión, la elaboración de las listas de comprobación y su posterior validación:

- *Talleres introductorios*. Se condujeron 12 seminarios virtuales que, además de una breve presentación inicial, establecieron discusiones basadas en preguntas y respuestas. El propósito de estos encuentros fue asegurar la participación informada y voluntaria de los profesionales con experiencia en las pruebas de regresión.
- *Entrevistas* (creación de listas de verificación). Se realizaron siete entrevistas individuales y cinco entrevistas grupales para recopilar información sobre cómo se realizan las pruebas de regresión. Las entrevistas individuales tuvieron una duración promedio de 60 minutos, mientras que las grupales duraron unos 75 minutos en promedio. Las entrevistas siguieron un formato semiestructurado que incluyó preguntas abiertas que fueron adaptadas al contexto de los participantes. Durante estas sesiones se exploraron diversas áreas relacionadas con la práctica al realizar las pruebas de regresión en las organizaciones, así como las actividades consideradas esenciales y los criterios utilizados en su uso.
- *Talleres de trabajo* (mejora de las listas de comprobación). Se llevaron a cabo talleres en línea para mejorar las listas de comprobación. Durante estos talleres, los participantes evaluaron cada elemento de estas listas para indicar si eran relevantes o no en la práctica. También se les pidió que sugirieran nuevos elementos o proporcionaran comentarios adicionales que pudieran contribuir con la mejora.

- *Encuestas* (evaluación de las listas de comprobación). Después de terminar las listas de comprobación, se les pidió a los profesionales que ofrecieran comentarios basados en su experiencia con las listas y las discusiones del equipo. En este sentido, se utilizó un cuestionario de *Google Forms* para abordar aspectos como la facilidad de comprehensión, utilidad, personalización y adoptabilidad de las listas de comprobación. También se incluyó una pregunta sobre la disposición de las organizaciones participantes para emplear estas listas.

Aunado a esto, el estudio requirió de un análisis detallado sobre los datos cualitativos, mientras que se obtuvieron resúmenes y gráficos de los datos recopilados durante los talleres y las encuestas que propiciaron la mejora de las listas de comprobación. La Figura 2.30 resume los pasos llevados a cabo para realizar el análisis de los datos.

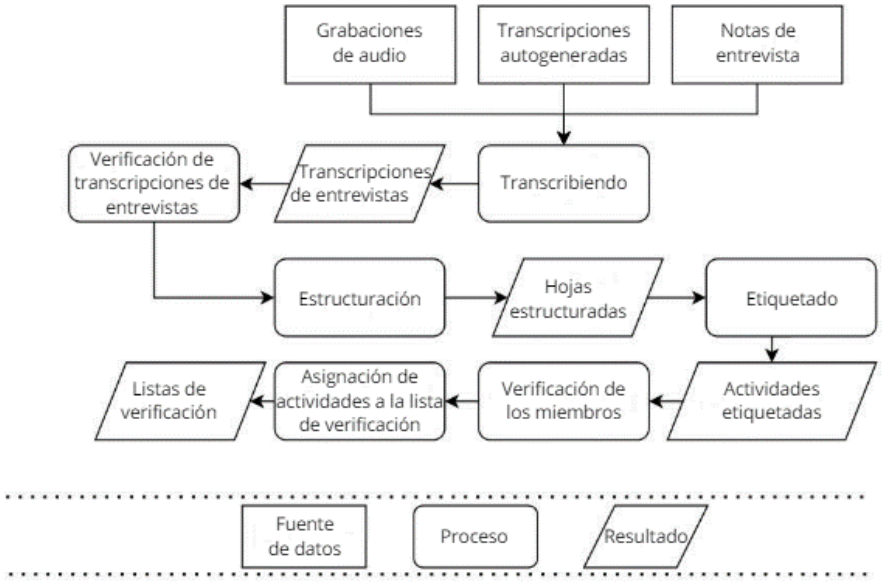


Figura 2.31. Pasos seguidos por el análisis de los datos. Nota: Traducida de Minhas et al., (2023)

Considerando esta figura, los datos recogidos fueron organizados en una hoja de cálculo de Excel con el fin de que fueran fáciles de recuperar. Sin embargo, se argumentó que no fue una tarea sencilla el encontrar la información deseada correspondiente con las RQ's establecidas. De acuerdo con los investigadores, a menudo fue necesario revisar las respuestas a varias preguntas para hallar la información pertinente. Tras transcribir los resultados de las entrevistas en un formato estructurado con códigos de colores adecuados, el siguiente paso fue asignar etiquetas apropiadas para temas similares. En este sentido, se utilizaron etiquetas para las actividades consideradas "esenciales" para las organizaciones antes y después de realizar las pruebas de regresión. Estas etiquetas fueron asignadas siguiendo la codificación axial para agrupar declaraciones similares. Por ejemplo, se agruparon las siguientes tres afirmaciones: *1. Seleccionar un subconjunto más pequeño, pero eficaz, de casos de prueba*, *2. Seleccionar los casos de prueba adecuados*, y *3. El conocimiento sólido de los casos de prueba ayudará a seleccionar los adecuados*, con el propósito de etiquetarlas como "*Selección de los casos de prueba correctos*". Después de esta clasificación, se les pidió a los participantes que confirmaran si la interpretación de sus respuestas había sido correcta. Los investigadores argumentaron que no recibieron ninguna corrección. Así, las actividades fueron divididas en subactividades, siempre que fue posible, para posteriormente transformar cada actividad en un elemento de la lista de comprobación (véase Figura 2.29). Por ejemplo, la actividad denominada

"Selección de casos de prueba correctos" se dividió en dos subactividades: (1) *Identificar los casos de prueba relacionados con los cambios* e (2) *Identificar los casos de prueba relacionados con los módulos afectados*. Después, estas actividades fueron asignadas a los siguientes elementos de la lista de comprobación:

- ¿Se han identificado los casos de prueba asociados con las partes cambiadas de código?
- ¿Se han identificado los casos de prueba asociados con el módulo afectado?

2.5.4.3. Resultados

La información recogida en las organizaciones permitió determinar que el panorama actual de las pruebas de regresión refleja su consideración como una práctica esencial por parte de la mayoría de los profesionales involucrados en su realización. En ciertas compañías se limita la ejecución de estas pruebas a cambios significativos, mientras que se recurre a pruebas exploratorias para modificaciones menores. Además, posterior a las pruebas de regresión, la mayoría de las organizaciones emplean pruebas exploratorias para asegurar el correcto funcionamiento de todas las áreas de riesgo. Por otro lado, una de las organizaciones participantes argumentó realizar *pruebas de humo*[13] antes de las pruebas de regresión con el fin de verificar la estabilidad del desarrollo. Cabe resaltar que la frecuencia de las pruebas de regresión varía entre organizaciones y se encuentra mayormente determinada por el dominio y la criticidad del producto o módulo sometido a prueba.

Otro hallazgo importante consistió en determinar que los profesionales participantes en el estudio fijan metas para las pruebas de regresión y, la mayoría de ellos, específicamente 10 de cada 12, lo hacen de manera informal. Solamente unos cuantos siguen una metodología claramente definida para establecer y estimar estos objetivos. De acuerdo con los testimonios recopilados de los participantes de varias organizaciones, el criterio de la experiencia es el principal factor para fundamentar la toma de decisiones. Por ejemplo, la determinación de cuándo finalizar las pruebas de regresión se basa en la opinión de expertos, siendo los líderes de equipo quienes, fundamentados en su experiencia y conocimiento, toman esta decisión. La información considerada para concluir las pruebas de regresión incluye a la ejecución del conjunto de pruebas planificado, la tasa de aprobación/rechazo, la comparación de la tasa de aprobación con un umbral predefinido (por ejemplo, si es del 90 % o más) y la gravedad de los errores encontrados.

Aunado a lo anterior, las organizaciones están cambiando de la realización manual y parcialmente automatizada de las pruebas de regresión a una automatización total. Algunas también han implementado canales de CI/CD y DevOps. El alcance de estas pruebas se determina de acuerdo con la relevancia de los cambios realizados y su impacto. Se ejecutan considerando un conjunto seleccionado de casos de prueba cada vez que se introduce un cambio, ya sea añadiendo una nueva característica o corrigiendo un error. Cerca del lanzamiento, es común ejecutar *suites* de regresión completas, generalmente automatizadas. Sin embargo, una de las organizaciones confirmó haber adoptado un enfoque diferente debido al alto costo de ejecutar todas las pruebas en sus extensos conjuntos de regresión. Para abordar este problema, argumentó estar experimentando con la ejecución de todas las pruebas sobre las funcionalidades más utilizadas, en lugar de correr todas las pruebas del conjunto de regresión, lo que han denominado "ejecutar todas las pruebas que realmente importan".

[13] De acuerdo con Sommerville (2021), las *pruebas de humo* son un tipo de pruebas funcionales que facilitan la revisión rápida de un producto de software para comprobar su funcionalidad básica con el fin de asegurar que no existen defectos evidentes que pudieran interrumpir su operación elemental.

Con relación a los hallazgos más puntuales, la información recogida para responder la RQ1: *¿Qué actividades consideran los profesionales al planificar, realizar y evaluar las pruebas de regresión?* estableció el fundamento para desarrollar las listas de comprobación destinadas a la definición de las pruebas de regresión (i.e., tareas implicadas en la planificación, ejecución y evaluación de estas pruebas). Se llevaron a cabo 12 entrevistas con 25 profesionales, quienes señalaron las actividades que consideraban fundamentales para este tipo de pruebas (véase Tabla 11).

Tabla 11. Actividades consideradas esenciales para las pruebas de regresión. Nota: Traducida de Minhas et al., (2023)

# de actividad	Actividades antes de las pruebas de regresión	Sugerido por las organizaciones
1	Adquirir conocimientos sobre el tema	C2, C6, C7, C8, C10, C11, C12
2	Garantizar la comunicación sobre los cambios	C1, C3, C5, C6, C12
3	Conocer las nuevas características/cambios	C3, C4, C5, C6, C7, C8, C9, C10, C12
4	Garantizar la realización de los cambios	C3, C6, C10, C11, C12
5	Asegurar que los cambios se hayan probado	C2, C4, C7, C9, C10, C12
6	Identificar el impacto de los cambios	C1, C2, C4, C5, C6, C7, C9, C10, C12
7	Establecer el alcance de las pruebas de regresión	C1, C2, C4, C8, C12
8	Seleccionar los casos de prueba adecuados	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12
9	Disponer el entorno apropiado de pruebas	C10, C11
10	Organizar los datos de prueba	C2, C6, C7, C8, C10, C11, C12
11	Definir los objetivos de las pruebas de regresión	C1, C2, C3, C4, C12
12	Elaborar el plan de pruebas de regresión	C7, C8, C10
13	Asignar responsabilidades	C2, C8
14	Dar mantenimiento al conjunto de pruebas	C1, C6, C7, C9, C11
# de actividad	**Actividades después de las pruebas de regresión**	**Sugerido por las organizaciones**
1	Ejecutar el conjunto de pruebas de regresión planificado	C1, C3, C5, C6, C8, C9, C10, C12
2	Crear informes de las pruebas	C2, C6, C7, C10, C11, C12
3	Analizar los resultados de las pruebas	C1, C2, C3, C5, C6, C7, C8, C9, C10, C11, C12
4	Evaluar el cumplimiento de los objetivos	C1, C3, C4, C5, C6, C7, C9, C11
5	Evaluar la proporción de aprobados y rechazados	C1, C4, C5, C6, C7, C9, C11, C12
6	Asegurar que los aprobados estén por encima del umbral	C2, C5, C6, C7, C9, C11, C12
7	Identificar y resolver los fallos críticos	C1, C2, C3, C4, C8, C10, C11, C12

Los profesionales destacaron la importancia de seleccionar los casos de prueba apropiados, enfatizando en lograr una cobertura amplia con la menor cantidad de casos posible. Esto, obviamente, requiere de un sólido conocimiento del dominio, comprensión de las especificaciones del sistema y seguimiento de los cambios y su impacto. Por lo tanto, para evaluar este impacto, es crucial entender las interdependencias entre los módulos o subsistemas. Además, es imperativo que todos los cambios

sean probados exhaustivamente y documentados antes de iniciar las pruebas de regresión. La disponibilidad del entorno de prueba y los datos pertinentes también son aspectos esenciales para iniciar estas pruebas. Posteriormente, los profesionales ejecutan las pruebas planificadas en su totalidad, generan informes, analizan los resultados y toman decisiones en función de éstos. Se evalúan los casos de prueba aprobados frente a los fallidos y se determina si el producto puede ser lanzado según un umbral predefinido. En algunos casos, se puede optar por lanzar el producto con errores de gravedad media, dejando la corrección para futuras versiones. Sin embargo, en casos de fallos graves, se debe posponer el lanzamiento. Muchos equipos establecen objetivos antes de iniciar las pruebas de regresión y posteriormente evalúan su cumplimiento al concluir la actividad.

La información recogida para responder la RQ2: *¿Qué listas de comprobación y elementos de estas listas pueden ser útiles para los profesionales al planificar, realizar y evaluar las pruebas de regresión?* le permitió a los investigadores obtener una comprensión más profunda de la visión de los expertos respecto a las listas de comprobación útiles en las pruebas de regresión, así como identificar las actividades clave requeridas para llevarlas a cabo. A partir de estos hallazgos, se asignaron las diferentes actividades de las pruebas de regresión a listas de comprobación individuales. La mayoría de los entrevistados expresó su convicción sobre la utilidad de estas listas, siempre y cuando se centraran únicamente en los aspectos esenciales. De hecho, destacaron que éstas podrían contribuir a establecer una estructura a la práctica de las pruebas de regresión y ayudar a los profesionales a seguir el plan con eficacia y no dejar de lado ningún paso importante durante su realización. El estudio determinó que, aunque algunos participantes no emplean listas formales, siguen de manera informal una serie de elementos esenciales. Por ejemplo, los líderes *senior* de pruebas evalúan la preparación de sus equipos a través de conversaciones informales. Por otro lado, la mayoría de los expertos propusieron listas para asistir a los profesionales tanto antes como después de las pruebas de regresión, y, además, sugirieron tres tipos de listas: una para evaluadores individuales, otra para actividades en equipo, y una más que combinara las dos anteriores. La Tabla 12 recopila las opiniones de los profesionales sobre los distintos tipos de listas de comprobación.

Tabla 12. Tipos de listas de comprobación sugeridas por los participantes. Nota: Traducida de Minhas et al., (2023)

Tipo de lista	**Sugerido por las organizaciones**
Listas de comprobación para realizar el seguimiento de las actividades antes de las pruebas de regresión (individual).	C1, C4, C6, C7, C9, C10, C12
Listas de comprobación para realizar el seguimiento de las actividades antes de las pruebas de regresión (equipo).	C1, C4, C6, C7, C9, C10, C12
Listas de comprobación para realizar el seguimiento de las actividades antes de las pruebas de regresión (combinadas).	C2, C3, C8, C11
Listas de comprobación para realizar un seguimiento de las actividades después de las pruebas de regresión (criterios de salida).	C1, C2, C3, C4, C7, C8, C11, C12

Para decidir qué elementos se incluirían en las listas de comprobación, los investigadores consideraron las actividades relacionadas con las pruebas de regresión identificadas por los profesionales (véase la Tabla 11). Posteriormente, para cada actividad se crearon elementos relevantes que se asignaron a las listas individuales de comprobación. Como resultado de este ejercicio, se crearon las listas CL 1.0, CL 2.0 y CL 3.0 que se presentan en las Tablas 13, 14 y 15, respectivamente. Así, para verificar la preparación de los integrantes del equipo, por ejemplo, un líder de pruebas puede pedirles que completen la lista de comprobación CL 1.0 proporcionada en la Tabla 13. Más adelante,

considerando los resultados obtenidos en CL 1.0, el líder de pruebas y los integrantes del equipo pueden completar la lista de comprobación CL 2.0 para evaluar la preparación del equipo. Finalmente, mientras se están concluyendo las pruebas de regresión, el líder de las pruebas y los integrantes del equipo pueden completar la lista de verificación CL 3.0. En cada lista de comprobación se han proporcionado dos columnas adicionales, "estado" y "comentarios". Usando la columna de estado, los *stakeholders* pueden informar el estado relacionado con el elemento de la lista de comprobación, mientras que en la columna de comentarios los stakeholders pueden proporcionar detalles relacionados con el estado informado. Por ejemplo, para el elemento "*¿Conoce las dependencias entre los subsistemas?*" de la Tabla 13, el interesado puede responder con "Sí", "Sí, pero no al 100%" o "No, no es aplicable". Por lo tanto, en la columna de comentarios tendría que proporcionar una explicación más detallada del estado. Si el estado informado es "Sí", entonces los *stakeholders* deberían reflexionar sobre las dependencias del sistema, y si el estado es "Sí, pero no al 100%", deberían de indicar los aspectos que hacen falta.

Tabla 13. Lista de comprobación para determinar la preparación que deben completar los *testers* del equipo de prueba (CL 1.0). Nota: Traducida de Minhas et al., (2023)

CLI	Elemento de la lista de comprobación	Estado	Comentarios
1	¿Conoce los objetivos de las pruebas de regresión del equipo?		
2	¿Tiene conocimientos esenciales sobre las especificaciones del sistema?		
3	¿Conoce las dependencias entre los subsistemas?		
4	¿Está usted al tanto de nuevos cambios en el sistema?		
5	¿Ha analizado el impacto de los cambios en las partes no modificadas del sistema?		
6	¿Está seguro de poder realizar las pruebas de regresión por su cuenta?		
7	¿Ha recibido capacitación sobre las herramientas utilizadas para las pruebas de regresión dentro del equipo/organización?		
8	¿Es usted consciente de la criticidad de los subsistemas a probar?		
9	¿Tiene acceso a los datos de las pruebas?		

Tabla 14. Lista de comprobación para determinar la preparación del equipo para ser completada por el líder de pruebas (CL 2.0). Nota: Traducida de Minhas et al., (2023)

CLI	Elemento de la lista de comprobación	Estado	Comentarios
1	¿Se han definido los objetivos de las pruebas de regresión?		
2	¿Los integrantes del equipo de pruebas conocen las especificaciones del sistema?		
3	¿Se han registrado todos los cambios?		
4	¿Se han comunicado los cambios al equipo de pruebas?		
5	¿Se han probado los cambios de forma aislada?		
6	¿Se ha determinado el impacto del cambio?		
7	¿Está actualizado el conjunto de pruebas de regresión?		
8	¿Se han identificado los casos de prueba asociados con piezas cambiadas?		
9	¿Se han identificado los casos de prueba asociados con el módulo afectado?		

CLI	Elemento de la lista de comprobación	Estado	Comentarios
10	¿Se ha determinado el alcance de las pruebas de regresión?		
11	¿Se han incorporado las pruebas de regresión al plan de pruebas?		
12	¿Se ha desarrollado el plan de prueba de regresión?		
13	¿Están disponibles los recursos necesarios?		
14	¿Se ha tomado la decisión entre pruebas de regresión manuales o automatizadas?		
15	¿Se han asignado responsabilidades claras a los integrantes del equipo?		
16	¿El equipo de pruebas estuvo de acuerdo en iniciar las pruebas de regresión?		

Tabla 15. Lista de comprobación para determinar los criterios de salida de las pruebas de regresión que debe completar el líder de pruebas junto con los integrantes del equipo (CL 3.0). Nota: Traducida de Minhas et al., (2023)

CLI	Elemento de la lista de comprobación	Estado	Comentarios
1	¿Se han ejecutado completamente los conjuntos de pruebas de regresión?		
2	¿Ha alcanzado el umbral la tasa de aprobación de los conjuntos de pruebas de regresión?		
3	¿Se han resuelto todos los defectos graves/críticos?		
4	¿Se han cerrado todos los defectos de gravedad media?		
5	¿Se han recopilado todas las medidas?		
6	¿Se han logrado los objetivos definidos de las pruebas de regresión?		
7	¿Los integrantes del equipo de pruebas están de acuerdo con el cierre de la prueba?		

Durante los talleres dedicados a la mejora de las listas de comprobación se solicitó a los participantes que expresaran su opinión sobre la relevancia de las listas y sus componentes. Se les pidió que seleccionaran “Sí” si consideraban relevante la lista o el elemento en cuestión, “No” si no lo era, y “No sé” si tenían dudas. Además, se proporcionó espacio adicional en los formularios para que los participantes pudieran añadir sugerencias y reflexiones. En este sentido, los investigadores compartieron el borrador inicial de las listas de comprobación y los formularios con los profesionales que participaron en la primera fase del estudio, quienes evaluaron los elementos y ofrecieron sus comentarios. Aquellos profesionales que participaron como grupo en fases anteriores proporcionaron su realimentación de manera colectiva. Las Tablas 16, 17 y 18 muestran un resumen de los comentarios proporcionados por los participantes del estudio. En cuanto a la relevancia de las listas de comprobación, los participantes estuvieron de acuerdo en que todas son pertinentes. Sin embargo, para algunos elementos específicos de las listas, algunos participantes seleccionaron “No” o “No sé”. Los colores rojo, gris y cian fueron utilizados para identificar los elementos con menos recomendaciones: rojo para aquellos que recibieron menos del 50%, gris para los que obtuvieron entre el 50% y el 60%, y cian para los que alcanzaron más del 60% pero menos del 80% de recomendaciones.

Tabla 16. Evolución de la lista de comprobación para determinar la preparación que deben completar los *testers* del equipo de prueba (CL 1.1). Nota: Traducida de Minhas et al., (2023)

CLI	¿Es relevante el elemento de la lista de comprobación?	Si	No	No lo sé
1	¿Conoce los objetivos del equipo en materia de pruebas de regresión?	9	1	0
2	¿Tiene conocimientos esenciales de las especificaciones del sistema?	9	0	1
3	¿Conoce las dependencias entre los subsistemas?	10	0	0
4	¿Está al tanto de los nuevos cambios en el sistema?	10	0	0
5	¿Ha analizado el impacto de los cambios en las partes inalteradas del sistema?	8	1	0
6	¿Se siente seguro realizando pruebas de regresión de forma independiente?	3	4	3
7	¿Ha recibido formación sobre las herramientas utilizadas para las pruebas de regresión dentro del equipo/organización?	10	0	0
8	¿Conoce la criticidad de los subsistemas que se van a probar?	9	0	1
9	¿Tiene acceso a los datos de las pruebas?	8	0	2

Tabla 17. Evolución de la lista de comprobación para determinar la preparación del equipo para ser completada por el líder de pruebas (CL 2.1). Nota: Traducida de Minhas et al., (2023)

CLI	¿Es relevante el elemento de la lista de comprobación?	Si	No	No lo sé
1	¿Se han definido los objetivos de las pruebas de regresión?	9	1	0
2	¿Conocen los integrantes del equipo de pruebas las especificaciones del sistema?	9	0	1
3	¿Se han comprobado todos los cambios?	8	1	1
4	¿Se han comunicado los cambios al equipo de pruebas?	9	0	1
5	¿Se han probado los cambios de forma aislada?	7	1	2
6	¿Se ha determinado el impacto del cambio?	8	0	2
7	¿Está actualizado el conjunto de pruebas de regresión?	9	0	1
8	¿Se han identificado los casos de prueba asociados a las partes modificadas?	10	0	0
9	¿Se han identificado los casos de prueba asociados al módulo afectado?	9	0	1
10	¿Se ha determinado el alcance de las pruebas de regresión?	10	0	0
11	¿Se han incorporado las pruebas de regresión al plan de pruebas?	5	3	2
12	¿Se ha desarrollado el plan de pruebas de regresión?	6	2	2
13	¿Se dispone de los recursos necesarios?	9	0	1
14	¿Se ha tomado la decisión entre pruebas de regresión manuales o automatizadas?	8	1	1
15	¿Se han asignado responsabilidades claras a los integrantes del equipo?	9	1	0
16	¿Ha acordado el equipo de pruebas iniciar las pruebas de regresión?	6	2	2

Tabla 18. Evolución de la lista de comprobación para determinar los criterios de salida de las pruebas de regresión que debe completar el líder de pruebas junto con los integrantes del equipo (CL 3.1). Nota: Traducida de Minhas et al., (2023)

CLI	¿Es relevante el elemento de la lista de comprobación?	Si	No	No lo sé
1	¿Se han ejecutado completamente las *suites* de pruebas de regresión?	10	0	0
2	¿Se ha alcanzado el umbral de aprobación de las *suites* de pruebas de regresión?	9	0	1
3	¿Se han resuelto todos los defectos graves/críticos?	10	0	0
4	¿Se han cerrado todos los defectos de gravedad media?	7	1	2
5	¿Se han recopilado todas las medidas?	6	0	4
6	¿Se han alcanzado los objetivos definidos para las pruebas de regresión?	10	0	0
7	¿Están de acuerdo los integrantes del equipo de pruebas con el cierre de las pruebas?	9	0	1

Por último, la información recogida para dar respuesta a la RQ 3: *¿Cuál es la perspectiva de los profesionales sobre las listas de comprobación propuestas?* permitió la obtención de realimentación de 23 de los 25 participantes, provenientes de 10 de las 12 organizaciones involucradas en el estudio. Los profesionales que participaron colectivamente en las fases anteriores del estudio brindaron sus comentarios como grupo, por lo que los resultados fueron organizados de manera similar (véase Figura 2.31). Los investigadores afirman que los comentarios obtenidos fueron positivos en lo general, ya que la mayoría de los participantes estuvo de acuerdo en que las listas de comprobación eran completas, útiles y personalizables. En cuanto a la disposición a utilizar las listas, la mayoría se mostró favorable. Sin embargo, se recogieron opiniones divididas sobre la facilidad de adopción en las organizaciones. De acuerdo con los argumentos proporcionados, este resultado era esperado, ya que durante las entrevistas muchos de los participantes señalaron que, aunque desearan adoptar las listas de comprobación u otra herramienta de mejora de procesos, podrían recibir una respuesta negativa de la alta dirección de las respectivas organizaciones.

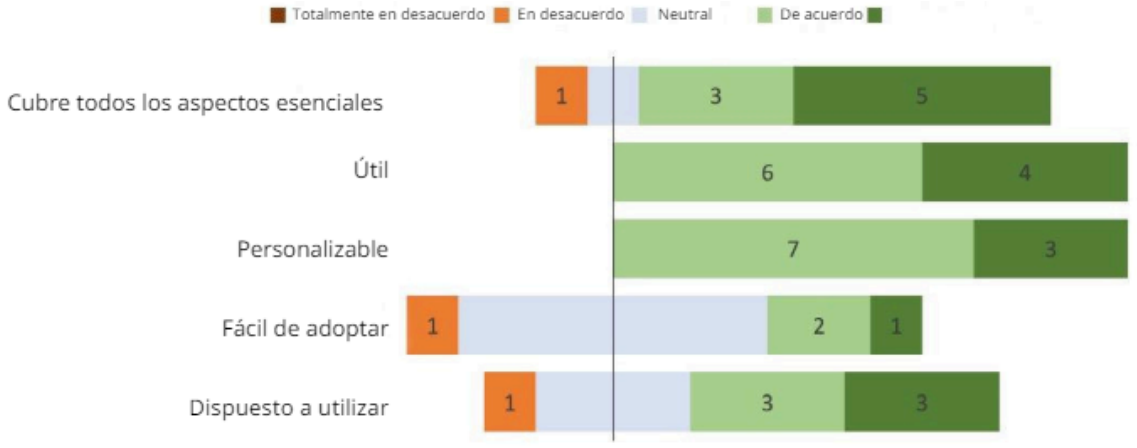


Figura 2.32. Realimentación de los participantes sobre la versión final de las listas de comprobación. Nota: Traducida de Minhas et al., (2023)

Con toda la información recogida en el estudio, los investigadores argumentan dos tipos principales de repercusiones. En primer lugar, los resultados específicos que contribuyen a la investigación en el campo de las pruebas de regresión. Este campo ha sido ampliamente estudiado y

la literatura ha propuesto, como se ha demostrado en este capítulo, numerosas técnicas al respecto. Sin embargo, el respaldo de las pruebas de regresión en la toma de decisiones, un área que a menudo es pasada por alto por los investigadores en Ingeniería de Software, es un aspecto que se destaca en este estudio. Por lo tanto, este enfoque basado en listas de comprobación abre nuevas posibilidades para otros investigadores, quienes ahora pueden trabajar en áreas como la gestión de las pruebas, el soporte a profesionales en tareas cruciales de las pruebas de regresión y la mejora del proceso en sí. Por otro lado, las repercusiones de la investigación empírica, puesto que la veracidad del estudio se respalda por la participación de los profesionales a lo largo de un período prolongado de tiempo, desde la identificación de las actividades relacionadas con las pruebas de regresión hasta la evaluación final de las listas de comprobación.

2.6. Conclusiones del capítulo

Como se podrá haber notado, la mayor parte de los modelos, técnicas y estrategias descritas en este capítulo de la tesis han sido diseñadas para planificar, realizar y gestionar las pruebas de regresión en las grandes organizaciones desarrolladoras de software, puesto que se asume un alto nivel de conocimiento, una amplia cantidad de recursos económicos y humanos, y una disponibilidad importante para ejecutar nuevos procesos. Sin embargo, y desafortunadamente, estas estrategias a menudo no son factibles en el contexto de las organizaciones de pequeño tamaño, puesto que éstas enfrentan múltiples limitaciones que dificultan su implementación. Estas limitaciones pueden ser analizadas desde diferentes perspectivas:

- Recursos financieros limitados. La implementación de estrategias avanzadas para realizar las pruebas de regresión requiere de una inversión significativa en herramientas y tecnologías también avanzadas. Sin embargo, las pequeñas organizaciones a menudo operan con presupuestos ajustados que no permiten asignar fondos suficientes para la adquisición de software especializado, infraestructuras robustas de hardware o servicios de consultoría para la integración y personalización de métodos complejos que, como se mencionó anteriormente, son creados de acuerdo con las características de las organizaciones de gran tamaño.
- Falta de capacitación y especialización del personal. La mayoría de las técnicas, modelos y estrategias descritas en este capítulo demandan conocimientos avanzados en, por ejemplo, minería de datos, NLP y modelos de aprendizaje computacional. Sin embargo, no todas las pequeñas organizaciones cuentan con el personal con las habilidades y conocimientos necesarios para desarrollar, implementar y mantener este tipo de propuestas. Aunado a lo anterior, la capacitación que le permita al personal adquirir tales conocimientos puede ser costosa y llevarse mucho tiempo. Un lujo que una pequeña organización difícilmente podrá asumir.
- Infraestructura tecnológica. Un modelo de recomendación de casos de prueba basado en el historial de pruebas, un sistema de apoyo para la toma de decisiones en la elección de una estrategia para las pruebas de regresión, los esquemas de recorrido de grafos para consultas en las pruebas requieren de una infraestructura tecnológica robusta que incluya servidores de alto rendimiento y proporcionen una capacidad suficiente de almacenamiento para gestionar grandes volúmenes de datos históricos de las pruebas. Sin embargo, las pequeñas organizaciones tampoco pueden disponer de la infraestructura necesaria dado su limitado presupuesto, afectando así la viabilidad técnica de implementar tales métodos o modelos.
- Gestión y mantenimiento continuo. Además de la dificultad que representa la implementación de los modelos, técnicas y/o estrategias analizadas en el estado del arte, su mantenimiento

implicaría un esfuerzo importante dada la necesidad de gestionar los datos y ajustar las técnicas para asegurar su precisión y relevancia. Este mantenimiento puede ser complejo y demandar recursos que las pequeñas organizaciones difícilmente pueden destinar.

- Complejidad de la implementación. La implementación de los modelos, técnicas y/o estrategias descritas anteriormente no solo requiere de tecnología y conocimientos, sino también de la definición de un proceso detallado de integración con los sistemas (rudimentarios o automatizados) ya existentes para la realización y gestión de las pruebas de regresión. Este proceso puede ser particularmente desafiante e innovador para las operaciones diarias de una pequeña organización, causando potencialmente más problemas de los que en verdad resuelve.
- Enfoque en la flexibilidad y agilidad. Finalmente, las pequeñas organizaciones suelen priorizar la flexibilidad y agilidad en sus procesos de desarrollo para responder rápidamente a las demandas de sus clientes y del mercado. Métodos o modelos complejos y estructurados como los analizados en el estado del arte pueden ser percibidos como estrictos y contraproducentes para la dinámica de trabajo rápida y adaptable que estas organizaciones necesitan mantener.

Así, todas estas limitaciones dificultan la implementación de métodos, técnicas y/o estrategias sofisticadas enfocadas a mejorar los resultados de las pruebas de regresión, lo que pone a las organizaciones pequeñas en desventaja frente a sus competidores más grandes. Por lo tanto, es evidente la necesidad de desarrollar enfoques más accesibles que se adapten a las necesidades y características de las organizaciones desarrolladoras de software de pequeño tamaño, permitiéndoles así mantener la calidad de sus productos sin incurrir en costos inasequibles.

En resumen, aunque los métodos o técnicas propuestos para mejorar las pruebas de regresión han demostrado ser efectivos, su implementación y operación presentan barreras significativas para las organizaciones pequeñas. Considerando lo anterior, en esta tesis se identificó la necesidad de crear una estrategia potencialmente compatible con las necesidades y capacidades operativas de este tipo de organizaciones, por lo que, después de haber investigado el tipo de propuestas existentes, se llega a las siguientes conclusiones:

- Se considerarán y ajustarán las recomendaciones de Minhas et al. (2023), quienes han identificado un conjunto de actividades esenciales para antes y después de la realización de las pruebas de regresión (véase Tabla 11) que se adecuarán al contexto de las pequeñas organizaciones desarrolladoras de software.
- Una vez que se haya establecido un conjunto de actividades que se alinee con los resultados del diagnóstico realizado sobre 34 organizaciones desarrolladoras de software de pequeño tamaño, se retomarán los principios de Jeon y von Mayrhauser (1994) y Middelveen (1998) para diseñar una estrategia basada en la gestión de conocimiento. Bajo esta premisa, no solamente se dotará a la organización de una estrategia de gestión, sino que se inculcará también la gestión de todo el conocimiento generado con la planeación, realización y gestión de las pruebas de regresión con el fin de mejorar la calidad de los productos generados.
- Por último, la estrategia diseñada será respaldada por una herramienta computacional que se creará considerando las recomendaciones de Taipale et at., (2007) y Liu et al. (2009), de tal manera que su integración en la organización sea no invasiva y le permita madurar el proceso relacionado con la realización de las pruebas de regresión.

3. Metodología: Diseño de la estrategia de gestión

El Capítulo 3 de esta tesis es la parte medular de la investigación, puesto que presenta el fundamento teórico de la solución propuesta, junto con detalles de su implementación. En este sentido, se fundamentan las decisiones metodológicas que fueron tomadas para diseñar una estrategia de gestión acorde a las características de las pequeñas organizaciones desarrolladoras de software.

3.1. La mejora del proceso de software

La investigación de Pesado et al., (2013) argumentó que la Mejora de los Procesos de Software (SPI, por sus siglas en inglés) podía verse como un conjunto de actividades que se realizan con la finalidad de que una organización desarrolladora de software pueda generar productos y/o servicios de alta calidad mediante la evaluación y adecuación de sus procesos. Es decir, si se considera que una organización es una proveedora de productos y servicios, el concepto de calidad está directamente vinculado con la idea de que dichos productos y servicios satisfagan las necesidades que el cliente demanda. Por lo tanto, el nivel de calidad de la organización dependerá de la manera en la que se generen tales productos y servicios. En este sentido, es lógico pensar que en la medida en que las organizaciones se vuelvan más metódicas y previsibles en la creación de sus productos de software, sus niveles de calidad se incrementarán positivamente. Así pues, las organizaciones desarrolladoras de software deben definir y utilizar procesos con el fin de establecer las actividades que deben realizarse para cada proyecto. Cuando estos procesos son utilizados en reiteradas ocasiones se convierten en normas o estándares organizacionales que garantizarán la calidad de los productos y servicios de software que se generen. Por consiguiente, Toapanta et al., (2017) consideraron precisamente que la SPI tiene como propósito analizar y definir formas de optimizar las prácticas de desarrollo de software dentro de una organización, basándose en una evaluación del proceso actual que considere las normas o estándares actuales. De esta manera, una iniciativa de mejora se enfoca en actualizar de manera disciplinada el rendimiento, la eficiencia y la efectividad de los procesos de acuerdo con lo que una práctica estándar establezca.

En este contexto, la SPI es un mecanismo clave para incrementar la competitividad y eficiencia dentro de las organizaciones desarrolladoras de software. Es decir, un enfoque que mejora parte de la idea de aumentar la madurez de un proceso de desarrollo de software, lo que a su vez conlleva a mejorar de forma efectiva la calidad de un producto y/o servicio de software, algo que diversos expertos asocian con un incremento significativo en la competitividad. Así, el objetivo en cuanto a la adopción de una iniciativa de SPI es alinear a la organización con el modelo, norma, o estándar de calidad anhelado para asegurar la calidad de los productos y/o servicios desarrollados a través de modelos, estándares y normas que regularmente se agrupan en dos categorías principales: aquellos enfocados en la calidad del producto y aquellos centrados en la calidad del proceso. Sin embargo, las

organizaciones desarrolladoras de software aún enfrentan desafíos importantes para definir y conducir una iniciativa de mejora puesto que regularmente no se considera adecuadamente el estado real de las organizaciones ni sus particularidades. Además, este tipo de iniciativas a menudo requieren de una inversión económica considerable que origina beneficios perceptibles a largo plazo, lo que lleva a que, en muchas ocasiones, se les considere como un gasto en lugar de una inversión, sin distinguir que la calidad es un valor que impulsa positivamente el crecimiento de la misma organización. Pero las organizaciones pueden mejorar si toman las decisiones correctas. La investigación realizada por Durán et al. (2017), por ejemplo, argumentó que es verdad que la SPI implicaba la adopción de estrategias y acciones enfocadas en optimizar y refinar las distintas fases y prácticas involucradas en el ciclo de vida del desarrollo de software, pero que también era importante considerar que, dado que una iniciativa podía abarcar más de una fase (e.g., planificación del proyecto, diseño, pruebas y el mantenimiento del software), debía priorizarse trabajar en aquellas que permitieran aumentar la eficiencia, efectividad y calidad del producto final a corto plazo, disminuyendo así el monto de la inversión inicial.

Considerando lo anterior, Morales-Aguiar y Vega Zepeda (2018) identificaron y agruparon una serie de Factores Críticos de Éxito (SCF, por sus siglas en inglés) que pueden influir en la definición y realización de las iniciativas de SPI que se realizan en las organizaciones desarrolladoras de software. Tales factores se describen en la Tabla 19.

Tabla 19. Factores críticos de éxito en iniciativas de SPI. Nota: Traducida de Morales-Aguilar y Vega Zepeda (2018)

Factor	Descripción
Orientación profesional	Este factor establece que las decisiones sobre la gestión de proyectos y la SPI deben estar alineadas con los objetivos estratégicos y comerciales de una organización, asegurando así que el esfuerzo técnico contribuya directamente al éxito del negocio.
Compromiso de la alta dirección	Este factor establece que se requiere de un alto nivel de compromiso y apoyo de los ejecutivos y de la alta dirección para lograr el éxito de la iniciativa de SPI.
Cultura de la organización	Este factor establece que los elementos internos que afectan la forma en que las organizaciones adoptan y mejoran sus procesos, tecnologías y metodologías, influyen directamente en la eficiencia y calidad de los proyectos.
Políticas de la organización	Este factor establece que las reglas, normas, directrices y decisiones establecidas por una organización afectan la implementación y gestión de los proyectos, especialmente en el ámbito del desarrollo de software y la mejora de los procesos.
Estructura de la organización	Este factor considera la configuración de la organización en términos de su jerarquía, comunicación, procesos y roles, y cómo ésta influye en la forma en que la organización gestiona sus proyectos y recursos. Por lo tanto, esto puede impactar en el éxito o fracaso de la gestión de la calidad en el desarrollo de software y otros procesos organizativos.
Asignación de recursos	Este factor considera la correcta distribución y disponibilidad de los recursos materiales, humanos y tecnológicos en los proyectos de SPI. Por lo tanto, es un factor clave para asegurar que los proyectos tengan lo necesario para avanzar de manera eficiente, evitando sobrecargas de trabajo o falta de insumos críticos.
Objetivos principales	Este factor se relaciona con los objetivos fundamentales que guían a un proyecto o proceso, particularmente en la SPI y la implementación de metodologías en la Ingeniería de Software.
Beneficios de SPI	Este factor considera los beneficios que la SPI proporciona a las organizaciones. Estos beneficios suelen incluir mejoras en la calidad del software, reducción de costos, mayor eficiencia y productividad, y una mejor alineación con las necesidades del cliente.

Factor	Descripción
Responsabilidades de roles	Este factor considera el compromiso que cada persona debe aportar para cumplir con las tareas asignadas a su rol dentro de una iniciativa de SPI.
Estabilidad política	Este factor establece que el contexto político de un país o región pueden afectar el desarrollo de software y la gestión de los proyectos. Por lo tanto, se debe considerar el entorno general en el cual las organizaciones operan y cómo éste puede afectar tanto la viabilidad como el éxito de los proyectos de desarrollo de software, especialmente en cuanto a la predictibilidad y la reducción de riesgos.
Gestión eficaz	Este factor se refiere a la capacidad de una organización para administrar de manera eficiente y efectiva sus recursos, procesos y prácticas para alcanzar los objetivos propuestos. Es decir, se enfoca en la importancia de la administración estratégica de recursos humanos, tecnológicos y financieros para mejorar los resultados en el desarrollo de software o la implementación de proyectos tecnológicos.
Visión de la organización	Este factor considera cómo una organización define su propósito, metas y dirección estratégica a largo plazo. Este factor es crucial en el contexto de la gestión y la estrategia empresarial, puesto que no solo se trata de un enunciado aspiracional, sino de un elemento crítico para el éxito organizacional, puesto que puede afectar tanto la estrategia como la cultura de la empresa.
Participación del personal	Este factor considera el grado en que el personal dedicado a la mejora de procesos participa y se involucra en la realización de las actividades de mejora y en la toma de decisiones.
Conciencia sobre SPI	Este factor establece que el personal involucrado en la mejora de procesos perciba los beneficios y esté informado de los avances obtenidos con el esfuerzo que se está realizando en la SPI.
Líderes de SPI	Este factor establece que los líderes de la SPI deben ser los responsables de guiar, implementar y fomentar prácticas efectivas que optimicen los procesos de desarrollo, asegurando que se alineen con las mejores prácticas y estándares de la industria.

Considerando el FCE número 11 de la Tabla 19, la "Gestión eficaz" es un pilar fundamental para la SPI, ya que tiene un impacto directo en la calidad, los costos, la flexibilidad y la capacidad de innovación dentro de las organizaciones (Singh et al., 2015). En el contexto de esta tesis, este factor es de una relevancia importante, puesto que implica la definición de un proceso de pruebas de regresión que considere estándares de calidad y/o buenas prácticas que garanticen que el software desarrollado sea confiable y cumpla con las expectativas del cliente. Además, a través de una gestión eficaz del proceso de pruebas de regresión se contribuiría a la reducción de costos y tiempos de desarrollo al optimizar estas pruebas y eliminar así ineficiencias. De hecho, esta capacidad de optimización permitiría que las pequeñas organizaciones se mantengan competitivas y maximicen el uso de sus recursos ya que, al establecer un marco de trabajo claro y eficiente, se liberan recursos y tiempo que pueden ser dedicados a explorar nuevas ideas y soluciones.

La consideración de estos SCF es crucial para aspirar al éxito de la iniciativa de mejora. Gasca‐Hurtado et al. (2021), por ejemplo, consideran que, dado que las organizaciones desarrolladoras de software requieren procesos maduros para cumplir con los estándares de calidad y productividad definidos por el mercado, deben prestar atención a la literatura que documenta aquellos FCE que promueven la satisfacción de los requisitos del software y las demandas de los usuarios. Con este objetivo en mente, se han creado diversos modelos y normas para establecer las mejores prácticas relacionadas con SPI. Sin embargo, el desarrollo de estos procesos se considera centrado en las personas y está fuertemente vinculado a factores sociales y humanos (SHF, por sus siglas en inglés)

que suelen estar intrínsecamente relacionados con los individuos involucrados en dicho proceso. De manera similar, las iniciativas de SPI a menudo están ligadas a otros factores que involucran cambios en los comportamientos de los actores involucrados. Estos cambios implican, a su vez, modificaciones en la manera de llevar a cabo las actividades relacionadas con el desarrollo de los proyectos de software. En este sentido, los modelos y normas vinculadas a las iniciativas de SPI y a la gestión del cambio se enfocan específicamente en fomentar el desarrollo de productos de software de mayor calidad, haciendo más énfasis en los aspectos técnicos, instrumentales y procedimentales; mientras que los SHF como el compromiso de la alta dirección, la participación del personal involucrado, la responsabilidad para desempeñar los roles en cada equipo de trabajo, la concientización sobre la SPI, la falta de adhesión y participación de todos los implicados en la mejora continua, la presión y la falta de planificación en el período de adaptación de un proyecto, suelen, erróneamente, quedar fuera del entorno de desarrollo de un proyecto. No obstante, dado que las iniciativas de SPI son conducidas por humanos, estos factores deben tenerse en cuenta para minimizar la resistencia al cambio y maximizar la posibilidad de lograr el éxito.

En el contexto actual, los modelos para mejorar los procesos de software han vuelto a captar la atención de la comunidad científica. La principal motivación para emplear este tipo de modelos radica en la necesidad de incrementar las probabilidades de éxito en los proyectos de software. De acuerdo con Cornide-Reyes et al., (2024), las iniciativas de SPI han representado un continuo desafío para la industria moderna de software. No obstante, las organizaciones desarrolladoras de software buscan constantemente experimentar con métodos eficientes que les permitan mejorar sus procesos con el fin de aumentar la eficacia, calidad, y satisfacción del cliente al entregar, en tiempo y forma, los productos/servicios que fueron solicitados. De hecho, durante los últimos tres años se ha sugerido el uso de modelos de predicción y crecimiento, junto con herramientas y prácticas organizacionales adecuadas a su entorno. Estos modelos y herramientas son valiosos para cualquier organización de desarrollo de software, ya que permiten identificar métodos, ecuaciones y criterios que ayudan a evaluar de forma cuantitativa la fiabilidad de un producto y/o servicio de software. Sin embargo, todo este esfuerzo realizado por una organización enfrenta diversas dificultades y barreras que deben ser superadas de forma efectiva. Por ejemplo, es importante destacar que, durante el desarrollo de los productos y/o servicios de software, las organizaciones generan una cantidad importante de información que no suele estar documentada y que se conoce como conocimiento tácito. Este tipo de conocimiento es especialmente valioso, ya que contribuye a mejorar los procesos y prevenir la repetición de errores. En este contexto, la implementación de mejoras en los procesos de software es cada vez más relevante debido a la creciente demanda de calidad en los productos y servicios que se ofertan. Sin embargo, muchas organizaciones no aprovechan el conocimiento y esto se ve a menudo reflejado en sus procesos de desarrollo de software (Mejía et al., 2019). Por ende, para abordar este desafío, a lo largo del tiempo se han identificado diversas técnicas, herramientas y estrategias que apoyan la formalización del conocimiento tácito, lo que contribuye directamente a alcanzar el éxito en las iniciativas de SPI. Entre estas técnicas se incluyen el uso de cuestionarios, herramientas de extracción de conocimiento (como los repositorios de datos), la minería de procesos y herramientas de indexación avanzada, todas ellas diseñadas para capturar, formalizar y aprovechar el conocimiento que reside en los empleados. Además, varios casos de estudio muestran cómo la formalización del conocimiento tácito ha permitido identificar procesos empíricos y áreas de mejora dentro de las organizaciones gubernamentales, laboratorios de software y otras instituciones. Estos casos han demostrado también cómo una adecuada gestión del conocimiento no solamente facilita la mejora de los procesos de software, sino que también facilita la creación de nuevas oportunidades de optimización y crecimiento organizacional.

En este sentido, la gestión del conocimiento juega un papel crucial en la mejora de los procesos, ya que permite formalizar y utilizar el conocimiento generado dentro de una organización, en particular el conocimiento tácito, que suele no estar registrado y es fundamental para evitar la repetición de errores y replicar casos de éxito. Sin embargo, muchas organizaciones, principalmente las pequeñas, enfrentan dificultades para capturar y formalizar este conocimiento, lo que afecta la calidad y eficiencia de sus procesos de desarrollo. Además, la situación de estas organizaciones es aún más complicada debido al tamaño reducido de sus equipos de trabajo, la falta de claridad en la definición de los roles, las responsabilidades mal delimitadas y los recursos limitados con los que suelen contar. No obstante, la combinación de la mejora de los procesos de software y la gestión del conocimiento puede mejorar la capacidad de estas organizaciones para capturar y formalizar el conocimiento interno logrando así mejoras sustanciales en la calidad del software que producen.

3.2. La gestión del conocimiento

Abdullah (2011) consideró que la Gestión del Conocimiento (KM, por sus siglas en inglés) se refiere al proceso de capturar y utilizar la experiencia colectiva de una organización, ya sea en forma de documentos, bases de datos o, posiblemente, en las mismas mentes de las personas. Así, su objetivo principal es mejorar la resolución de problemas, el aprendizaje dinámico, la planificación estratégica y la toma de decisiones. Aunado a esto, la KM facilita a una organización la identificación, selección, organización, diseminación y transferencia de información importante, asegurando que el conocimiento acumulado esté accesible y sea útil para toda la organización. Por otra parte, Durán et al., (2017) consideran que la KM es un conjunto de procesos que regulan la creación, difusión y uso del conocimiento dentro de una organización. No obstante, estos procesos se enfocan en optimizar el conocimiento ya existente, tanto tácito (i.e., no documentado) como explícito (i.e., documentado), para generar valor dentro de la organización. Esto implica identificar, adquirir, conservar, distribuir y utilizar el conocimiento con el objetivo de mejorar la toma de decisiones y las operaciones organizacionales.

Desde un punto de vista más actual, Candal et al., (2022) afirman que la KM es un proceso organizado que abarca la creación, almacenamiento, intercambio y utilización del conocimiento en una organización, y cuyo propósito es administrar eficazmente los recursos de conocimiento, permitiendo a la organización maximizar el uso del conocimiento disponible y promover la innovación y la mejora continua. Para lograr dicho objetivo se requiere de un Sistema de Gestión del Conocimiento (KMS, por sus siglas en inglés) que utilice tecnologías modernas, como Internet, *intranets* y *data warehouses* con el objetivo de organizar y acelerar la gestión del conocimiento dentro de una organización. El ciclo tradicional de vida de un KMS incluye los siguientes puntos clave:

- Crear el conocimiento: Desarrollar nuevas formas de hacer las cosas o adquirir conocimientos externos.
- Capturar el conocimiento: Identificar y representar de manera útil el conocimiento nuevo.
- Refinar el conocimiento: Colocar el conocimiento en contexto para hacerlo procesable.
- Almacenar el conocimiento: Guardar el conocimiento en repositorios accesibles.
- Gestionar el conocimiento: Mantener el conocimiento actualizado y relevante.
- Diseminar el conocimiento: Hacer que el conocimiento esté disponible para todos los que lo necesiten en la organización.

Sin embargo, uno de los principales desafíos en las organizaciones de desarrollo de software es la falta de transparencia en la gestión del conocimiento. Frecuentemente, el conocimiento valioso no se documenta adecuadamente, lo que conduce a su pérdida cuando las personas dejan la organización. La implementación de estrategias efectivas de KM ayuda a cerrar estas brechas, permitiendo que el conocimiento se comparta y se utilice de manera óptima dentro de toda la organización. Por lo tanto, como se ha mencionado anteriormente, para lograr una KM efectiva, es importante distinguir entre los dos tipos de conocimiento: tácito y explícito. El conocimiento tácito es subjetivo y está basado en la experiencia individual, mientras que el conocimiento explícito es aquel que se encuentra documentado y es accesible para toda la organización. Sin embargo, el mayor desafío de la KM es convertir el conocimiento tácito en explícito, ya que esto permite su diseminación y reutilización a lo largo de la organización, facilitando así la integración de nuevos miembros al equipo y acelerando su curva de aprendizaje. La KM, por ende, puede verse como un conjunto de procesos que gobiernan la creación, diseminación y utilización del conocimiento. Así, sus tres elementos fundamentales son:

- Diferenciación entre datos, información y conocimiento: Los datos son hechos aislados, la información es un conjunto de datos con significado, y el conocimiento es la apropiación de la información por parte de las personas para generar nuevos conocimientos.
- Uso de datos e información en un contexto específico: Mediante el procesamiento cognitivo, las personas transforman los datos y la información en conocimiento útil en situaciones concretas.
- Compartir, interpretar y socializar el conocimiento: Para que el conocimiento sea útil y se transforme en un generador de nuevo conocimiento, es esencial su interpretación y socialización entre personas.

Por otro lado, el conocimiento dentro de las organizaciones desarrolladoras de software se clasifica en cuatro tipos importantes:

- Conocimiento organizacional: Está relacionado con la gestión de la organización, los recursos humanos y los objetivos empresariales.
- Conocimiento de gestión: Involucra la planificación, liderazgo y seguimiento de proyectos.
- Conocimiento técnico: Abarca el análisis de requisitos, diseño, programación y pruebas de software.
- Conocimiento del dominio: Se refiere a conocimientos específicos de industrias o áreas, como, por ejemplo, la banca, telecomunicaciones, seguros, entre otros.

3.3. La gestión del conocimiento en la mejora del proceso de pruebas de software

La KM fortalece la mejora continua de un proceso de software al facilitar la creación, compartición y retención del conocimiento, lo que impacta directamente en la calidad del software y la eficiencia de los equipos de desarrollo (Ramachandran, 2011). La KM es pues una herramienta esencial en la SPI, ya que permite optimizar el flujo de información, la toma de decisiones y la actualización continua de las prácticas de desarrollo. El proceso comienza con la creación y compartición de conocimiento dentro de los equipos de desarrollo. A través de la comunicación abierta e informal, los miembros del equipo intercambian tanto conocimiento tácito como explícito, lo que facilita la transmisión de ideas, experiencias y lecciones aprendidas. Este flujo constante de

información permite mejorar los procesos al evitar la repetición de errores y aprovechar soluciones exitosas del pasado. A medida que se comparte el conocimiento, éste proporciona un soporte crucial para la toma de decisiones. Por lo tanto, contar con un proceso efectivo de gestión del conocimiento ayuda a los equipos a tomar decisiones más informadas y rápidas, ya que el conocimiento colectivo permite resolver problemas de desarrollo con mayor eficiencia y adaptarse a cambios en el mercado o en los requisitos del cliente. El siguiente paso en el ciclo es la evolución de las prácticas de desarrollo. A través del conocimiento acumulado, las organizaciones pueden actualizar continuamente sus procesos de software. Finalmente, la KM es clave para mitigar la pérdida de conocimiento. Al establecer sistemas que capturen y retengan el conocimiento de los empleados, las organizaciones pueden asegurar que la salida de personal no afecte negativamente a los procesos. Esto garantiza la continuidad del desarrollo de software y la mejora de procesos, preservando el conocimiento adquirido y permitiendo que las lecciones aprendidas se transmitan de manera efectiva. En conjunto, estos elementos muestran cómo la KM impulsa la SPI al facilitar la creación, retención y uso eficiente del conocimiento dentro de las organizaciones.

De acuerdo con Candal et al. (2022), la KM facilita la administración de los activos de conocimiento dentro de las organizaciones, principalmente a través de la implementación de procesos, prácticas y herramientas. De este modo, diversas prácticas de KM se enfocan en manejar el conocimiento tácito y pueden ser aplicadas durante las etapas del proceso de desarrollo de software. Así mismo, como ya se ha indicado, la SPI consiste en optimizar y perfeccionar los métodos y prácticas empleados en el desarrollo de software. Esto implica la adopción de metodologías que incrementen la eficiencia, la calidad y la productividad en la creación de los productos de software, ajustándose, al mismo tiempo, a las demandas cambiantes del mercado y aprovechando las innovaciones tecnológicas más recientes. Por lo tanto, la KM puede combinarse con la SPI para generar valor en las organizaciones, apoyándose en procesos, herramientas y actividades que les permitan obtener mayores beneficios. La KM, por ende, se emplea en la SPI principalmente mediante la implementación de prácticas que permiten capturar y compartir el conocimiento tácito entre los miembros del equipo, facilitando la optimización de los procesos, mejorando la calidad del software y apoyando la sostenibilidad del conocimiento dentro de la organización. La aplicación de estas prácticas puede mejorar diversos aspectos del desarrollo de software, como los artefactos y el producto final.

Así pues, la KM se incorpora en la mejora de los procesos de software al ofrecer un marco que facilita la reutilización, conservación y aplicación del conocimiento obtenido durante los proyectos de software. En el ámbito de las pruebas de software, por ejemplo, la KM permitiría capturar y reutilizar conocimientos previos sobre problemas y soluciones, lo que optimiza las pruebas futuras y reduce los riesgos asociados a la pérdida de conocimiento debido a la rotación de personal o la falta de documentación adecuada. En este sentido, Soto et al., (2017) argumentan que en el proceso de pruebas de software se han identificado varios problemas clave que afectan su efectividad, tales como la baja reutilización del conocimiento, las barreras en la transferencia de éste, y la existencia de entornos inadecuados para su intercambio. Además, la alta rotación de personal provoca la pérdida de conocimientos acumulados, lo que dificulta la continuidad y mejora del proceso. No obstante, estos mismos investigadores coinciden en que los enfoques dirigidos específicamente a la KM permiten a las organizaciones optimizar sus procesos de pruebas de software al utilizar el conocimiento como un activo esencial para mejorar la eficiencia y efectividad del equipo de pruebas. Por lo tanto, una estrategia efectiva de KM puede ayudar a mitigar riesgos y a cerrar brechas de conocimiento, con el objetivo de mejorar la calidad y reducir el tiempo y los costos de retrabajo. Este enfoque puede mejorar la efectividad del proceso de pruebas al identificar, preservar y gestionar el conocimiento crítico, facilitando así la detección y resolución de defectos en el software.

Considerando lo anterior, Maciel et al., (2018) afirmaron que la gestión del conocimiento juega un papel fundamental en el ámbito de las pruebas de software, un proceso altamente intensivo en conocimiento. De hecho, argumentaron que la reutilización de las experiencias y los conocimientos adquiridos en proyectos anteriores, como el diseño de casos de prueba y la detección de fallos, es clave para mejorar la calidad del producto y optimizar los tiempos y costos de desarrollo. Al utilizar la KM en este contexto, se logra capturar, almacenar y compartir información valiosa que puede utilizarse en futuras etapas del desarrollo de software. En este sentido, uno de los aspectos más destacados de la KM en el ámbito de las pruebas de software es su aplicación en las actividades de planificación de las pruebas y en la reutilización de los casos de prueba, puesto que se obtienen mayores beneficios que permiten reducir significativamente los costos y tiempos de desarrollo. Además, la planificación estratégica, así como el uso repetido de pruebas previamente documentadas, optimizan el esfuerzo y mejoran la eficiencia general del proceso de pruebas.

Haciendo un análisis similar, Wnuk y Garrepalli (2018) argumentan que la KM es crucial para capturar, compartir, distribuir y comprender el conocimiento acumulado por los equipos de pruebas. Esto no solo mejora la calidad del proceso, sino que también reduce costos y optimiza la ejecución de las pruebas. Así, uno de los aspectos clave de la KM en las pruebas de software es el incremento de la efectividad. Es decir, a través de la gestión adecuada del conocimiento, los equipos pueden seleccionar y aplicar las técnicas de prueba más adecuadas, lo que resulta en una mayor efectividad general del proceso de pruebas. Además, la reutilización del conocimiento existente y las lecciones aprendidas permite disminuir los costos y el esfuerzo necesarios, lo que acorta los tiempos de desarrollo y, en consecuencia, la entrega del software. Otro beneficio importante de la KM, de acuerdo con estos investigadores, es que facilita la toma de decisiones, ya que proporciona a los equipos información clave para elegir las mejores técnicas de prueba y mejorar los procesos. Asimismo, la reutilización del conocimiento es uno de los aspectos más valiosos, ya que permite aprovechar la experiencia previa y evitar la repetición de errores, optimizando así los recursos y el tiempo.

De manera similar, la investigación de Ibitowa y Akinola (2021) establece que la KM en las pruebas de software es vista como una estrategia esencial para mejorar la calidad, eficiencia y efectividad en los procesos de desarrollo y pruebas, impulsando la reutilización y la distribución del conocimiento. En este sentido, la KM en el contexto del desarrollo de software, particularmente en las pruebas de software, es un proceso esencial que implica la distribución, compartición, captura, creación y comprensión del conocimiento dentro de una organización. Este enfoque es fundamental para mejorar la eficiencia y productividad, permitiendo a las organizaciones aprovechar tanto el conocimiento explícito como el tácito. El conocimiento explícito es objetivo, documentado y fácil de compartir, como los manuales de pruebas y artefactos de pruebas. Por otro lado, el conocimiento tácito reside en la experiencia personal, creencias y valores de los individuos, y generalmente no está documentado, lo que lo hace más difícil de transmitir a otros miembros de la organización. Ambos tipos de conocimiento juegan un papel crucial en el proceso de pruebas de software.

Para gestionar el conocimiento de manera efectiva, las organizaciones de desarrollo de software suelen adoptar seis prácticas clave: la adquisición, creación, compartición, almacenamiento, organización y aplicación del conocimiento. Estas prácticas permiten una mejor gestión del saber acumulado, lo que facilita el acceso a información valiosa para mejorar los procesos de pruebas y evitar la duplicación de esfuerzos. En consecuencia, la aplicación de la gestión del conocimiento en las pruebas de software tiene múltiples beneficios tales como:

- La facilidad de identificación de fallos y la mejora de la capacidad de los *testers* para detectar y reconocer defectos en el software.

- El conocimiento del dominio, del sistema y de la Ingeniería de Software en general es esencial para que el proceso de pruebas sea efectivo a través de la definición de tres componentes clave:
    - Cumplimiento de la estructura de un proceso que esté alineado con un estándar, modelo o marco de trabajo que defina las etapas de las pruebas de software.
    - Estrategia de pruebas basada en riesgos que sea capaz de identificar y mitigar riesgos en el proceso de pruebas.
    - Tipología del conocimiento que defina el tipo de conocimiento necesario para agregar valor en la identificación y resolución de fallos en las pruebas de software.

Como se podrá entender, la KM puede también mejorar la calidad de las pruebas de software al permitir la reutilización del conocimiento previamente generado, evitando así la "reinvención de la rueda" y el gasto innecesario de tiempo. Además, permite mejorar la calidad, eficiencia y efectividad en los procesos de desarrollo y pruebas de software, puesto que facilita la reutilización y compartición del conocimiento, asegurando que la organización pueda aprovechar al máximo su saber acumulado para optimizar sus operaciones y resultados, y asegurando el incremento de la productividad de las organizaciones, haciéndolas más eficientes y efectivas en la gestión de sus procesos de pruebas de software. De hecho, tanto en el ámbito académico como en el empresarial, se han desarrollado diversas iniciativas para integrar la KM en las pruebas del software, enfrentando como desafío común la falta de reutilización del conocimiento, que se ve afectada por factores como las barreras para compartir información, la alta rotación de personal, y modelos de gestión del conocimiento con enfoque corporativo y de implementación compleja, entre otros. Sin embargo, la implementación efectiva de este tipo de gestión aún enfrenta desafíos significativos. Entre los principales obstáculos se encuentran la falta de sistemas adecuados para gestionar el conocimiento y la resistencia de los empleados a compartirlo, lo que dificulta la creación de un repositorio accesible y actualizado. Por otro lado, la transferencia y captura del conocimiento, tanto explícito como tácito, es fundamental en las pruebas de software. Sin una gestión adecuada, se corre el riesgo de una baja reutilización del conocimiento y la aparición de barreras en su transferencia. La falta de prácticas adecuadas puede generar varios desafíos, como una tasa de reutilización de conocimiento baja, dificultades en la transferencia de información, entornos ineficientes para compartir conocimientos, pérdida de valioso capital intelectual por la rotación de personal y problemas para lograr una óptima distribución de recursos humanos durante las pruebas.

A pesar de estas barreras, las organizaciones que logran implementar la gestión de conocimiento de manera efectiva experimentan importantes beneficios en términos de calidad, eficiencia y reducción de costos en sus procesos de pruebas de software. Es por ello, que se considera que la KM ofrece un enorme potencial para mejorar el desempeño de las pruebas de software, siempre y cuando se superen los desafíos asociados a su implementación y se logre una conversión eficaz del conocimiento tácito en explícito, beneficiando así a toda la organización. En conclusión, la implementación de la gestión del conocimiento en las pruebas de software no solo puede mejorar la calidad y la eficiencia, sino que también permite enfrentar los retos asociados a la falta de estas prácticas, contribuyendo a la definición de un proceso de pruebas más efectivo y sostenible.

3.4. Enfoques para implementar la gestión del conocimiento en la mejora de los procesos de software

La *minería de procesos* es una técnica que combina métodos de minería de datos con enfoques de Gestión de Procesos de Negocio (BPM, por sus siglas en inglés), cuyo objetivo principal es

descubrir, analizar y mejorar los procesos operativos de una organización a partir de los datos generados en sus sistemas de información. Entre las principales actividades de la minería de procesos, Van der Aalst (2016) destaca las siguientes:

- Descubrimiento de los procesos, que consiste en crear un modelo basado en datos reales de eventos registrados por los sistemas, sin información previa del proceso, utilizando algoritmos que representan los procesos como modelos gráficos.
- Revisión de conformidad, que compara el modelo descubierto o teórico con los registros reales, permitiendo así la verificación de que las actividades se ejecuten conforme a lo planificado con el fin de identificar desviaciones y excepciones.
- Mejora del proceso, que se enfoca en ajustar o mejorar los modelos de proceso existentes a partir de datos adicionales o el análisis de los tiempos de ciclo, costos o calidad con el fin de optimizar el rendimiento.

Las aplicaciones prácticas de la minería de procesos incluyen a la mejora continua de los procesos, la auditoría y verificación de procesos, la optimización de los flujos de trabajo en Sistemas de Planificación de Recursos (ERP, por sus siglas en inglés) o en sistemas de Gestión de Relaciones con los Clientes (CRM, por sus siglas en inglés), la identificación de cuellos de botella y la mejora de áreas operativas. Es importante también mencionar que la minería de procesos no solo proporciona una visión retrospectiva para la organización, sino que también ofrece información predictiva y prescriptiva, facilitando así la toma de decisiones estratégicas para la mejora de procesos.

Por otro lado, la *gestión del conocimiento basada en procesos* es un método que combina la gestión del conocimiento con la BPM con el objetivo de capturar, organizar y reutilizar el conocimiento generado a partir de las actividades y procesos de una organización. De acuerdo con Nonaka y Takeuchi (2007), este enfoque permite que tanto el conocimiento explícito como el implícito se integren en los flujos de trabajo y procedimientos internos, garantizando su uso efectivo a lo largo de todo el ciclo de vida de los procesos. Los principales elementos de este enfoque incluyen:

- La captura del conocimiento, que consiste en identificar y documentar el conocimiento obtenido durante la ejecución de los procesos, ya sea a través de procedimientos formales o de las experiencias de los empleados.
- La distribución del conocimiento, que se centra en hacer accesible este conocimiento a las personas y equipos que participan en procesos similares, utilizando herramientas de colaboración, bases de datos y repositorios de información.
- La aplicación del conocimiento, que impulsa la reutilización y adaptación del conocimiento para aumentar la eficiencia, minimizar errores y fomentar la innovación.

Este enfoque facilita la realimentación continua, donde el conocimiento adquirido en la práctica diaria contribuye a la mejora de los procesos actuales. Las aplicaciones de la gestión del conocimiento basada en procesos incluyen a la estandarización de mejores prácticas, el aumento de la eficiencia en la formación de empleados y la optimización de procesos operativos a través del aprovechamiento de conocimientos previos. En este mismo sentido, el modelo SECI, propuesto por Nonaka y Takeuchi (2009), introdujo cuatro modos de conversión de conocimiento que dieron origen a su nombre:

- Socialización: Establece que el intercambio de experiencias y conocimientos tácitos se realice a través de la interacción social. El conocimiento se transmite de manera implícita, por ejemplo a través de la observación, la imitación o la tutoría.
- Externalización: Establece la necesidad de convertir el conocimiento tácito en explícito. Esto puede ocurrir a través de la creación de metáforas, analogías, modelos o prototipos.
- Combinación: Establece que el conocimiento explícito sea sistematizado, combinado y reorganizado para crear nuevo conocimiento. Esto implica la utilización de bases de datos, sistemas de información y otras herramientas útiles para gestionar la información.
- Internalización: Establece que el conocimiento explícito sea convertido nuevamente en tácito a través de la experiencia personal. Los individuos internalizan el conocimiento y lo integran a sus propias capacidades y habilidades.

De esta manera, se asume que el conocimiento cambie continuamente entre tácito (i.e., difícil de formalizar y comunicar) y explícito (i.e., fácilmente documentado). Este proceso cíclico es clave para crear nuevo conocimiento organizacional y aplicar mejoras en los procesos. Por lo tanto, el modelo SECI representa una herramienta valiosa para comprender cómo las organizaciones generan nuevo conocimiento. De hecho, Nonaka y Takeuchi conciben este proceso como una espiral continua, donde el conocimiento se crea, se comparte y se transforma de manera constante. Cada vuelta de la espiral enriquece el conocimiento organizacional y permite a las empresas adaptarse a un entorno cambiante.

Desde la perspectiva organizacional, el conocimiento se puede definir como la información que impulsa la ejecución de acciones orientadas a satisfacer las necesidades del mercado y a aprovechar nuevas oportunidades mediante el uso de las competencias clave de la organización. Este conocimiento combina valores, información contextualizada y experiencias, ofreciendo un marco de referencia que permite evaluar e integrar nuevas experiencias e información. En este sentido, Ibidunni et al., (2020) consideran que es verdad que el conocimiento se origina y se aplica en la mente de las personas, pero que en las organizaciones también se almacena en forma de documentos y bases de datos, y que se refleja en procesos, prácticas y normas internas. Por lo tanto, las herramientas para la extracción de conocimiento son esenciales en la era actual de la información, en la que el volumen y la complejidad de los datos aumentan rápidamente en todos los ámbitos. Su función principal es procesar, organizar y convertir datos en bruto en información valiosa y útil, ayudando de esta manera a organizaciones e investigadores a descubrir patrones, relaciones y conocimientos que no son fácilmente visibles. A través de técnicas avanzadas como la minería de datos, el NLP, el aprendizaje automático y el análisis semántico, estas herramientas automatizan la identificación de patrones y la creación de modelos, facilitando una toma de decisiones más precisa y fundamentada en datos. En la Tabla 20 se muestran algunos ejemplos de herramientas que han sido utilizadas exitosamente para la extracción del conocimiento a lo largo del tiempo.

Tabla 20. Herramientas de apoyo para la extracción de conocimiento

Nombre de la herramienta	Descripción	Aplicaciones más comunes
Prolog (Colmerauer, 1990)	Es un lenguaje de programación lógico utilizado principalmente en Inteligencia Artificial. Prolog es útil para la extracción de conocimiento basado en reglas.	Representación de conocimiento y extracción basada en inferencias lógicas, principalmente en sistemas expertos y sistemas de diagnóstico

Nombre de la herramienta	Descripción	Aplicaciones más comunes
RapidMiner (Mierswa et a., 2006)	Esta plataforma, anteriormente conocida como YALE, facilita la minería de datos, el aprendizaje automático y la preparación de datos. Permite a los usuarios la realización de procesos complejos de extracción de conocimiento sin necesidad de programar extensamente.	Exploración de datos, construcción de modelos predictivos y preparación de datos en aplicaciones empresariales.
NLTK (Natural Language Toolkit) (Bird, 2006)	Se trata de una biblioteca de Python para el NLP que permite analizar textos y extraer conocimiento a partir de datos lingüísticos.	Análisis de textos, minería de opiniones, y extracción de entidades nombradas (e.g., personas, lugares, organizaciones) en datos no estructurados.
KNIME (Konstanz Information Miner) (Berthold et al., 2009)	Es una herramienta de análisis y extracción de datos que usa una interfaz visual para construir flujos de trabajo de análisis de datos. Soporta la integración de herramientas de código abierto como *R* y Python.	Procesamiento de datos, minería de texto, y extracción de datos en investigación académica y entornos empresariales.
SPARQL (SPARQL Protocol and RDF Query Language) (Pérez et al., 2009)	SPARQL es el lenguaje de consulta para bases de datos de tripletas RDF (*Resource Description Framework*), lo cual es fundamental para la web semántica	Extracción de información estructurada y conocimiento a partir de datos interrelacionados en RDF, como ontologías y gráficos de conocimiento.
IBM Watson (Ferrucci, 2012)	Watson es una plataforma de inteligencia artificial de IBM que incluye capacidades de procesamiento del NLP, aprendizaje automático y análisis de datos. Watson Discovery es específico para la extracción de conocimiento.	Extracción de información en grandes volúmenes de datos no estructurados, como documentos y transcripciones, utilizado en campos como la medicina y el sector financiero.
Orange (Demšar et al., 2013)	Es una herramienta de minería de datos visual basada en Python. Orange permite crear flujos de trabajo para procesamiento de datos y visualización de información.	Análisis exploratorio de datos, visualización, y minería de datos en áreas de ciencias de la vida, economía, y educación.
Stanford Core NLP (Manning et al., 2014)	*Standford* CoreNLP es una suite de herramientas para el NLP que fue desarrollada por la Universidad de *Stanford*. Proporciona capacidades de análisis gramatical, etiquetado de entidades y análisis de sentimientos.	Procesamiento de textos para la extracción de conocimiento en documentos no estructurados.
Apache Mahout (Withanawasam, 2015)	Apache Mahout es una librería de código abierto para el aprendizaje automático que soporta algoritmos para *clustering*, clasificación y filtrado colaborativo.	Construcción de modelos para minería de datos y recomendaciones; se utiliza en entornos de *Big Data* en combinación con Apache Hadoop.

Finalmente, los Repositorios o Bibliotecas de Activos del Proceso (PAL, por sus siglas en inglés) son colecciones organizadas de documentos, plantillas, guías, herramientas, procedimientos y otros recursos que apoyan a la gestión de procesos y prácticas organizacionales. Estos repositorios sirven para estandarizar, mejorar y facilitar el acceso a la información esencial para la ejecución de proyectos y la mejora de procesos. No obstante, para mejorar la calidad de un proceso en las organizaciones es fundamental almacenar, como activos, el conocimiento que con éste se genera dentro de la organización. De acuerdo con Arcilla-Cobián et al. (2017), un activo es todo elemento

que es utilizado para describir, implementar y mejorar un proceso. Su relevancia fue reconocida en 1993 con la introducción del concepto de "activos del proceso" en el Modelo de Madurez y Capacidad Versión 1.1 (CMMI) y se continuó desarrollando con las siguientes versiones de este modelo hasta llegar a una definición más formal que indica que un activo de un proceso es "cualquier recurso que la organización considere útil para alcanzar los objetivos de un área de proceso". De hecho, la versión más reciente del modelo, CMMI Versión 3.0, especifica que los activos organizacionales de un proceso son "artefactos relacionados con la descripción, implementación y mejora de los procesos". Entre estos activos se incluyen las definiciones de proceso, los modelos de ciclo de vida, las guías para ajustar proyectos y la base de lecciones aprendidas. Por lo tanto, una PAL se define para almacenar y facilitar el acceso a los activos de un proceso que resultan útiles para quienes definen, implementan y gestionan los procesos dentro de la organización (Medina-Domínguez et al., 2008; Bermón-Angarita et al., 2009).

En este sentido, la investigación de Costa et al., (2020) argumenta que una PAL introduce ciertos beneficios para que una organización pueda mejorar considerablemente el rendimiento de un proceso de software, entre las que destacan las siguientes: al definir una PAL se crea un repositorio común de información que fortalece la toma de decisiones, se promueve el concepto de estandarización puesto que todos los proyectos de una organización pueden hacer uso de un proceso estándar a través de la información contenida en el repositorio, y se facilita la documentación de las lecciones aprendidas y, por consecuencia, la identificación de mejoras en un proceso. Por otro lado, esta misma investigación identificó tres problemas con los que es necesario tener cuidado al crear una PAL: no existe una estrategia sistemática que indique cómo estructurar y utilizar una PAL, la existencia de múltiples características que deberán hacerse específicas para definir un proceso (en la PAL) de tal manera que pueda ser utilizado en diferentes dominios de aplicación, y la diversidad de conocimiento y de información que existe sobre un proceso específico dificulta la generación de recomendaciones y buenas prácticas.

Considerando todo lo anteriormente expuesto, en esta tesis se pretende implementar el concepto de una PAL como estrategia para definir, al interior de una pequeña organización desarrolladora de software, un repositorio de conocimientos que permita establecer y facilitar la gestión de las pruebas de regresión con el fin de mejorar los resultados de su uso. Esta estrategia es detallada en las siguientes secciones.

3.5. Definición de una PAL como estrategia para mejorar la gestión de las pruebas de regresión en el software

Dadas las características de las pequeñas organizaciones desarrolladoras de software, una PAL representa la herramienta más factible para definir y mejorar un proceso. En este sentido, se propone que la estrategia que se implemente en este tipo de organizaciones sea de utilidad para:

- Establecer y mantener un conjunto de activos de procesos organizacionales que brinde soporte para la definición de un proceso estándar para la realización y gestión de las pruebas de regresión; creación de directrices y criterios para la adecuación de dicho proceso estándar; y la definición del repositorio organizacional de medidas, de activos y de lecciones aprendidas.
- Establecer procesos de gestión de las pruebas de regresión para cada proyecto de la organización. Estos procesos se adaptan a partir del conjunto organizacional de activos de procesos.
- Establecer un proceso de gestión de pruebas de regresión en cada proyecto mediante su adaptación a partir del conjunto organizacional de procesos estándar.

- Definir prácticas que ayuden a la organización a planificar, implementar y desplegar las pruebas de regresión, considerando al factor humano para lograr el despliegue total.
- Establecer la cultura de gestión y uso de información histórica organizacional que genere información útil para la mejora continua.

Por lo tanto, la estrategia que se propone en el contexto de esta tesis para este tipo de organizaciones se conforma de dos componentes principales: la Biblioteca de Activos del Proceso de Pruebas de Regresión (PAL-RT, por sus siglas en inglés) y una herramienta web de soporte que facilite su uso en un contexto organizacional. A continuación, se detallan ambos componentes.

3.5.1. Biblioteca de activos del proceso de pruebas de regresión (PAL-RT)

La PAL-RT se crea con el propósito de brindar el soporte necesario para representar y gestionar el conocimiento relacionado con el proceso de pruebas de regresión desde un enfoque integrador. Así, la PAL-RT representa el medio necesario para definir, realizar, gestionar y medir la realización de las pruebas de regresión en una pequeña organización, abordando la complejidad intrínseca de este proceso y dando soporte, al mismo tiempo, al personal involucrado en aspectos clave para la realización exitosa de este tipo de pruebas. En la Figura 3.1 se muestra la estructura y los elementos principales de la PAL-RT.

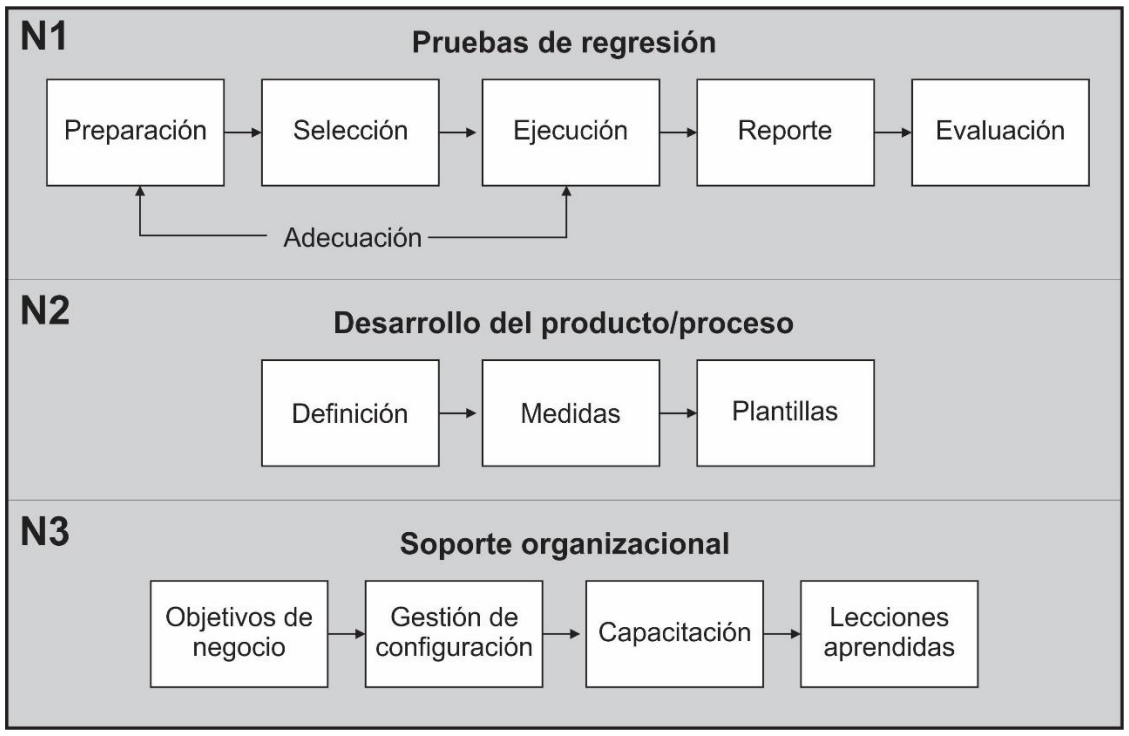


Figura 3.1. Arquitectura de la PAL-RT

La arquitectura conceptual de los metadatos que componen a la PAL-RT se conforma de tres capas de abstracción (i.e., N1, N2 y N3), cuyo fin es representar e integrar a los elementos relacionados con el proceso para realizar las pruebas de regresión y gestionar la información cuantitativa que se genera. La definición de cada capa de abstracción es la siguiente:

- N1 (Nivel 1 – Pruebas de regresión). Esta capa define, a partir del consenso obtenido sobre las respuestas que los profesionales de la industria brindaron en el marco del estudio presentado en el Capítulo 2 de esta tesis, un proceso específico para realizar y gestionar las pruebas de regresión. Es importante hacer notar que este proceso podrá ser modificado para cada proyecto que se realice en la organización, si así fuera necesario.
- N2 (Nivel 2 – Desarrollo del producto/proceso). Esta capa establece básicamente la naturaleza de la PAL, puesto que contiene todos los activos necesarios para realizar eficientemente el proceso definido en la capa N1. Los activos contenidos en N2 pueden utilizarse en cada proyecto de la organización de acuerdo con las guías de adaptación que se proporcionan, o bien pueden ser añadidos y/o modificados de acuerdo con las necesidades de la organización.
- N3 (Nivel 3 – Soporte organizacional). Esta capa delimita el uso de activos para dar soporte organizacional al proceso de pruebas de regresión establecido en N1 y N2. De esta manera, se pretende que la realización de estas pruebas se alinee con el(los) objetivo(s) de negocio establecido(s) por la organización, además de que se promueve la recolección y documentación de lecciones aprendidas.

La integración que se busca en N2 se lleva a cabo mediante el concepto de *patrón*, el cual representa al elemento crucial para establecer toda la información requerida en la definición de un proceso de pruebas de regresión (e.g., fases, actividades, productos, medidas). En este sentido, de acuerdo con la experiencia de las organizaciones desarrolladoras de software que cuentan con niveles avanzados de madurez/capacidad, la incorporación de este tipo de arquitectura permite mantener el conocimiento bien estructurado y organizado y, como consecuencia, facilita la incorporación de una cultura de trabajo centrada en la madurez de los procesos.

De acuerdo con Chen y Lee (2022), al seguir esta estrategia se le está proporcionando a una organización pequeña el conocimiento esencial para definir, establecer y difundir su proceso de pruebas de regresión. Así pues, la PAL-RT es un repositorio común de datos (i.e., activos) que permite la estandarización de procesos dado que todos los proyectos pueden utilizar el mismo proceso estándar o diferentes adaptaciones aprobadas de éste. Además, para cada proyecto que se realice dentro de la organización, se podrá definir, a partir de la PAL-RT, un proceso específico que considera los criterios y pautas de adaptación. De esta manera, el proceso que se defina en la organización para realizar y gestionar las pruebas de regresión podrá ser utilizado en diferentes proyectos promoviendo así la actualización del conocimiento contenido en la PAL-RT en dos niveles: (i) a nivel de proceso, producto y tarea con medidas reales, y (ii) a nivel de mejora con lecciones aprendidas y las mejoras identificadas.

Por lo tanto, la PAL-RT almacena las mediciones recogidas antes, durante y después de la realización de las pruebas de regresión para todos los proyectos que se lleven a cabo. Si se considera que cada proyecto debe seguir un proceso estándar definido, otros diferentes proyectos pueden utilizar las mismas actividades o crear diferentes, mejorando así el conocimiento de la biblioteca, y realizar análisis estadísticos para mejorar la efectividad de las pruebas de regresión con los datos históricos que se vayan recopilando. Considerando esta estrategia, el proceso estándar para las pruebas de regresión que se encuentra contenido en la PAL-RT se define en términos de actividades, productos, medidas (de proceso, actividades y productos) y activos (también de proceso, actividades y productos). Todos estos activos son artefactos o mecanismos que brindan soporte para realizar el proceso de pruebas de regresión, llevar a cabo las actividades, obtener los productos, y recoger la información relacionada con las medidas (véase la Figura 3.2). La figura muestra que, siguiendo esta estrategia, la realización y gestión de las pruebas de regresión parte de la descripción de un *proceso*

estándar de la organización que puede ser ajustado para diferentes *proyectos* de una organización, teniendo en cuenta pautas y criterios de adaptación (i.e., patrones). En el contexto de la Ingeniería de Software, los *patrones* representan soluciones que se basan en la experiencia de profesionales y/o especialistas y que han sido probadas en repetidas ocasiones, con buenos resultados, a lo largo del desarrollo de software (Riehle y Züllighoven, 1996). En este sentido, los patrones deben ser correctamente documentados para que puedan ser utilizados en los diferentes proyectos que una organización puede llevar a cabo. Considerando la presente tesis, los patrones se generan en forma de *procesos* que representan instancias del proceso estándar de la organización para las pruebas de regresión. Por otro lado, las *actividades* son los pasos que definen a un proceso de pruebas de regresión, mientras que los *productos* son los entregables que se generan en una o más actividades descritas por el proceso como resultado de su ejecución. Los *activos* son artefactos o mecanismos que proporcionan un soporte a los usuarios para la correcta realización de las actividades; es decir, cualquier plantilla, guía o recomendación que facilite el entendimiento y realización de una actividad. Finalmente, las *medidas estándar* son indicadores cuantitativos del proceso que facilitan la toma de decisiones en la organización.

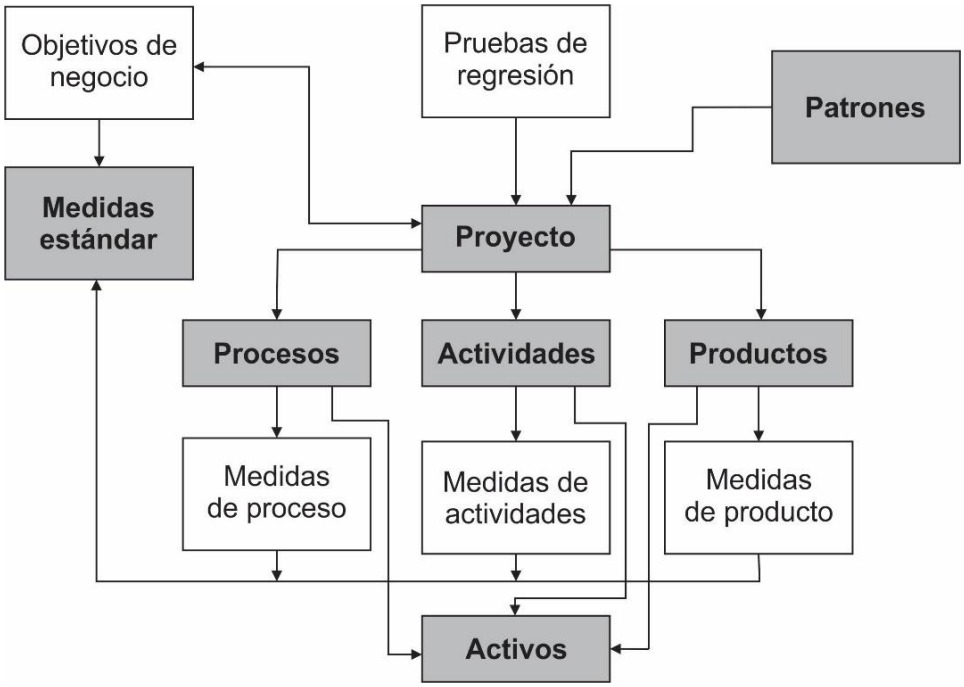


Figura 3.2. Esquema detallado de la PAL-RT

La Figura 3.3 describe la manera en que la estrategia propuesta en esta tesis hace uso de la PAL-RT para generar una cultura de trabajo diferente en el contexto de una pequeña organización. En esta figura, los procesos que son definidos a partir del proceso estándar para las pruebas de regresión (ya contenido en la biblioteca) se adaptarán a cada proyecto en función de los activos específicos que sean seleccionados. En este sentido, los procesos que se adapten a cada proyecto se podrán diferenciar dependiendo de los activos escogidos de la siguiente manera:

A Cualquier cambio que se realice al código fuente requiere de un análisis cuidadoso para realizar las pruebas de regresión. Dicho análisis se realiza en la fase de Preparación, donde los cambios son documentados y gestionados mediante el control de versiones. De esta manera, la PAL-RT facilita esta actividad puesto que simplifica la planificación y el seguimiento y control de los cambios que deberán ser probados en la correspondiente regresión.

B Posteriormente, en las fases de Selección, Ejecución, Reporte y Evaluación se definen y utilizan los casos de prueba que deberán abordar los cambios realizados sobre el código. Posteriormente y con la información recogida en el proyecto, estos casos de prueba pueden ser priorizados y/o categorizados dependiendo su desempeño en el proyecto. Esta información será útil para que la organización mejore su madurez en base a la experiencia, puesto que aquellos casos de prueba que conduzcan a una buena detección de defectos y/o fallos serán identificados y, como consecuencia, representarán a un conjunto ideal para la reutilización.

C Dado que la PAL-RT introduce la definición y gestión del conocimiento generado durante la realización de las pruebas de regresión, se requiere que la organización siga utilizando las herramientas de soporte (manuales o automatizadas) con las que habitualmente realiza este tipo de pruebas.

D Uno de los propósitos fundamentales de la PAL-RT es que los activos que la conforman evolucionen a través del uso repetido del proceso de pruebas de regresión. Los activos que se actualicen con el tiempo determinarán la calidad de dicho proceso. Es importante recalcar que la estructura con la que fue diseñada la PAL-RT incorpora un conjunto base de activos que deberá evolucionar con el paso del tiempo.

E La evolución de los activos conducirá a la creación de nuevos procesos de pruebas de regresión a partir del proceso organizacional (i.e., proceso estándar de la organización), el cual servirá como guía. Así, cada proyecto que involucra a los activos e información contenida en la PAL-RT generará un conjunto de lecciones aprendidas que permitirá la mejora continua de la organización.

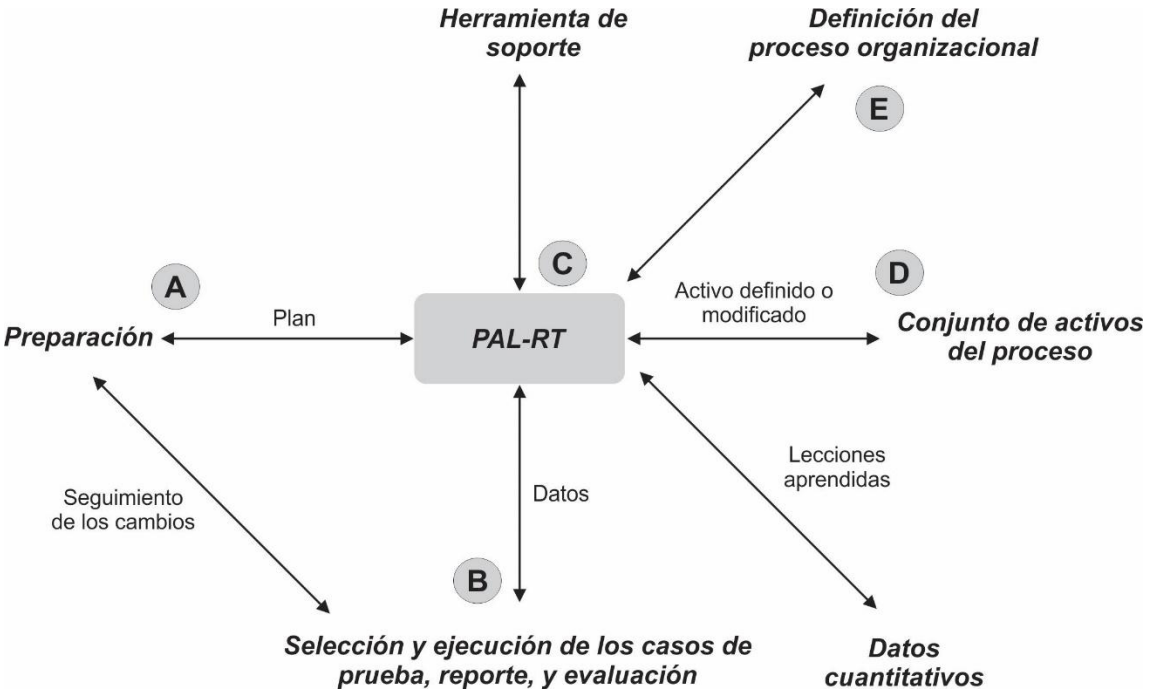
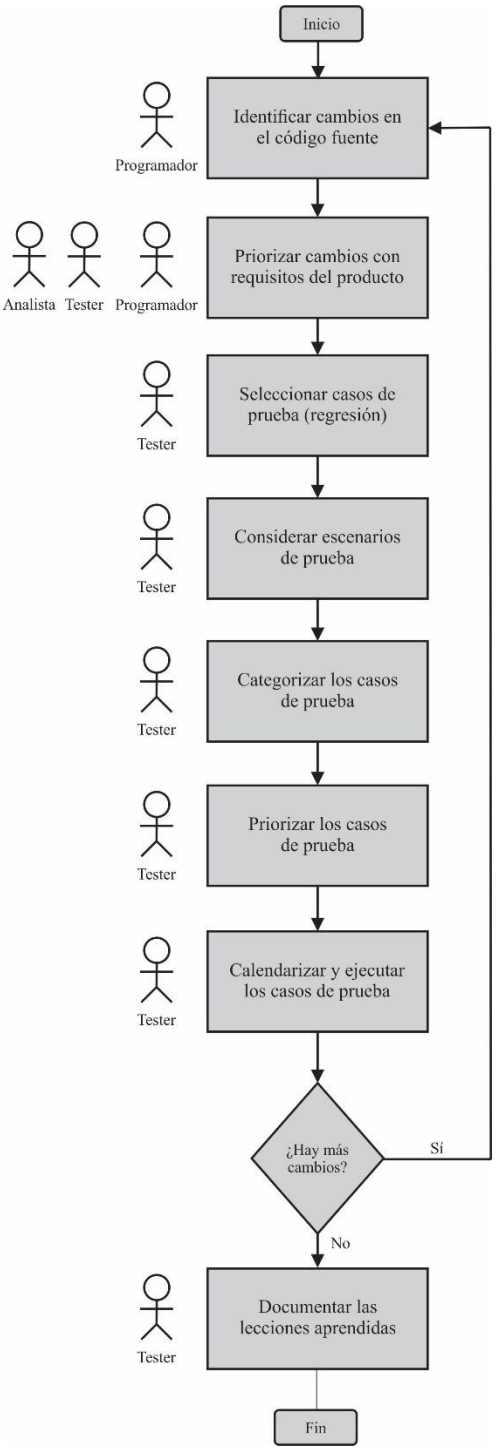


Figura 3.3. Ciclo de trabajo de la PAL-RT

3.5.2. Actividades para realizar y gestionar las pruebas de regresión

A través del consenso entre las opiniones de los 135 participantes en el estudio presentado en el Capítulo 2, se propone un proceso definido por ocho fases principales que se resumen a continuación:



El proceso inicia con la identificación de los cambios que se hayan realizado al código fuente, lo cual se denomina formalmente como "regresión". Esta identificación la deberá hacer el *programador* con el objetivo de que se realicen las pruebas necesarias que eviten la aparición posterior de defectos que puedan conducir a fallos en el software.

Posteriormente, el *analista*, *tester* y *programador* deberán analizar los cambios que se hicieron al código con la intención de identificar qué requisitos del producto están involucrados en la modificación a fin de priorizar qué se probará primero.

Una vez que la priorización de los cambios sobre los requisitos se ha llevado a cabo, el *tester* deberá seleccionar qué casos de prueba probará nuevamente o bien si es necesario crear nuevos casos de prueba para tener mayor cobertura.

Con la intención de formalizar la prueba, el *tester* deberá documentar descripciones de los casos de prueba (escenarios) que le permitan validar que el software cumple con los requisitos tal y como se espera.

Dado que la tesis se enfoca en el desarrollo de software dentro del ciclo de vida de software, el *tester* podrá realizar pruebas de sistema y/o pruebas de aceptación, por lo que deberá decidir si los casos de prueba corresponden a pruebas funcionales o no funcionales, y si se llevarán a cabo de forma manual o automatizada.

Una vez que se tenga la categorización de los casos de prueba, el *tester* deberá seleccionar y priorizar aquellos casos que serán ejecutados con el fin de gestionar el avance sobre la profundidad de las pruebas.

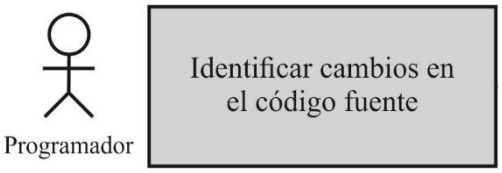
Posteriormente, el *tester* planificará la ejecución de los casos de prueba seleccionados con el fin de gestionar el avance de sus actividades. Es importante considerar una vez más que, dado que las pruebas se contextualizan al desarrollo de software, esta planificación será independiente del plan de proyecto original.

Una vez que el *tester* haya agotado toda la lista de casos de prueba, deberá evaluar si existen cambios que se hayan agregado para abordarlos de nuevo. En caso contrario, el producto se libera y se procede a cerrar las pruebas de regresión para ese producto en específico.

Con el fin de proceder al cierre del proceso, el *tester* deberá documentar las lecciones aprendidas con el fin de contribuir a la generación de conocimiento que ayude a la organización a no cometer errores innecesarios.

A continuación, se proporciona una descripción detallada de cada una de estas fases y, al mismo tiempo, se presenta el activo o activos creados para soportar su correcta ejecución en el contexto de una pequeña organización desarrolladora de software.

3.5.2.1. La importancia de documentar los cambios en el software



Para definir correcta y eficientemente un proceso para gestionar las pruebas de regresión, en el contexto de una pequeña organización, es necesario que se cuente con un mecanismo que permita al *programador* documentar los cambios que haya realizado al código fuente. De esta manera se podrá controlar el impacto que tiene dicho cambio en los requisitos funcionales del producto que se encuentra en etapa de desarrollo o construcción. En este sentido, esta fase está conformada por las actividades mostradas en la Figura 3.4.

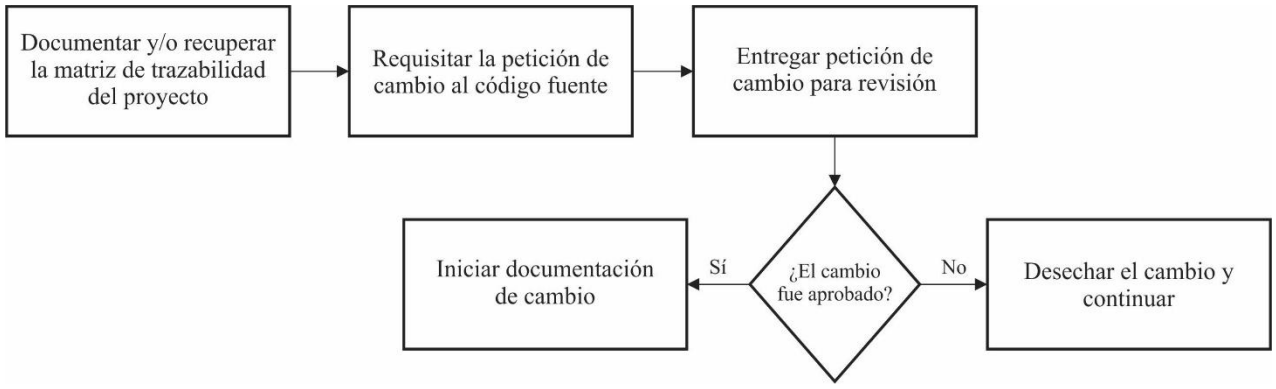


Figura 3.4. Actividades relacionadas con la fase "Identificar cambios en el código fuente"

Como se ha mencionado en repetidas ocasiones a lo largo de esta tesis, uno de los problemas importantes en las pequeñas organizaciones desarrolladoras de software es la falta de información histórica. De hecho, de acuerdo con Pardo (2021), es común que las decisiones que se toman en este tipo de organizaciones se basen en la experiencia previa que la mayoría de las veces suele estar poco o nada relacionada con el contexto de un proyecto específico. Por lo tanto, con estas actividades se pretende inculcar en estas organizaciones la cultura de documentación eficiente con el fin de crear un acervo digital de información histórica que no solamente les permita generar una base reutilizable de conocimiento, sino que también les facilite la mejora continua de sus prácticas cotidianas y, como consecuencia, el madurar sus procesos de software.

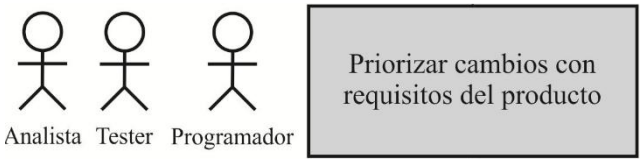
En este sentido, la primera actividad implica que la organización defina y mantenga actualizada una matriz de trazabilidad para el proyecto con el fin de que las modificaciones que se realicen sobre el código estén documentadas. Con este objetivo en mente se propone el activo MATRAZ (véase la Figura 3.5). Duraisamy y Atan (2013) consideran que la matriz de trazabilidad es una tabla que captura los requisitos completos del usuario y del sistema y que, por ende, ayuda a "rastrear" desde la fuente de un requisito hasta la prueba que verifique que éste sea cumplido. En el contexto de esta tesis, el activo MATRAZ deberá tenerse listo una vez que se culmine la tarea de especificación de los requisitos, por lo que se estaría sugiriendo con esto que el proceso de la pequeña organización también se modifique para incluir dicho activo en la fase de requisitos. Por consiguiente, en el contexto de esta tesis, el activo MATRAZ sería utilizado por el programador para actualizar la información que ahí se encuentra en relación con el cambio que se pretende hacer al código. De esta

manera, se pretende controlar dos cosas: (1) identificar qué requisitos funcionales están involucrados en dicho cambio y que por ende se pueden ver afectados y (2) identificar cuáles son los casos de prueba que fueron diseñados para probar la funcionalidad, puesto que es bastante probable que, además de que crear nuevos casos de prueba, el *tester* deba volverlos a ejecutar. Es importante mencionar que cada activo propuesto en esta tesis se acompaña de una guía de adaptación que establece el procedimiento que se deberá seguir para el correcto llenado de la información. Para el caso del activo MATRAZ, este procedimiento es el establecido por la Tabla 21.

Posteriormente, antes de que el *programador* modifique el código deberá realizar una petición de cambio con el fin de que ésta sea evaluada por el *jefe de proyectos* y se tome una decisión informada. Una vez más, considerando la información recogida con el estudio realizado sobre las 34 organizaciones desarrolladoras de software, uno de los problemas que enfrentan este tipo de empresas es que precisamente no se tiene un control sobre las modificaciones que se realizan en el código, ya sea durante la fase de codificación del software o en la de mantenimiento cuando se pretende hacer mejoras al producto. Aunado a esto, se presenta el problema de que no se realizan pruebas sobre el código que se modifica y, además, se asume erróneamente que los casos de prueba que ya habían sido probados, antes de cualquier modificación, seguirán aprobando a pesar de los cambios que se hayan realizado al código, por lo cual no se vuelven a probar. Esta situación no solamente genera un error importante en la modificación de un producto de software, sino que también incrementa el costo y tiempo de entrega de la mejora. Por lo tanto, se pretende que cada cambio que se realice al código fuente sea documentado y fácil de rastrear mediante el activo SOLCA (véase la Figura 3.6), de esta manera se establece un proceso de revisión/autorización para que se gestione correctamente cada modificación que se pretenda hacer al producto de software. Así, con el uso de este activo se propone un medio que le permita al *programador* proponer modificaciones al código del software a través de solicitudes que serán revisadas y, en el mejor de los casos, aprobadas por el *jefe de proyectos*.

Una vez que el activo SOLCA ha sido revisado por el *jefe de proyectos*, se toma la decisión de aceptar o rechazar la solicitud de cambio sobre el código fuente del producto de software. En caso de que la solicitud sea válida, se comienza con la documentación del cambio con el fin de que éste sea colocado en una pila priorizada de funcionalidades por probar (i.e., la regresión en sí). En caso contrario la solicitud se rechaza y no se autoriza la modificación al código.

3.5.2.2. La priorización de los requisitos sujetos a prueba



Considerando que los cambios solicitados por el *programador* hayan sido aprobados por el jefe de *proyectos*, es necesario que se haga un análisis específico de aquellos requisitos funcionales que se verán afectados por las modificaciones que se realicen sobre el código fuente. De esta manera, se tendrá un conocimiento más específico sobre qué debe probarse primero con el fin de evitar que la regresión ponga en riesgo la funcionalidad general del producto. Por lo tanto, esta actividad involucra tanto al *analista* como al *tester* y al *jefe de proyectos*, puesto que es probable que la modificación realizada origine una modificación tanto en los requisitos del producto de software como en los casos de prueba que fueron creados durante el desarrollo de éste, lo que conducirá a que se incremente la cantidad de casos de prueba en el repositorio del proyecto.

Información de la matriz de trazabilidad: Activo MATRAZ																								
Nombre del proyecto:			Requisitos			Plan	Diseño		Creación	Pruebas	Mantto.													
Descripción del proyecto:			ID	Fuente	# de RF	Descripción del requisito	Caso de uso	Entregable WBS	Clase	Tabla	Código	Caso de prueba	# de cambio											

Figura 3.5. Formulario para el activo MATRAZ

Tabla 21. Instrucciones para el llenado del activo MATRAZ

Nombre del proyecto	Nombre que le fue asignado al proyecto desde su inicio. En caso de que no exista un nombre formal se puede utilizar el código o algún identificador que le haya sido asignado durante la planificación del proyecto.
Descripción del proyecto	Descripción detallada del proyecto que suele incluir al objetivo de negocio que el o los clientes pretenden alcanzar con su desarrollo. Muchas veces esta descripción suele darse en términos de la funcionalidad general que se verá reflejada en el producto (por ejemplo "Se realizará un punto de venta que incluye a una pistola de código de barras y una impresora térmica. No se incluye la gestión de proveedores".
ID	Identificador único que recibe cada elemento dentro de la matriz de trazabilidad. Se sugiere utilizar números enteros consecutivos a partir del número 1.
Fuente	El origen del deseo o necesidad proporcionado por un *stakeholder* y que condujo a la definición de un comportamiento específico del software o mejor conocido como el "requisito funcional". Ejemplos de *stakeholders* son los usuarios finales, el cliente, un software anterior, manuales de procedimientos, reglamentos, etc.
# de RF	Una clave o identificador único asignado por el *analista* a cada requisito funcional que se encuentra en el documento de Especificación de Requisitos. Por ejemplo, en algunos casos se sugiere utilizar un identificador como RF 1.1, con lo que se describiría que se trata del requisito funcional 1 (número a la derecha del punto decimal) de un caso de uso 1 (número a la izquierda del punto decimal).
Descripción del requisito	Definición detallada que el *analista* dio al requisito funcional para hacerlo específico, medible, alcanzable y relevante. Es importante considerar que esta descripción debe abordar las acciones que el software realiza para cumplir con las necesidades de los *stakeholders*.
Caso de uso	Identificador único que el *analista* asigna a cada caso de uso en el documento de Especificación de Requisitos. Por ejemplo, es común que se utilice un identificador del tipo CU 1.1 para indicar que se trata de un caso específico de uso 1 (número a la derecha del punto decimal) de un caso de uso más general 1 (número a la izquierda del punto decimal).
Entregable de la WBS	Identificador o nombre del entregable comprometido por el *jefe de proyectos* para una actividad específica dentro de la WBS cuya generación puede verse afectada por algún cambio, ya sea en los requisitos funcionales o en el código ya escrito.
Clase	Nombre o identificador de la clase que se ha generado como parte del diseño de bajo nivel y a partir de uno o más requisitos funcionales.
Tabla	Nombre o identificador de la tabla, dentro de la base de datos, que se ha generado como parte del diseño de alto nivel y a partir de uno o más requisitos funcionales.
Código	Líneas o bloques de código que el *programador* ha implementado para cubrir uno o más requisitos funcionales.
Caso de prueba	Identificador único que se le asigna a cada caso de prueba en el documento de Plan de Pruebas. Por ejemplo, es común que se utilice un identificador del tipo CP-001 considerando utilizar números enteros consecutivos a partir del número 1.
# de cambio	Identificador único que se le asigna dentro de la Solicitud de Cambios a cada cambio requerido por el programador.

La fase para priorizar los cambios considerando los requisitos funcionales involucrados se compone de las actividades mostradas en la Figura 3.7.

Solicitud de cambios: Activo SOLCA					
			Fecha:		
Nombre del producto:			**Referencia:**		
Nombre del proyecto:					
Nombre del jefe de proyecto:					
Nombre del programador (quien solicita el cambio):					
Descripción del cambio solicitado:					
Justificación del cambio solicitado:					
Análisis de impacto					
Cronograma estimado para la mejora del producto:					
Alcance (funcionalidad del producto):					
Presupuesto destinado a la mejora del producto:					
¿El cambio solicitado está justificado para llevarse a cabo?		**Sí**		**No**	
Justificación:					
Aprobación de la solicitud					
Fecha de aprobación/rechazo:					
Nombre y firma del jefe de proyecto		**Nombre y firma del programador solicitante**			
Nombre		**Nombre**			

Figura 3.6. Formulario para el activo SOLCA

Tabla 22. Instrucciones para el llenado del activo SOLCA

Fecha	La fecha en que el programador entrega la solicitud de cambio.
Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Referencia	Código o identificador que suele asignar el *jefe de proyectos* al producto creado.
Nombre del proyecto	Nombre que le fue asignado al proyecto desde su inicio. En caso de que no exista un nombre formal se puede utilizar el código o algún identificador que le haya sido asignado durante la planificación del proyecto.
Nombre del programador	Nombre de la persona, con el rol de *programador*, que está solicitando la autorización para la modificación del código fuente del producto.
Descripción del cambio solicitado	Descripción detallada del cambio que el *programador* está considerando realizar con el código fuente para mejorar la funcionalidad del producto. La mayoría de las veces esta descripción suele hacer referencia a las funcionalidades que serán agregadas y/o modificadas con el fin de generar una nueva versión del producto.
Justificación del cambio solicitado	Descripción detallada de las razones por las cuales el *programador* considera necesario modificar el código. Esta justificación suele darse en función de la mejora que se pretende realizar e, incluso, suele proporcionarse información sobre qué elementos, relacionados con el código, de la matriz de trazabilidad están involucrados en el cambio.
Cronograma estimado para la mejora del producto	Análisis realizado por el *jefe de proyectos* para determinar el impacto que tendrá el o los cambios solicitados por el *programador* sobre el cronograma (tiempo) estimado para realizar la mejora del producto.
Alcance	Análisis realizado por el *jefe de proyecto* para determinar el impacto que tendrá el o los cambios solicitados por el *programador* en los requisitos funcionales que están reflejados en el producto de software. A menudo suele indicarse, a partir de la matriz de trazabilidad, cuáles son los requisitos funcionales que están involucrados en la modificación solicitada por el *programador*. De esta manera se determina si existe algún riesgo de afectar la funcionalidad general del producto.
Presupuesto destinado a la mejora del producto	Análisis realizado por el *jefe de proyectos* para determinar el impacto que tendrá el o los cambios solicitados por el *programador* en el presupuesto asignado para la mejora del producto.
Justificación	Resultado del análisis realizado por el *jefe de proyectos* sobre la solicitud de cambio hecha por el programador. Es común que la decisión tomada por el *jefe de proyectos* se base en información objetiva y no provenga de un juicio de valor.
Fecha de aprobación/rechazo	La fecha en que el *jefe de proyectos* aprueba o rechaza la solicitud de cambio del *programador*.

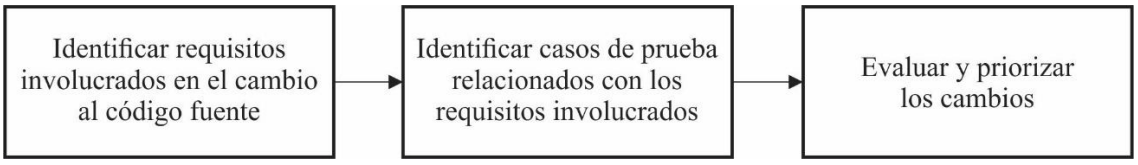


Figura 3.7. Actividades relacionadas con la fase "Priorizar cambios con requisitos del producto"

La correcta ejecución de las actividades dependerá del uso del activo MATRAZ y de un nuevo activo que es la documentación del caso de prueba (activo DOCP), el cual se muestra en la Figura 3.8. El procedimiento de llenado de este nuevo activo es el mostrado en la Tabla 23.

Descripción del caso de prueba: Activo DOCP											
Nombre del producto:							**Fecha:**				
ID del caso de prueba:				**Nombre del *tester* que diseñó el caso de prueba:**							
Entorno de prueba:					**Requisitos funcionales involucrados:**						
Propósito del caso de prueba:											
Prioridad del caso de prueba											
Probabilidad de ocurrencia:	Frecuente		Posible		Ocasional		Remota		Improbable		
Impacto:	Catastrófico		Crítico		Marginal		Improbable				

Descripción de las acciones y/o condiciones de prueba			
#	**Acciones**	**Salida esperada**	**Salida real**

Resultado											
Resultado de la prueba:	Aprobado		No aprobado		No aplica						
Seguimiento:	Sí		No		**Severidad:**	Alta		Seria		Media		Baja	
Nombre y firma del *tester* que ejecutó el caso de prueba:						**Nombre y firma**					

Figura 3.8. Formulario para el activo DOCP

Tabla 23. Instrucciones para el llenado del activo DOCP

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el *tester* está documentando el caso de prueba.
ID del caso de prueba	Código o identificador que suele asignar el *tester* al caso de prueba que está diseñando. En la mayoría de los casos se sugiere usar un identificador tipo CP-001, para denotar que se trata del caso de prueba 1 que se asocia con el producto.
Nombre del *tester* que diseñó el caso de prueba	Nombre de la persona que funge como el *tester* que está diseñando el caso de prueba para su posterior ejecución.
Entorno de prueba	Configuración requerida para llevar a cabo la ejecución del caso de prueba que suele incluir hardware, software, configuraciones de red, datos requeridos para realizar la prueba. Puede indicarse como en desarrollo, pruebas, puesta en escena, o producción.
Requisitos funcionales involucrados	Requisitos funcionales, tomados de la matriz de trazabilidad (activo MATRAZ), que se relacionan con el caso de prueba. La identificación de estos requisitos es fundamental para el éxito de la prueba puesto que se controla el impacto de los cambios que se realicen al producto como resultado negativo en la ejecución de un caso de prueba.
Propósito del caso de prueba	Descripción de la funcionalidad que se pretende validar contra los requisitos funcionales involucrados en la definición del caso de prueba. Por ejemplo: “Se pretende probar que el inicio de sesión a la aplicación se realiza correctamente”.
Probabilidad de ocurrencia	Posibilidad de que se produzca un problema potencial en las funcionalidades del software derivado de no aprobar el caso de prueba en cuestión. Para mejorar la priorización de los casos de prueba, el riesgo de ocurrencia debe ser valorado por el *tester* como frecuente, posible, ocasional, remota o improbable.
Impacto	Resultado, efecto o consecuencia que está en función de la probabilidad de ocurrencia. Para mejorar la priorización de los casos de prueba, el impacto debe ser valorado por el *tester* como catastrófico, crítico, marginal o improbable.
#	Número secuencial asignado a cada acción utilizada para describir las acciones que se realizarán para ejecutar el caso de prueba.
Acciones	Descripción puntual de los pasos que el *tester* deberá seguir para probar las funcionalidades del producto de software que están asociadas al caso de prueba.
Salida esperada	Resultado que se espera obtener después de ejecutar cada acción del caso de prueba.
Salida real	Resultado que se obtiene después de ejecutar cada acción del caso de prueba.
Resultado de la prueba	Información obtenida por el *tester* que le permite determinar si las funcionalidades probadas, a través de la ejecución del caso de prueba, se comportan correctamente. El *tester* debe considerar dos cosas: (1) que el comportamiento se alinee con lo establecido por los requisitos funcionales asociados al caso de prueba, y (2) la correspondencia entre la salida esperada y la real. Debe indicarse como aprobado o no aprobado.
Seguimiento	Dependiendo de la severidad de los fallos detectados como consecuencia de ejecutar el caso de prueba, el *tester* de indicar si se requiere o no la monitorización del comportamiento del producto.
Severidad	Impacto del fallo (o fallos) detectado(s) sobre la interacción con el usuario final y/o negocio. La severidad puede ser valorada por el *tester* como alta, seria, media o baja.
Nombre y firma del *tester* que ejecutó el caso de prueba	Nombre y firma de la persona que realizó la ejecución del caso de prueba y reporta los hallazgos.

Es importante mencionar que, en caso de que la pequeña organización no documente los casos de prueba, el activo DOCP será el mecanismo principal para hacerlo con el fin de que sea posible la reutilización y/o creación de los casos de prueba para cada proyecto. Como se mencionó anteriormente, con esto se pretende inculcar en la organización la práctica de identificar, priorizar y controlar la evolución de los requisitos que serán modificados con los cambios y, posteriormente, probados a lo largo de las regresiones. Por otro lado, si la organización ya contara con un mecanismo para documentar los casos de prueba, se sugiere que éste se siga utilizando con una pequeña modificación para que sea posible agregar y relacionar los requisitos funcionales de la matriz de trazabilidad que se relacionan con cada caso de prueba.

Un error común en las pequeñas organizaciones es determinar correctamente la probabilidad de ocurrencia y el impacto asociado con fallos al implementar los requisitos funcionales del producto de software. En este sentido, la elección correcta de ambas variables dependerá en gran medida del análisis cualitativo y cuantitativo que el *tester* realice sobre los fallos que se pueden presentar con los cambios realizado al código, lo cual depende en gran medida de su experiencia. En el contexto de la estrategia definida en esta tesis, se sugiere que el *tester* considere el análisis resumido en la Figura 3.9, el cual considera la severidad detectada en los fallos observados una vez que se tenga el resultado de haber ejecutado el caso de prueba. Dicha severidad se denota con un formato parecido a los colores de un semáforo tradicional. De esta manera, si se trata, por ejemplo, de un fallo que puede impactar negativamente tanto al producto de software como a los objetivos de negocio, deberá realizarse su corrección a la brevedad. Considerando lo anterior, con el uso de los activos y un análisis basado en la severidad de los fallos detectados en los casos de prueba, el *tester* podrá identificar qué requisitos funcionales del activo MATRAZ deben priorizarse para modificación con el fin de reducir o eliminar la severidad de los fallos detectados.

------Probabilidad de ocurrencia------ \ ----------Impacto----------	**Catastrófico (4)**	**Crítico (3)**	**Marginal (2)**	**Insignificante (1)**
Frecuente (5)	Alta (20)	Alta (15)	Seria (10)	Media (5)
Posible (4)	Alta (16)	Alta (12)	Seria (8)	Media (4)
Ocasional (3)	Alta (12)	Seria (9)	Media (6)	Baja (3)
Remota (2)	Seria (8)	Media (6)	Media (4)	Baja (2)
Improbable (1)	Media (4)	Baja (3)	Baja (2)	Baja (1)

Figura 3.9. Matriz de severidad para determinar la probabilidad de ocurrencia e impacto de un fallo

La matriz de severidad le proporcionará al *tester* una visión rápida de la importancia de los fallos y la prioridad con la que cada uno debe ser abordado. La probabilidad de ocurrencia se da en términos de un porcentaje de tal manera que la elección de un valor subjetivo corresponda con otro cuantitativo. Es decir, el *tester* deberá considerar que la probabilidad se da en función de los siguientes valores:

- Frecuente: Se espera que el fallo ocurra en la mayoría de las ocasiones (91%-100%).
- Posible: Es posible que el fallo ocurra en varias ocasiones (61%-90%).

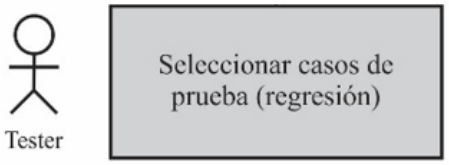
- Ocasional: Puede que el fallo ocurra en alguna ocasión (41-60%).
- Remoto: Existen pocas posibilidades de que el fallo ocurra en algún momento (11%-40%).
- Improbable: El fallo, si es que ocurre, será bajo condiciones raras y excepcionales (0%-10%).

Por otro lado, el impacto es la gravedad con la que el fallo puede golpear al software en sí, que puede ir desde que éste no responda hasta la pérdida de información. El impacto se puede evaluar considerando cuatro valores:

- Catastrófico: Las consecuencias del fallo, si no se corrige, son desastrosas a tal grado de que existe la posibilidad de que el producto fracase. Se le asigna un valor numérico de 4.
- Crítico: El fallo genera consecuencias graves que pueden originar graves pérdidas. El producto se ve seriamente amenazado. Se le asigna un valor de 3.
- Marginal: Los daños causados por el fallo pueden revertirse mediante modificaciones rápidas al código. Se le asigna un valor de 2.
- Insignificante: El fallo causa un mínimo daño que puede ser controlado y gestionarse como un remanente en el código. Se le asigna un valor de 1.

Considerando lo anterior, el *tester* utilizará la severidad como indicador de prioridad mediante los siguientes cuatro niveles: alta (i.e., se requieren acciones inmediatas para corregir el fallo detectado, se rastreará hacia los requisitos implementados), seria (i.e., se detiene el avance de las regresiones para aislar o eliminar el fallo, se rastreará hacia los requisitos implementados), media (i.e., se deben tomar medidas razonables y seguimiento para controlar el fallo, no se rastreará hacia los requisitos implementados), o baja (i.e., se requiere de la revisión periódica para garantizar que todo funciona bien, no es necesario el rastreo).

3.5.2.3. La selección de los casos de prueba más importantes



Una vez que se ha determinado la severidad (o criticidad) de los fallos relacionados con los requisitos funcionales que están involucrados en la modificación al código, es necesario que el *tester* seleccione del repositorio del proyecto aquellos casos de prueba que se relacionan directamente con la regresión. Con el objetivo de no complicar este proceso a las pequeñas organizaciones, se plantea que la selección de los casos de prueba se base en dos aspectos fundamentales: (1) la severidad de los fallos relacionados con los requisitos funcionales involucrados en las modificaciones al código y (2) la repetitividad de un fallo en los casos de prueba. Con relación al segundo aspecto, se ha considerado la definición de una categoría para los fallos que pueden originarse durante la ejecución de los casos de prueba con el fin de que el *tester* de la organización pueda documentarlos y, como consecuencia, sea capaz de determinar la cantidad de veces que un fallo está repitiéndose en los casos de prueba. Así, la categorización mostrada en la Tabla 24 considera únicamente los fallos que se pueden detectar mediante la realización de pruebas funcionales al código, módulo o software sometido a regresión.

Tabla 24. Categorización para fallos en la ejecución de casos de prueba

Fallo	Descripción
Ejecución	Se genera cuando un software se bloquea o genera una salida incorrecta durante su ejecución a causa de errores en el código, *add-ons* o extensiones (e.g., un software que se bloquea al momento de usar una extensión para calcular el pago de impuestos).
Flujo de trabajo	Se asocia con la navegación deficiente del usuario en el software (e.g., cuellos de botella que originan retrasos, confusión y frustración).
Fuera de límites	Se produce cuando un usuario interactúa con la interfaz del software de forma no deseada a causa de errores lógicos y aritméticos que exceden los límites permitidos para una operación específica (e.g., un usuario escoge incorrectamente una fecha posterior al día actual para generar una cita médica y el software agenda la cita pero con la fecha actual, es decir, una fecha fuera del rango aceptable).
Funcional	Modifica la funcionalidad prevista del software (e.g., cálculos incorrectos, comportamientos inesperados, bloqueos o funcionalidades que no responden).
Integración	Surge cuando diferentes componentes o subsistemas de un software no logran funcionar juntos sin problemas (e.g., una aplicación web para la cual diferentes equipos desarrollaron módulos como “registro de usuario”, “inicio de sesión”, “pago” y “procesamiento de pagos”; sin embargo, al ser integrados para que la aplicación funcione correctamente se genera un error cuando el usuario hace clic en “proceder al pago”, lo que significa que el software falló en la integración del pago).
Lógico	Interrumpe el flujo de trabajo previsto para el software puesto que el software se ejecuta sin fallar, pero produce resultados incorrectos (e.g., una página web de comercio electrónico donde el botón “agregar al carrito” redirige al usuario al catálogo de productos en lugar de a la página de pago).
Memoria	Se genera cuando no existe suficiente memoria para realizar una operación mientras el software está en ejecución (e.g., el administrador de memoria de Windows reporta un error de corrupción mientras el software se ejecuta: FAULTY_HARDWARE_CORRUPTED_PAGE).
Rendimiento	Se presenta cuando el software se comporta incorrectamente dado que se afecta su velocidad, estabilidad y capacidad de respuesta (e.g., una página web tarda más de lo esperado en cargarse, especialmente durante los picos de tráfico).
Seguridad	Resulta a causa de ataques de seguridad derivados de las modificaciones incorrectas al código (e.g., el usuario introduce credenciales incorrectas en el inicio de sesión y se le permite el acceso, lo cual genera una brecha de seguridad puesto que se corre el riesgo de que cualquiera pueda acceder a la aplicación y a la información).
Sintaxis	Se presenta cuando, de manera similar que un fallo lógico, el software se ejecuta sin fallar pero existen errores tipográficos y/o ortográficos en las interfaces (e.g., una interfaz con múltiples errores ortográficos).
Unitario	Se dan a nivel de módulo, función o declaración en el código (e.g., dar clic en un botón de la interfaz que no conduce a ninguna acción).
Usabilidad	Obstaculiza la capacidad de un usuario para interactuar con el software (e.g., diseños desordenados, demasiados submenús o enlaces que se abren en nuevas ventanas).

La Figura 3.10 muestra las actividades que deben realizarse para seleccionar correctamente los casos de prueba que serán ejecutados en la regresión.

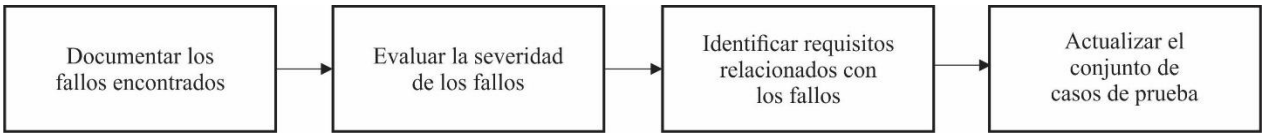


Figura 3.10. Actividades relacionadas con la fase "Seleccionar los casos de prueba"

Para iniciar con la realización de las actividades, es necesario que el *tester* genere, a partir del activo DOCP, un registro de los fallos que se presentaron al ejecutar las pruebas funcionales descritas en los casos de prueba. En este sentido, se introduce un nuevo activo para gestionar el registro de los fallos identificados con los casos de prueba (activo REFAL), el cual se muestra en la Figura 3.11. El procedimiento correspondiente para el llenado de este activo es el mostrado en la Tabla 25.

Registro de fallos detectados en los casos de prueba: Activo REFAL				
Nombre del producto:		**Fecha:**		
ID del caso de prueba:		**Nombre del *tester* que ejecutó el caso de prueba:**		
Resumen de fallos detectados				
#	**Descripción del fallo**	**Tipo**	**Severidad**	**Requisitos funcionales involucrados**

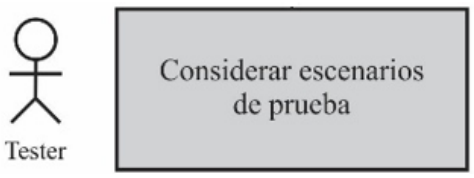
Figura 3.11. Formulario para el activo REFAL

Tabla 25. Instrucciones para el llenado del activo REFAL

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el *tester* está registrando los fallos detectados al ejecutar el caso de prueba.
ID del caso de prueba	Código o identificador que suele asignar el *tester* al caso de prueba que diseñó. En la mayoría de los casos se sugiere usar un identificador tipo CP-001, para denotar que se trata del caso de prueba 1 que se asocia con el producto.
Nombre del *tester* que ejecutó el caso de prueba	Nombre de la persona que funge como el *tester* que está ejecutando el caso de prueba.
#	Número secuencial asignado a cada fallo detectado con la ejecución del caso de prueba.
Descripción del fallo	Descripción textual del problema o fallo que está provocando un comportamiento no deseado o incorrecto en el software.
Tipo	Clasificación que se le da al fallo detectado con la ejecución del caso de prueba. En el contexto de esta estrategia, el *tester* deberá seleccionar un tipo de fallo funcional considerando la categorización presentada en la Tabla 24.
Severidad	Medida de evaluación subjetiva que el *tester* realiza sobre el fallo detectado. En el contexto de esta estrategia, el *tester* deberá considerar la probabilidad de ocurrencia y el impacto del fallo tomando en cuenta la matriz de severidad mostrada en la Figura 3.9. La severidad puede ser valorada por el *tester* como alta, seria, media o baja.
Requisitos involucrados	Requisitos funcionales, tomados de la matriz de trazabilidad (activo MATRAZ), que se relacionan con el fallo detectado. La identificación de estos requisitos es fundamental para el éxito de la prueba puesto que se controla el impacto de los cambios que se realicen al producto como resultado negativo en la ejecución de un caso de prueba.

Considerando lo anterior, el *tester* podrá identificar y seleccionar aquellos casos de prueba que deberá considerar inicialmente para ser ejecutados a causa de las modificaciones realizadas al código (i.e., regresión). Es importante hacer notar que, dado que se trata de un proceso cíclico, el conjunto de casos de prueba irá aumentado con cada regresión y que, incluso, es posible que se deban agregar nuevos requisitos al activo MATRAZ, lo que implicaría generar también nuevos casos de prueba.

3.5.2.4. La creación de escenarios de prueba para la regresión



Con los casos de prueba seleccionados, el *tester* debe definir ahora los escenarios de prueba funcional con la intención de establecer y controlar un entorno apropiado para su correcta ejecución. Una práctica sencilla, para el contexto de las pequeñas organizaciones, es partir de los requisitos funcionales que ya se tienen documentados en el activo MATRAZ y que se relacionan directamente con el caso de prueba que ha sido seleccionado. De esta forma, debe entenderse que un escenario de prueba representará a una simulación de una situación real que se utilizará para probar las funcionalidades del producto de software que se hayan modificado. Por lo tanto, resulta conveniente distinguir que los escenarios de prueba son diferentes de los casos de prueba ya que, como se explicó anteriormente, estos últimos son descripciones detalladas de las acciones, condiciones y entradas necesarias para probar aspectos particulares del producto de software. Un escenario de prueba, por otro lado, ayudará al *tester* a lograr un mejor entendimiento del contexto y familiarizarse con la prueba

dado que se asume como un usuario potencial. Así pues, bajo este razonamiento, el *tester* puede simular situaciones de la vida real y observar cómo interactúan los usuarios con el software mientras realizan acciones específicas para verificar que éste hace lo que debería y no lo que no debería. En el contexto de esta tesis, se sugiere que el *tester* de la pequeña organización simplifique esta labor a través de la creación de escenarios positivos y negativos. De acuerdo con Hooda y Chhillar (2015), un escenario positivo de prueba es aquel que verifica, mediante los pasos definidos, que el software acepta entradas válidas y realiza las acciones previstas; es decir, que cumple con los requisitos funcionales por los cuales fue creado. Por otro lado, un escenario negativo de prueba verifica que el software no realice acciones no deseadas al recibir entradas no validas, inesperadas o erróneas; es decir, que el software no se comporte de manera inesperada.

Tomando en cuenta esto, y con la finalidad de facilitar a la pequeña organización la creación de los escenarios de prueba, se plantea que el *tester* utilice el enfoque de Desarrollo Guiado en el Comportamiento (BDD, por sus siglas en inglés) para describir sus escenarios. En este sentido, la investigación de Farooq et al., (2023) establece que el BDD es un método ampliamente adoptado en el desarrollo ágil de software que enfatiza el comportamiento de una aplicación como una serie de escenarios de prueba, utilizando palabras clave como “Dado”, “Cuando” y “Entonces”. El BDD está basado en los enfoques utilizados por el TDD (Diepenbeck y Drechsler, 2015) pero opera a un nivel superior ya que integra al enfoque de las pruebas la identificación de los comportamientos esperados de un software. Además, se considera un enfoque valioso para la gestión de los requisitos y escenarios del cliente, que se expresan en forma de escenarios de prueba. Con esta idea en mente, los escenarios de prueba que se derivan de los requisitos desempeñan un papel fundamental en las interacciones verbales entre el equipo ágil y el usuario final. Por lo tanto, en el BDD se proporciona una serie de entradas para interactuar con la aplicación o el software, que se escriben como frases en lenguaje sencillo y que se organizan en un patrón definido «*Dado-Cuando-Entonces*» (Mishra, 2017; Smart y Molak, 2023), que describe los detalles como:

- *Dado*: muestra el contexto inicial.
- *Cuando*: presenta la ocurrencia de un evento.
- *Entonces*: demuestra la promesa de un resultado según lo esperado (i.e., salida esperada).

Así, los scripts definidos por el BDD utilizan un patrón que facilita la automatización de las pruebas. A continuación, se presenta un ejemplo con el fin de que se facilite la comprensión de este método:

- Escenario: Un asesor del banco finaliza la gestión de un préstamo bancario a un cliente que ya fue aprobado previamente.
- *Dado*: Que un cliente ha sido previamente aprobado para recibir un préstamo.
- *Cuando*: El asesor completa toda la información del crédito.

  | Nombre | Apellido | Número de INE
  | Julián | Velásquez | 10203936
- *Entonces*: El asesor debería visualizar el valor del préstamo.

De esta manera, el *tester* estaría creando escenarios para aquellos requisitos que están involucrados en los cambios realizados con la regresión y, por ende, se estaría preparando para generar casos de prueba que cubran las modificaciones llevadas a cabo por el programador. Con este proceso, se está asegurando de que, además de los casos de prueba que ya existen para el producto, el *tester*

genere nuevos casos de prueba que involucren a las funcionalidades modificadas y/o agregadas. La Figura 3.12 muestra el activo ESPRU, que se creó para ayudar al *tester* a realizar esta tarea.

Definición del escenario de prueba: Activo ESPRU			
Nombre del producto:		**Fecha:**	
ID del escenario:		**Nombre del *tester* que diseñó el escenario de prueba:**	
Requisito(s) funcional(es) involucrado(s) en el escenario:			
Actores involucrados en el escenario:			
Descripción del escenario positivo:		**Pasos**	
Dado:		1. 2. 3. 4.	
Cuando:			
Entonces:			
Descripción del escenario negativo:		**Pasos**	
Dado:		1. 2. 3. 4.	
Cuando:			
Entonces:			
ID del escenario:		**Nombre del *tester* que diseñó el escenario de prueba:**	
Requisito(s) funcional(es) involucrado(s) en el escenario:			
Actores involucrados en el escenario:			
Descripción del escenario positivo:		**Pasos**	
Dado:		1. 2. 3. 4.	
Cuando:			
Entonces:			
Descripción el escenario negativo:		**Pasos**	
Dado:		1. 2. 3. 4.	
Cuando:			
Entonces:			

Figura 3.12. Formulario para el activo ESPRU

La Figura 3.13 muestra las actividades que deben realizarse para que el *tester* genere escenarios (para las regresiones) que se complementen con los casos de prueba que fueron creados con el proyecto original. Como se puede observar en la figura, se parte de la identificación previa de los requisitos funcionales que están involucrados en la regresión. Posteriormente, se considera el identificar correctamente a los actores que se involucran con estas funcionalidades con el fin de describir un escenario positivo y otro negativo. Ambos escenarios deben representarse utilizando el patrón <<*Dado-Cuando-Entonces*>> definido por el BDD. El paso crucial en el proceso es describir el escenario en función de los pasos (i.e., acciones) que el *tester* deberá realizar para probar la funcionalidad asociada, puesto que, después de que cada escenario sea revisado y aceptado por el *jefe de proyectos*, se procederá a formalizarlo en un caso de prueba que se agregará al conjunto de casos

de prueba para una nueva versión del producto de software. Si el escenario no está descrito correctamente, éste deberá corregirse hasta que se cubra la funcionalidad que se pretende someter a prueba.

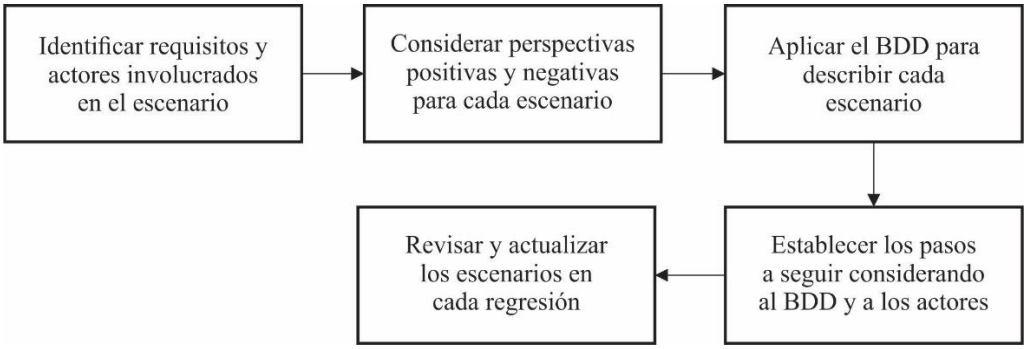


Figura 3.13. Actividades relacionadas con la fase "Considerar escenarios de prueba"

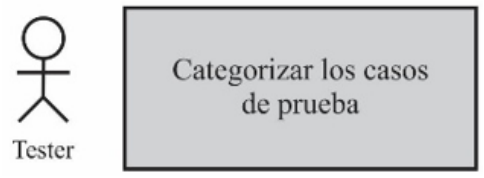
Debe entenderse que, en el caso del activo ESPRU mostrado en la Figura 3.12, solamente se contempla la descripción de dos escenarios a modo de ejemplo, pero que el *tester* podrá agregar tantas filas y pasos como sea necesario para cubrir cada funcionalidad afectada por los cambios realizados por el *programador*. Este activo se acompaña de la guía de llenado que se muestra en la Tabla 26.

Tabla 26. Instrucciones para el llenado del activo ESPRU

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el *tester* está creando los escenarios para describir las acciones que deberán probarse en el producto modificado.
ID del escenario	Código o identificador que suele asignar el *tester* al escenario. En la mayoría de los casos se sugiere usar un identificador tipo EP-001, para denotar que se trata del escenario de prueba 1 que se asocia con el producto.
Nombre del *tester* que diseñó el escenario de prueba	Nombre de la persona que funge como el *tester* que está diseñando el escenario de prueba.
Requisito(s) funcional(es) involucrado(s) en el escenario	Requisitos funcionales, tomados de la matriz de trazabilidad (activo MATRAZ), que se relacionan con el escenario. La identificación de estos requisitos es fundamental para el éxito de la prueba puesto que se controla el impacto de los cambios que se realicen al producto como resultado negativo en la ejecución de un caso de prueba.
Actores involucrados en el escenario	Usuarios que deberán interactuar directamente con el software para realizar una acción específica.
Descripción del escenario positivo	Descripción de una acción que se debe ejecutar en condiciones normales para verificar que el software funciona de acuerdo con lo esperado (e.g., comprar un artículo en un portal web y pagar con tarjeta de crédito).
Descripción del escenario negativo	Descripción de una acción que se realiza para verificar que un software no funciona con datos o comportamientos inesperados (e.g., el mismo portal web solicita que el usuario introduzca un PIN dentro de un determinado rango de caracteres para confirmar la compra, el escenario negativo implicaría introducir contraseñas que no cumplen con esos parámetros).
Dado	Contexto inicial del escenario. A menudo, algunos especialistas de la industria hacen referencia a las suposiciones que se asumen para que se puedan ejecutar los pasos que definen al escenario.
Cuando	Ocurrencia del evento o condiciones de los pasos a realizar.

Entonces	Promesa de un resultado, según lo esperado, de los pasos realizados.
Pasos	Acciones puntuales que describen las acciones que un supuesto usuario debería realizar para verificar tanto el escenario positivo como el negativo

3.5.2.5. La categorización de los casos de prueba que serán ejecutados



Los escenarios creados servirán ahora para que el *tester* genere casos de prueba que cubran las funcionalidades modificadas y/o agregadas por el *programador*. En este sentido, la categorización de los casos de prueba es fundamental en la estrategia propuesta en esta tesis, puesto que con ésta se pretende mejorar la eficiencia de una pequeña organización al realizar eficientemente las pruebas de regresión y, como consecuencia, mejorar también su cobertura. Por lo tanto, esta categorización representa, en el contexto de esta tesis, un enfoque sistemático que facilita a los *testers* de la pequeña organización la gestión y ejecución de las pruebas. Se pretende entonces definir un proceso de categorización que proporcione una guía para la organización de los casos de prueba en secciones que puedan ser gestionadas con facilitad, lo que hará más sencilla la tarea de que los *testers* localicen, ejecuten y gestionen las pruebas de regresión a lo largo del tiempo. Aunado a esto, se pretende que la pequeña organización sea capaz de:

- Promover la asignación eficiente de recursos y la priorización en la ejecución de los casos de prueba, de acuerdo con los criterios definidos por la categorización.
- Garantizar que se cubren todas las funcionalidades (i.e., requisitos funcionales) y aspectos del software (i.e., requisitos no funcionales). Esto le facilitará al *tester* la identificación de partes no probadas y garantizará un proceso de pruebas integral que abarca todas las características y escenarios de prueba definidos.
- Agrupar pruebas similares, que pueden ejecutarse por lotes, con el fin de facilitar la ejecución paralela de pruebas, reduciendo así el tiempo total asignado a la tarea.
- Generar fácilmente informes y análisis, puesto que los *testers* pueden obtener informes específicos basados en las categorías definidas y, de esta manera, realizar un análisis de los resultados de las pruebas, identificar patrones y facilitar la toma de decisiones informadas.

Los casos de prueba se pueden categorizar considerando diversos parámetros, cada uno con una finalidad distinta puesto que proporcionan información específica. En el contexto de esta tesis, la categorización se realiza considerando tres parámetros:

- Funcionales: Como su nombre lo indica, los casos de prueba funcionales se centran en las características y funcionalidades del código, módulo y/o componente modificado.
- No funcionales: Los casos de prueba no funcionales abarcan aspectos como el rendimiento, la seguridad y la usabilidad.
- Severidad: Los casos de prueba con mayor severidad identifican qué pruebas son críticas y deben ejecutarse primero. Esto garantizará que los fallos más importantes se detecten y solucionen cuando las regresiones comiencen a realizarse.

Por consiguiente, se deberá modificar el activo DOCP para agregar estos elementos de categorización o bien usar el nuevo activo llama DOCPR para denotar que son casos de prueba que

corresponden directamente con las regresiones realizadas. Considerando la futura automatización de estos activos se trata de una simple activación y desactivación de campos al requisitar la información; no obstante, el activo DOCPR se muestra en la Figura 3.14 para que se comprenda esta modificación. Las correspondientes instrucciones de llenado se proporcionan en la Tabla 27.

Descripción del caso de prueba para la regresión: Activo DOCPR									
Nombre del producto:						**Fecha:**			
ID del caso de prueba:				**Nombre del *tester* que diseñó el caso de prueba:**					
ID del escenario de prueba involucrado:				**Entorno de prueba:**					
Requisitos funcionales involucrados:									
Propósito del caso de prueba:									
Clasificación y prioridad del caso de prueba									
Tipo de caso de prueba	Funcional				No funcional				
Descripción de las acciones y/o condiciones de prueba									
#	**Acciones**				**Salida esperada**			**Salida real**	
Resultado									
Resultado de la prueba:	Aprobado		No aprobado				No aplica		
Seguimiento:	Sí		No		**Prioridad:**	Alta		Seria		Media		Baja	
Nombre y firma del *tester* que ejecutó el caso de prueba:						**Nombre y firma**			

Figura 3.14. Formulario para el activo DOCPR

Tabla 27. Instrucciones para el llenado del activo DOCPR

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el *tester* está documentando el caso de prueba.
ID del caso de prueba	Código o identificador que suele asignar el *tester* al caso de prueba que está diseñando. En la mayoría de los casos se sugiere usar un identificador tipo CP-001, para denotar que se trata del caso de prueba 1 que se asocia con el producto.
Nombre del *tester* que diseñó el caso de prueba	Nombre de la persona que funge como el *tester* que está diseñando el caso de prueba para su posterior ejecución.
ID del escenario de prueba involucrado	Código o identificador que suele asignar el *tester* al escenario. En la mayoría de los casos se sugiere usar un identificador tipo EP-001, para denotar que se trata del escenario de prueba 1 que se asocia con el producto.
Entorno de prueba	Configuración requerida para llevar a cabo la ejecución del caso de prueba que suele incluir hardware, software, configuraciones de red, datos requeridos para realizar la prueba. Puede indicarse como en desarrollo, pruebas, puesta en escena, o producción.
Requisitos funcionales involucrados	Requisitos funcionales, tomados de la matriz de trazabilidad (activo MATRAZ), que se relacionan con el caso de prueba. La identificación de estos requisitos es fundamental para el éxito de la prueba puesto que se controla el impacto de los cambios que se realicen al producto como resultado negativo en la ejecución de un caso de prueba.
Propósito del caso de prueba	Descripción de la funcionalidad que se pretende validar contra los requisitos funcionales involucrados en la definición del caso de prueba. Por ejemplo: “Se pretende probar que el inicio de sesión a la aplicación se realiza correctamente”.
Tipo de caso de prueba	Información que corresponde con aspectos funcionales o no funcionales que se abordan con el caso de prueba.
#	Número secuencial asignado a cada acción utilizada para describir las acciones que se realizarán para ejecutar el caso de prueba.
Acciones	Descripción puntual de los pasos que el *tester* deberá seguir para probar las funcionalidades del producto de software que están asociadas al caso de prueba.
Salida esperada	Resultado que se espera obtener después de ejecutar cada acción del caso de prueba.
Salida real	Resultado que se obtiene después de ejecutar cada acción del caso de prueba.
Resultado de la prueba	Información obtenida por el *tester* que le permite determinar si las funcionalidades probadas, a través de la ejecución del caso de prueba, se comportan correctamente. El *tester* debe considerar dos cosas: (1) que el comportamiento se alinee con lo establecido por los requisitos funcionales asociados al caso de prueba, y (2) la correspondencia entre la salida esperada y la real. Debe indicarse como aprobado o no aprobado.
Seguimiento	Dependiendo de la severidad de los fallos detectados como consecuencia de ejecutar el caso de prueba, el *tester* de indicar si se requiere de monitorizar el comportamiento del producto.
Prioridad	Medida de evaluación subjetiva que el *tester* realiza sobre el caso de prueba. En el contexto de esta estrategia, el *tester* deberá considerar la probabilidad de ocurrencia y el impacto de los fallos detectados con la ejecución del caso de prueba y tomando en cuenta la matriz de severidad mostrada en la Figura 3.9. En este caso, la prioridad dependerá de las decisiones que tome el *tester* (ver siguiente fase) para elegir entre alta, seria, media o baja.
Nombre y firma del *tester* que ejecutó el caso de prueba	Nombre y firma de la persona que realizó la ejecución del caso de prueba y reporta los hallazgos.

La Figura 3.15 resume las actividades para categorizar los casos de prueba a partir del uso de los activos ESPRU y DOCPR.

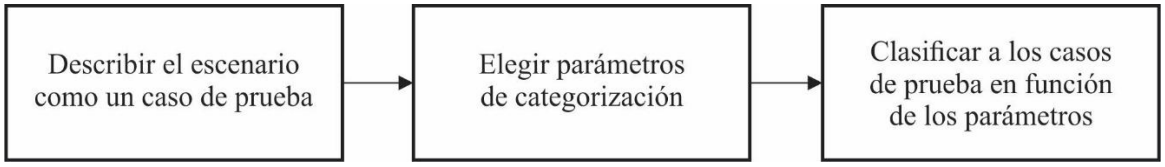
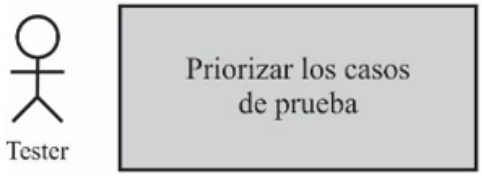


Figura 3.15. Actividades relacionadas con la fase "Categorizar los casos de prueba"

3.5.2.6. El establecimiento de prioridad en la ejecución de los casos de prueba



Independientemente del tamaño de la organización desarrolladora de software, la priorización se enfoca en la organización de los casos de prueba considerando su importancia y urgencia, de tal manera que se realicen primero aquellos que resulten ser los más importantes. De acuerdo con Catal (2012), este enfoque, aplicado a las pruebas de regresión, puede dar buenos resultados puesto que garantiza la evaluación exhaustiva de las funcionalidades críticas dentro del marco de tiempo disponible, especialmente cuando se realizan modificaciones continuas al producto de software.

En el contexto de esta tesis, se recomienda que el *tester*, una vez que haya categorizado y documentado los casos de prueba para la regresión a partir de los escenarios de prueba diseñados, los priorice considerando la severidad que cada uno tiene en el proyecto. Por lo tanto, la estrategia propuesta en esta tesis plantea que la pequeña organización se base principalmente en la priorización basada en la probabilidad de ocurrencia y el impacto, o bien lo que se definió como severidad en la Figura 3.9 para priorizar los requisitos que debían modificarse con la regresión. De esta manera, y considerando la misma matriz de severidad presentada en la figura, el *tester* puede determinar qué tan críticos y esenciales son los casos de prueba tanto para el negocio como para el producto de software. Para esto, hará uso del activo DOCPR definido en la fase anterior, prestando especial atención al campo de "prioridad" que es donde deberá realizar la valoración considerando la matriz de severidad mencionada con anterioridad.

La Figura 3.16 presenta las actividades que deben realizarse para que el *tester* considere, en primera instancia, aquellos casos de prueba que aportan mayor valor al negocio y, posteriormente, realice una segunda priorización considerando como más importantes a aquellas funcionalidades que afecten, en mayor medida, el comportamiento del producto de software.

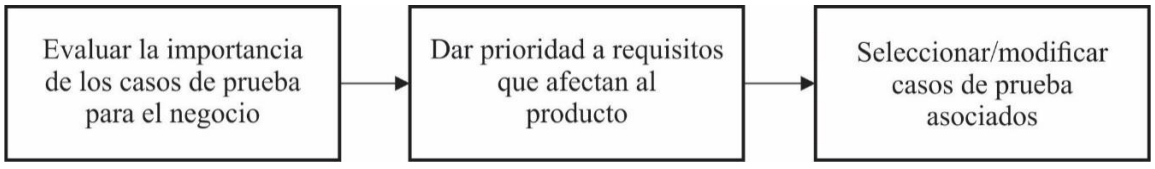


Figura 3.16. Actividades relacionadas con la fase "Priorizar los casos de prueba"

A modo de ejemplo ilustrativo, la Figura 3.17 muestra el análisis que un *tester* debería realizar para priorizar los casos de prueba que deben ejecutarse primero. Como se podrá observar, la correcta evaluación tanto de la probabilidad de ocurrencia como del impacto que tiene cada caso de prueba tanto para el negocio como para el producto en sí dependerá principalmente de la experiencia del

tester. Finalmente, una vez que los casos de prueba se hayan priorizado, dependerá del mismo *tester* reutilizar y/o modificar los casos de prueba que se encuentran en el repositorio, con el fin de lograr una mayor cobertura en las pruebas.

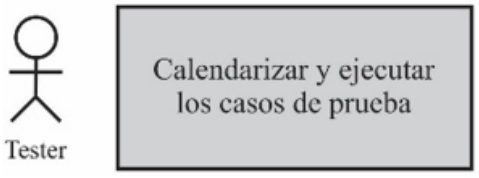
#	Descripción del propósito del caso de prueba	Probabilidad	Impacto	Prioridad
1	Verificar que el inicio de sesión de la app funcione correctamente	Remota (2)	Crítico (3)	Media (6)
2	Validar la coincidencia de la contraseña ingresada por el usuario con la asignada por el administrador	Improbable (1)	Insignificante (1)	Baja (1)
3	Evaluar cómo responde el software bajo una gran cantidad de usuarios simultáneos	Posible (4)	Crítico (3)	Alta (12)
4	Evaluar el rendimiento, la usabilidad y la seguridad de la app	Ocasional (3)	Marginal (2)	Media (6)
5	Probar que la app funciona correctamente en situaciones reales	Frecuente (5)	Catastrófico (4)	Alta (20)

#	Descripción del propósito del caso de prueba	Probabilidad	Impacto	Prioridad
1	Probar que la app funciona correctamente en situaciones reales	Frecuente (5)	Catastrófico (4)	Alta (20)
2	Evaluar cómo responde el software bajo una gran cantidad de usuarios simultáneos	Posible (4)	Crítico (3)	Alta (12)
3	Verificar que el inicio de sesión de la app funcione correctamente	Remota (2)	Crítico (3)	Media (6)
4	Evaluar el rendimiento, la usabilidad y la seguridad de la app	Ocasional (3)	Marginal (2)	Media (6)
5	Validar la coincidencia de la contraseña ingresada por el usuario con la asignada por el administrador	Improbable (1)	Insignificante (1)	Baja (1)

Figura 3.17. Ejemplo de análisis para priorizar los casos de prueba

Debe recordarse que los activos de la estrategia pueden ser adaptados a las necesidades de la pequeña organización y que, posiblemente, se desee agregar otros procedimientos y/o criterios para priorizar los casos de prueba considerando, por ejemplo, que se prueben primero aquellos que tienen mayor valor y beneficio para el producto o los usuarios. Independientemente del criterio que se desee utilizar, se considera que la matriz de severidad podría ser de utilidad para ayudar a los *testers* en la asignación y optimización de su tiempo y recursos para ejecutar las pruebas de regresión.

3.5.2.7. La programación de la ejecución de los casos de prueba seleccionados



La creación del plan de pruebas de regresión le facilitará al *tester* de la organización la definición de las pruebas que se realizarán para confirmar que las modificaciones realizadas al código del producto (a causa de cambio en las funcionalidades o actualización de las mismas) no han introducido errores que generen fallos en el producto. El objetivo de este plan es que

el *tester* o *jefe de proyectos* pueda vincular toda la información que ya ha sido obtenida en las fases anteriores con el fin de facilitarle la detección y tratamiento de los efectos negativos secundarios que pudieron haber ocasionado los cambios realizados al software. De esta manera, el plan de pruebas establecerá la pauta para la ejecución de los casos de prueba con el fin de encontrar la mayor cantidad de fallos en el software generados, en teoría, a partir de las modificaciones que se hayan realizado al código fuente. Así pues, la persona responsable de preparar este plan deberá contemplar que las pruebas pueden ejecutarse de forma manual o con herramientas automatizadas, ya que la estrategia propuesta no pretende modificar esta práctica puesto que el beneficio que otorga la automatización de las pruebas a una pequeña organización es importante. De hecho, la investigación realizada por Gamido y Gamido (2019) proporciona evidencia que permite asegurar que la automatización brinda a las organizaciones de software la facilidad de realizar pruebas más efectivas cuando se trata de tiempo, costo y usabilidad. Sin embargo, también es verdad que Wiklund et al., (2017) concluyeron que los costos de inversión para que una organización automatice la realización de pruebas pueden ser significativos. A pesar de que es imposible generalizar los costos de dicha automatización, estos investigadores identificaron que el tamaño y la complejidad de un sistema automatizado para pruebas pueden ser de la misma magnitud, o incluso mayores, que la complejidad y el tamaño del producto que se pretende probar. Este análisis les permitió conjeturar que el costo de adquisición de un sistema de ejecución automatizada de pruebas es similar al costo de desarrollo del producto de software que se está probando. También es importante considerar que si la automatización de las pruebas no funciona tan bien como se esperaba, es poco probable que la inversión realizada se recupere según lo previsto, lo que puede tener consecuencias significativas para la organización. Aunado a esto, es importante también saber que los sistemas de pruebas sufren problemas de deuda técnica similares a los de los sistemas convencionales. Por lo tanto, la decisión de realizar pruebas manuales o automatizadas dependerá completamente de la pequeña organización, dados los limitados recursos con los que a menudo cuenta.

En este sentido, una vez que los casos de prueba con las modificaciones más relevantes y recientes fueron seleccionados por el *tester*, es necesario calendarizar su ejecución. Por lo tanto, el activo PPRUR se creó para dividir este proceso en tres tareas básicas y orientar correctamente a la persona responsable de planificar la ejecución de los casos de prueba de la regresión (véase Figura 3.18). Estas tareas se describen de la siguiente manera:

- Preparación: Una vez que los casos de prueba fueron priorizados deben programarse para su ejecución, por lo tanto, se respeta el orden de importancia resultante y se toman tanto los propósitos como los IDs de la lista priorizada (activo DOCPR con la prioridad ya establecida). Es importante que se relacione a cada caso de prueba con los requisitos funcionales que se pudieron ver afectados con las modificaciones al código. Aunado a lo anterior, será necesario identificar el entorno de prueba requerido para ejecutar correctamente cada caso de prueba.
- Ejecución: Para optimizar el uso del tiempo y los recursos de la organización, es necesario que se identifique el origen de los datos de entrada que serán utilizados para verificar el comportamiento del software con relación a las funcionalidades agregadas y/o modificadas con los cambios al código. Se debe prestar especial atención en utilizar datos realistas. Además, será necesario que se especifique si la ejecución de un caso de prueba requiere de recursos especiales o no. Finalmente, se deberá programar una fecha de inicio y otra de fin para probar cada caso de prueba.
- Cierre: Por último, se registra el resultado de haber ejecutado el caso de prueba en el entorno definido, con los datos identificados, y dentro de las fechas establecidas.

Plan de pruebas de regresión: Activo PPRUR								
Nombre del producto:								
Descripción general de los cambios realizados:								
Preparación					**Ejecución**			**Cierre**
Propósito del caso de prueba	**ID**	**Requisitos funcionales**	**Entorno de prueba**	**Origen de los datos de entrada**	**Fecha de inicio**	**Recursos**	**Fecha de fin**	**Resultado de la prueba**

Figura 3.18. Formulario para el activo PPRUR

La guía de llenado correspondiente al activo PPRUR se muestra en la Tabla 28.

Tabla 28. Instrucciones para el llenado del activo PPRUR

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Descripción general de los cambios realizados	Resumen que el jefe de proyectos o *tester* responsable hace sobre los cambios realizados al producto durante el ciclo de regresión.
Propósito del caso de prueba	Descripción de la funcionalidad que se pretende validar contra los requisitos funcionales involucrados en la definición del caso de prueba. Por ejemplo: "Se pretende probar que el inicio de sesión a la aplicación se realiza correctamente". Se obtiene del activo DOCPR.
ID	Código o identificador que suele asignar el *tester* al caso de prueba que está diseñando. En la mayoría de los casos se sugiere usar un identificador tipo CP-001, para denotar que se trata del caso de prueba 1 que se asocia con el producto. Se obtiene del activo DOCPR.
Requisitos funcionales	Requisitos funcionales, tomados de la matriz de trazabilidad (activo MATRAZ), que se relacionan con el caso de prueba. La identificación de estos requisitos es fundamental para el éxito de la prueba puesto que se controla el impacto de los cambios que se realicen al producto como resultado negativo en la ejecución de un caso de prueba.
Entorno de prueba	Configuración requerida para llevar a cabo la ejecución del caso de prueba que suele incluir hardware, software, configuraciones de red, datos requeridos para realizar la prueba. Puede indicarse como en desarrollo, pruebas, puesta en escena, o producción. Se obtiene del activo DOCPR.
Origen de los datos de entrada	Identificación de la fuente a partir de la cual se generarán los datos de prueba, puede ser manual, generación automatizada, archivos externos, o interacción con otros módulos.
Fecha de inicio	La fecha en que el *tester* iniciará la ejecución del caso de prueba.
Recursos	Los recursos requeridos para ejecutar correctamente el caso de prueba, tales como humanos, equipos y herramientas automatizadas que se utilizan para evaluar y verificar que el software funciona correctamente después de la regresión.
Fecha de fin	La fecha en que el *tester* terminará la ejecución del caso de prueba.
Resultado de la prueba	Resultado obtenido después de haber ejecutado el caso de prueba, puede ser "Aprobado" o "No aprobado".

La Figura 3.19 resume las actividades que el jefe de proyectos o *tester* responsable deben llevar a cabo para calendarizar la ejecución de los casos de prueba en función de la prioridad que se le asignó previamente a cada uno y considerando, principalmente, las funcionalidades que fueron afectadas con las modificaciones realizadas al producto de software.

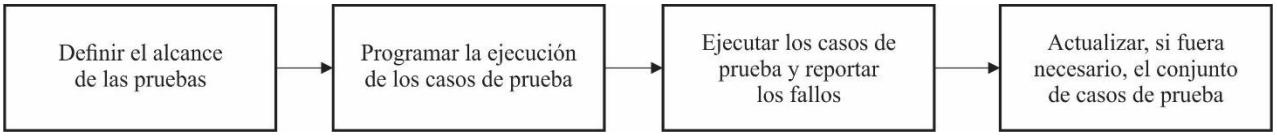


Figura 3.19. Actividades relacionadas con la fase "Priorizar los casos de prueba"

De manera adicional, el activo REFAL fue creado para generar un reporte de fallos para cada caso de prueba agregado al plan de pruebas (activo PPRUR). El activo REFAL es mostrado en la Figura 3.20 y su guía de llenado se encuentra en la Tabla 29.

Reporte de fallos: Activo REFAL			
Nombre del producto:		**Fecha:**	
ID del caso de prueba:		**Nombre del *tester* que diseñó el caso de prueba:**	

Fallos detectados		
#	**Descripción del Fallo**	**Tipo de fallo**

Nombre y firma del *tester* que ejecutó el caso de prueba:	**Nombre y firma**

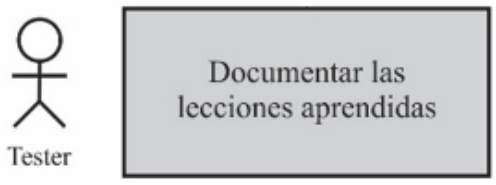
Figura 3.20. Formulario para el activo REFAL

Tabla 29. Instrucciones para el llenado del activo REFAL

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el *tester* está ejecutando el caso de prueba y documentando los fallos encontrados.
ID del caso de prueba	Código o identificador que suele asignar el *tester* al caso de prueba que está diseñando. En la mayoría de los casos se sugiere usar un identificador tipo CP-001, para denotar que se trata del caso de prueba 1 que se asocia con el producto. Se obtiene del activo PPRUR.
Nombre del *tester* que diseñó el caso de prueba	Nombre de la persona que funge como el *tester* que diseñó el caso de prueba que está bajo ejecución.
#	Número secuencial asignado a cada fallo encontrado al ejecutar el caso de prueba.
Descripción del fallo	Explicación detallada sobre el comportamiento incorrecto que el tester está encontrando al ejecutar el software. Mientras más detalle se proporcione en la descripción, más ayuda se le dará al *programador* para que corrija los errores.
Tipo de fallo	Información adicional que se le proporciona al *programador* y que se relaciona con la manifestación del fallo que origina que el software no funcione como se esperaba. Puede ser un tipo de fallo funcional (lógico, de recursos, algorítmico, de hardware) o no funcional (de interfaz gráfica, ortográfico, de sintaxis, tiempo de respuesta, de seguridad).
Nombre y firma del *tester* que ejecutó el caso de prueba	Nombre y firma de la persona que realizó la ejecución del caso de prueba y reporta los fallos encontrados.

Este proceso se repetirá hasta que el *programador* haya corregido todos los fallos detectados o bien, que el producto cumpla con los umbrales o políticas de liberación establecidas por la pequeña organización.

3.5.2.8. La generación de conocimiento a través de las lecciones aprendidas



Al cerrar un ciclo o terminar las modificaciones al código que amerite dejar de probar, el *tester* deberá recoger todo el conocimiento que se generó entre las personas que participaron en las regresiones, con el objetivo de generar un documento de lecciones aprendidas que sea de utilidad para que la pequeña organización mejore en futuras actualizaciones y/o mejoras de un producto de software. Por lo tanto, con esta fase se plantea que el *tester* y el equipo reconozcan acciones, decisiones o factores que hayan afectado positiva o negativamente su trabajo o hayan puesto en riesgo el éxito de la regresión en sí. En este sentido, las investigaciones de Wellman (2007) y Rhodes y Dawson (2013) destacaron el hecho de que toda organización, sea eficiente o no, tiene la obligación de aprender continuamente a realizar de manera más efectiva el trabajo considerando cuatro variables: la cultura de trabajo, las buenas prácticas, el manejo del conocimiento, y los procesos que le permitan obtener y reutilizar información en pro de ser más productiva y, en consecuencia, generar mejores resultados. Sin embargo, en el contexto de las pequeñas organizaciones desarrolladoras de software se opta más por la improvisación que por la documentación del conocimiento adquirido. Así, la estrategia presentada en esta tesis pretende inculcar en la cultura de trabajo de este tipo de organizaciones la identificación, documentación y gestión de incidencias con el fin de identificar medidas de acción que mitiguen o extrapolen sus

efectos. Considerando lo anterior, la Figura 3.21 resume las actividades que son requeridas para identificar, documentar y divulgar las lecciones que se aprenden a lo largo de cada ciclo de regresión.

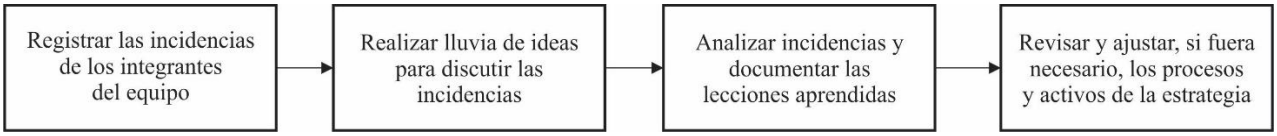


Figura 3.21. Actividades relacionadas con la fase "Documentar las lecciones aprendidas"

Como se observa en la figura, la estrategia establece que cada *tester* o participante del equipo en la regresión que se encuentre con alguna incidencia durante la realización de su trabajo, deberá registrarla de tal manera que se pueda establecer una estrategia efectiva de resolución, de tal manera que, si la misma incidencia volviera a aparecer en un futuro, se puedan mitigar sus efectos o mejorarlos en caso de que sea de naturaleza negativa o positiva, respectivamente. En este caso, lo ideal sería que este registro se realice en el momento en que la incidencia aparece con el fin de que se facilite su posterior clasificación y análisis. El activo REGI (véase Figura 3.22) facilita a cualquier integrante del equipo de trabajo la documentación tanto de incidencias negativas como positivas, con el fin de facilitar su gestión. La guía de llenado correspondiente con este activo se muestra en la Tabla 30.

Registro de incidencias: Activo REGI			
Nombre del producto:		**Fecha:**	
Nombre de la persona que identificó la incidencia:		**Rol:**	
Descripción de la incidencia ocurrida durante la regresión			
¿Qué ocurrió?			
¿Qué consecuencia tuvo en la regresión?			
Identificación del origen de la incidencia			
¿Qué la causó?			
Identificación de medidas de intervención para evitar que la incidencia vuelva a ocurrir			
¿Qué se propone para contrarrestar el impacto negativo o promover el impacto positivo de la incidencia?			

Figura 3.22. Formulario para el activo REGI

Tabla 30. Instrucciones para el llenado del activo REGI

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el integrante del equipo está documentando la incidencia.
Nombre de la persona que identificó la incidencia	Nombre de la persona que detecta una incidencia, positiva o negativa, que puede afectar la calidad de su trabajo.
Rol	Función que realiza la persona dentro de las regresiones realizadas al producto de software.
¿Qué ocurrió?	Descripción clara y concisa de la incidencia experimentada, mientras más detalles se den más conocimiento se puede generar con su tratamiento.
¿Qué la causó?	Descripción clara y concisa de las razones que llevaron a que la incidencia apareciera durante el ciclo y/o proyecto.
¿Qué se propone?	Descripción clara sobre las medidas de acción que se deberían tomar para resolver eficientemente incidencias negativas, si se volvieran a presentar en futuros proyectos, o para impulsar las incidencias positivas.

Se asume que todas las incidencias reportadas deberán ser tratadas por el *jefe de proyectos* o *tester* responsable de las regresiones con el objetivo de que organicen una lluvia de ideas con todos los participantes para aclarar la naturaleza de la incidencia y definir qué es y qué no una lección aprendida. Es recomendable que todos los participantes en las regresiones intervengan en esta sesión de trabajo, hayan o no reportado incidencias, puesto que se pretende hacer un ejercicio de reflexión y análisis que mejore paulatinamente la forma de realizar las pruebas de regresión. Una vez que las incidencias hayan sido tratadas y discutidas en la lluvia de ideas, el *tester* deberá documentar y clasificar las incidencias con el fin de tener un registro que incremente la base de conocimientos de la pequeña organización. En este sentido, el activo LECCA, mostrado en la Figura 3.23, se ha definido para que las lecciones aprendidas sean documentadas por el jefe de proyectos o *tester* responsable apropiadamente. La correspondiente guía para su llenado se visualiza en la Tabla 31.

Cabe mencionar que, con la intención de facilitar la selección de medidas correctivas, se plantea que las incidencias sean clasificadas de acuerdo con su tipo. Para el contexto particular de esta tesis se definieron tipos específicos de incidencias con el fin de facilitar la labor del *tester*, pero vale la pena recordad que una PAL permite incrementar la cantidad de conocimiento (i.e., activos) a medida que la organización va adquiriendo más experiencia, por lo que esta línea base (i.e., tipo de incidencias) puede adecuarse a las necesidades de las pequeñas organizaciones. Por lo tanto, los tipos definidos son los siguientes:

- Activo: Incidencia que surge a causa de haber utilizado incorrectamente alguno de los activos de la PAL-RT.
- Comunicación: Incidencia que se genera a causa de una mala comunicación entre los integrantes del equipo asignado a la regresión.
- Deficiencia: Incidencia que genera efectos negativos en la regresión y que debe resolverse a la brevedad.

Documentación de las lecciones aprendidas: Activo LECCA						
Nombre del producto:				**Fecha:**		
Nombre de la persona que documenta las lecciones aprendidas:					**Rol:**	
Descripción de las incidencias positivas y negativas durante la regresión						
Fecha	**Descripción de la incidencia**	**Tipo**	**Estado**	**Impacto**	**Dificultades futuras**	**Acciones tomadas**
Nombre y firma del *tester* que analizó las lecciones aprendidas:				**Nombre y firma**		

Figura 3.23. Formulario para el activo LECCA

Tabla 31. Instrucciones para el llenado del activo LECCA

Nombre del producto	Nombre que le fue asignado al producto una vez que fue liberado. En algunas ocasiones este nombre se asigna desde la concepción del proyecto y a menudo es propuesto por el cliente.
Fecha	La fecha en que el jefe de proyectos o *tester* responsable de la regresión está analizando y documentando las incidencias previamente reportadas.
Nombre de la persona que documenta las lecciones aprendidas	Nombre de la persona que analizó y está documentando las incidencias, tanto positivas como negativas, con el fin de generar conocimiento para que el personal de la pequeña organización logre un aprendizaje productivo.
Rol	Función que realiza la persona dentro de las regresiones realizadas al producto de software.
Fecha	La fecha en que el integrante del equipo documentó la incidencia.
Descripción de la incidencia	Descripción clara y concisa de las razones que llevaron a que la incidencia apareciera durante el ciclo y/o proyecto.
Tipo	Clasificación de la incidencia de acuerdo con el análisis que se esté realizando sobre los reportes del equipo de trabajo. Puede ser activo, comunicación, deficiencia, factor de éxito, gestión, liderazgo, persona, problema, reutilización, refactorización, técnica.
Estado	Evaluación que se hace sobre el estado de la incidencia, puede ser "Resuelto" si se trata de una incidencia que fue resuelta sin contratiempos o "Sin resolver" si no se ha podido corregir la mala práctica.
Impacto	Descripción concreta sobre las áreas, actividades y/o funcionalidades que se vieron o siguen afectadas por la incidencia.
Dificultades futuras	Descripción de las consecuencias que originó el no resolver las incidencias a la brevedad.
Acciones tomadas	Descripción de la forma en la que se eliminaron, mitigaron o intentaron eliminar los efectos adversos de la incidencia.
Nombre y firma del *tester* que analizó las lecciones aprendidas	Nombre y firma de la persona que analizó las incidencias reportadas por los integrantes del equipo.

- Factor de éxito: Alguna incidencia que no genera efectos negativos en la regresión sino que, por lo contrario, aporta valor agregado a uno a más aspectos asociados con el desarrollo del producto de software como, por ejemplo, la productividad, la calidad, el comportamiento del software.
- Gestión: Incidencia que ocurre debido a que se realizó incorrectamente la gestión de algún elemento, activo, persona, o caso de prueba.
- Liderazgo: Incidencia cuyo origen se da por la incapacidad del personal al que se le dio la responsabilidad de entregar y/o finalizar una actividad.
- Persona: Incidencia que se da a razón de que la persona responsable de alguna actividad la haya realizado deficientemente.
- Problema: Incidencia que, en caso de no controlarse, puede escalar a deficiencia y se dificulta su tratamiento.
- Reutilización: Incidencia que ocurre por no tener disponible información requerida para realizar eficientemente la regresión (e.g., documentación de casos de prueba, matriz de trazabilidad, plan de pruebas).

- Refactorización: Incidencia que se hace presente debido a múltiples errores al realizar la integración de componentes de código.
- Técnica: Incidencia relacionada con la mala aplicación de una técnica o más técnicas durante la regresión.

La estrategia establece que este proceso se repita hasta que las regresiones hayan sido terminadas y se cuente con un producto mejorado de software que pueda ser utilizado para cubrir nuevas necesidades de negocio.

Una vez que se finalizó el diseño de la estrategia, se procedió a evaluar su adherencia en las pequeñas organizaciones a través de una evaluación subjetiva de las fases, actividades, y activos contenidos en la PAL-RT. La siguiente sección de la tesis presenta los resultados obtenidos.

3.6. Evaluación de la adherencia de la estrategia en las pequeñas organizaciones

Aunque los métodos o técnicas que se encuentran en la literatura especializada para mejorar las pruebas de regresión han demostrado ser efectivos, su implementación y operación presentan barreras significativas para las pequeñas organizaciones. En este contexto, la estrategia potencialmente compatible con las necesidades y capacidades operativas de este tipo de organizaciones considera lo siguiente:

- Las recomendaciones de Minhas et al. (2023), quienes identificaron un conjunto de actividades esenciales para antes y después de la realización de las pruebas de regresión, al adecuarlas al contexto de las pequeñas organizaciones desarrolladoras de software.
- Se analizaron también los principios definidos por Jeon y von Mayrhauser (1994) y Marenbach y Albert (2018) para diseñar una estrategia basada en la gestión de conocimiento. Bajo esta premisa, no solamente se dotaría a la pequeña organización de una estrategia de gestión, sino que se inculcaría también en el personal la gestión de todo el conocimiento generado con la planeación, realización y gestión de las pruebas de regresión con el fin de mejorar la calidad de los productos generados.
- Por último, la estrategia diseñada es respaldada por una herramienta computacional que se diseñó en base a las recomendaciones vertidas en las investigaciones de Wnuk y Garrepalli (2018) y de Souza et al. (2021), de tal manera que su integración en las pequeñas organizaciones no sea invasiva y les permita madurar el proceso relacionado con la realización de las pruebas de regresión.

Sin embargo, una vez que se finalizó el diseño de la estrategia, y antes de crear una herramienta computacional que facilité su implementación en las pequeñas organizaciones, se procedió a evaluar su adherencia a través de una evaluación subjetiva de las fases, actividades, activos, medidas, y guías contenidos en la PAL. A continuación, se resumen los resultados obtenidos.

3.6.1. Participantes

La elección de los participantes se llevó a cabo a través de un muestreo intencional, enfocado a involucrar a profesionales con roles específicos dentro de sus organizaciones (i.e., jefes de proyectos, programadores y *testers*). En este sentido, se invitó a participar en la evaluación a las mismas 34 micro

y/o pequeñas organizaciones desarrolladoras de software que lo habían hecho en el estudio mostrado en la sección 2.5 de esta tesis, las cuales volvieron a responder positivamente. Por lo tanto, la muestra incluyó a los mismos 135 profesionales de organizaciones de los estados de Baja California Norte, Ciudad de México, Oaxaca, Puebla, Querétaro y Tlaxcala. Estos participantes, con una antigüedad promedio de cinco años, aportaron información valiosa para valorar la adherencia de la estrategia a sus respectivos entornos organizacionales.

3.6.2. Instrumentos

Los datos fueron recolectados de dos maneras: (1) a través de la aplicación remota de un cuestionario que fue diseñado para recoger las percepciones de los profesionales sobre la aplicabilidad de la estrategia en sus organizaciones (ver Tabla 32) y (2) mediante entrevistas semiestructuradas que se realizaron tanto de forma presencial como virtual, dependiendo de la disponibilidad de los participantes y las condiciones logísticas del estudio. Cada sesión de trabajo tuvo una duración aproximada de 45 a 60 minutos y fue grabada en audio, contando con la previa autorización de los entrevistados. El cuestionario se estructuró para explorar, a través de 15 ítems, aspectos como el ajuste de la estrategia a la cultura de trabajo de la organización, la capacidad del personal para realizar las actividades establecidas por la estrategia, la facilidad de uso de los activos, la claridad de las guías de adaptación y llenado, y la formalidad en la definición del proceso para realizar y gestionar las pruebas de regresión. Dicho instrumento se basó en una escala Likert de 5 puntos con las siguientes alternativas de respuesta: “Totalmente de acuerdo” (TA=2), “De acuerdo” (DA=1), “Neutral” (N=0), “En desacuerdo” (ED=-1) y “Totalmente en desacuerdo” (TD=-2), que se convirtieron a un valor numérico para obtener una medida cuantitativa. Además, se calcularon la mediana (M) y la desviación estándar (DE) de las respuestas proporcionadas por todos los participantes en la evaluación.

El cumplimiento de los principios éticos de la investigación fue asegurado mediante la protección de la confidencialidad y el anonimato de los participantes. Previo a su inclusión en el estudio, cada participante firmó un documento de consentimiento informado donde se especificaban los objetivos del estudio, el tratamiento que se daría a los datos y la libertad de retirarse del proceso cuando lo consideraran oportuno. Además, toda la información recopilada se almacenó en la nube con un acceso limitado.

3.6.3. Proceso y resultados

Como se indica, se diseñó un estudio bajo un enfoque cualitativo con la finalidad de analizar y entender a detalle las percepciones de los profesionales de la industria de software sobre la estrategia propuesta para definir un proceso que facilite la realización y gestión de las pruebas de regresión. En este sentido, se decidió seguir un enfoque cualitativo por su capacidad para percibir las opiniones, experiencias y dinámicas humanas en un entorno organizacional (Creswell, 2021), obteniendo así una percepción puntual sobre la manera en que la estrategia propuesta podría impactar en la mejora de la efectividad de las prácticas de desarrollo de software de las pequeñas organizaciones. De esta manera se pretendió determinar el grado de adherencia de la estrategia, presentada en la sección 3.5 de esta tesis, en las pequeñas organizaciones con el fin de hacer las adecuaciones necesarias y crear una herramienta computacional que implemente una versión refinada de la misma.

Por lo tanto, el estudio fue de tipo exploratorio y descriptivo, con el objetivo de examinar las opiniones de los profesionales sobre la estrategia propuesta. Este enfoque facilita la identificación de patrones y tendencias de uso en un contexto organizacional sin que el investigador se involucre directamente.

Tabla 32. Cuestionario diseñado para evaluar la adherencia de la estrategia propuesta en las pequeñas organizaciones desarrolladoras de software

#	Ítem	Respuestas					M	DE
		TA	DA	N	D	TD		
1	La documentación que revisé, relacionada con la estrategia para definir un proceso para realizar y gestionar las pruebas de regresión, es fácil de entender.	62	50	15	8	0	1.2	0.9
2	El lenguaje utilizado para describir todos los elementos de la estrategia es el adecuado.	42	93	0	0	0	1.3	0.5
3	Considero que esta estrategia se puede adaptar fácilmente a las necesidades de una pequeña organización desarrolladora de software.	42	45	12	21	15	0.6	1.4
4	El proceso definido para realizar y gestionar las pruebas de regresión podría adecuarse al contexto de mi organización.	97	28	10	0	0	1.6	0.6
5	La adopción de este nuevo proceso alteraría considerablemente nuestra cultura de trabajo con relación a las pruebas de regresión.	127	8	0	0	0	1.9	0.2
6	El éxito en la adopción del nuevo proceso dependerá, de manera importante, de la capacitación que se reciba.	38	97	0	0	0	1.3	0.4
7	Las fases y actividades del proceso descrito para realizar y gestionar las pruebas de regresión son descritas con el nivel de detalle necesario.	79	38	18	0	0	1.5	0.7
8	Los diagramas que acompañan la descripción detallada de las actividades son claros y representativos del trabajo que se requiere realizar.	90	42	3	0	0	1.6	0.5
9	Los activos de la PAL están definidos de tal manera que se facilita considerablemente su uso en mi organización.	4	116	11	4	0	0.9	0.5
10	La cantidad de activos creados para facilitar la implementación del proceso es suficiente para realizar y gestionar las pruebas de regresión.	134	1	0	0	0	2.0	0.1
11	Las guías de llenado que acompañan a los activos son claras y fáciles de seguir.	128	5	2	0	0	1.9	0.3
12	Considero que en mi organización se cuenta con las habilidades (suaves y duras) necesarias para implementar este tipo de procesos.	84	35	16	0	0	1.5	0.7
13	Entendí fácilmente las guías de adaptación para adecuar un proceso institucional a diferentes proyectos que requieran la realización de las pruebas de regresión.	0	5	46	63	21	-0.7	0.8
14	Estoy seguro de que con la orientación adecuada, yo y mis compañeros de equipo podríamos seguir puntualmente la estrategia sin problema alguno.	123	12	0	0	0	1.9	0.3
15	Estoy convencido de que una herramienta computacional simplificaría la implementación de la estrategia en mi organización.	89	46	0	0	0	1.7	0.5

En este sentido, se comenzó por compartir a través de un repositorio en la nube toda la información correspondiente a la estrategia para que fuera revisada por los participantes en la evaluación, incluyendo la descripción detallada del proceso mostrado en la sección 3.5.2 a nivel de fases y actividades, definición de medidas, guías de adaptación, políticas de uso, activos, y guías de llenado. La información estuvo disponible por 30 días para promover una revisión escrupulosa. Posteriormente, se aplicó un cuestionario en línea para recoger información sobre las percepciones de los participantes sobre la estructura, elementos y utilidad de la estrategia compartida. Por último, se llevaron a cabo entrevistas semiestructuradas con el objetivo de escudriñar en las percepciones y discrepancias de los participantes respecto a los elementos incorporados en la estrategia para mejorar el proceso de pruebas de regresión. El cumplimiento de los principios éticos de la investigación fue asegurado mediante la protección de la confidencialidad y el anonimato de los participantes. Previo a su inclusión en el estudio, cada participante firmó un documento de consentimiento informado donde se especificaban los objetivos del estudio, el tratamiento que se daría a los datos y la libertad de retirarse del proceso cuando lo consideraran oportuno. Además, toda la información recopilada se almacenó en la nube con un acceso limitado. La información recopilada fue transcrita y examinada mediante el análisis descriptivo, lo que ayudó a detectar opiniones recurrentes, tendencias y discrepancias relevantes en las respuestas proporcionadas por los encuestados. Para garantizar la validez del estudio se aplicó la triangulación de datos, contrastando las transcripciones y los resultados con los mismos participantes.

Analizando la información recogida con el cuestionario se observó que la mayoría de los participantes consideró que la documentación compartida sobre la estrategia es clara y, por ende, fácil de entender (M=1.2, DE=0.9). Sin embargo, el 17.0% no lo considera así. En la entrevista estos participantes argumentaron que (1) era difícil determinar un entendimiento adecuado al revisar únicamente descripciones, diagramas y plantillas y (2) que su opinión probablemente estuviera sesgada por la falta de una herramienta real. Vale la pena mencionar que se determinó que el 85.0% de estos participantes tenían la formación de ingenieros electrónicos y mecatrónicos, por lo que una falta de formación en el proceso de pruebas pudo haber sesgado también su opinión.

A pesar de la opinión anterior, el 100% de los participantes opinó que el lenguaje utilizado para desarrollar la estrategia era el adecuado (M=1.3, DE=0.5). La ausencia de ambigüedad en las respuestas demuestra que el lenguaje empleado en la estrategia no genera confusión y, de acuerdo con los encuestados, puede favorecer a una implementación fluida. Aunado a esto, el 36.0% de los encuestados consideró que la adaptación de la estrategia al contexto de las pequeñas organizaciones no sería fácil (M=0.6, DE=1.4).

Al indagar sobre esta percepción, los profesionales tuvieron diferentes puntos de vista: algunos consideraron que, si la estrategia se basaba únicamente en descripciones textuales y plantillas, representaría más trabajo burocrático; otros respondieron no estar seguros de que en sus organizaciones tendrían el tiempo para generar plantillas; y un grupo más pequeño se mostró inseguro de seguir correctamente lo indicado en la estrategia debido a su formación académica. Esta información será de mucha utilidad más adelante para seguir estrategias adecuadas que reduzcan el rechazo cultural de la mejora.

Al ser más específicos en la implementación de la estrategia en sus organizaciones, los participantes dieron opiniones totalmente diferentes, puesto que el 92.0% consideró que el proceso descrito por la estrategia podría adecuarse a su entorno de trabajo (M=1.6, DE=0.6). Al indagar en las entrevistas, el resto de los participantes argumentó, principalmente, que su opinión negativa se debía al desconocimiento de si en su organización habría disposición por cambiar la forma de trabajo.

Por otro lado, el 100% de los encuestados consideró que la implementación de un proceso diferente para realizar y gestionar las pruebas de regresión alteraría considerablemente la cultura de trabajo de su organización (M=1.9, DE=0.2). No obstante, al indagar en la entrevista, el 93.0% consideró que sería un cambio positivo.

Además, la mayoría de los participantes coincidió en que el éxito en la adopción del nuevo proceso de pruebas de regresión dependería significativamente de la capacitación que se reciba (M=1.3, DE=0.4). Esto refleja un reconocimiento colectivo de que, si bien, la estrategia descrita es clara y estructurada, su implementación requerirá el desarrollo de competencias prácticas y una comprensión a detalle de los flujos de trabajo para realizar la priorización de los cambios, la selección de los casos de prueba o la documentación de las lecciones aprendidas.

A pesar de que a la mayoría de los participantes le pareció clara y fácil de entender la documentación de la estrategia (diagramas, descripciones y plantillas), el 17.0% no consideró que las fases y actividades estuvieran descritas con el nivel de detalle necesario (M=1.5, DE=0.7), principalmente por dos razones: si bien el proceso y las descripciones son comprensibles, estos recursos por sí solos no les permiten adquirir un entendimiento adecuado de cómo se ejecutarían las pruebas en la práctica, y la falta de una validación práctica generó opiniones negativas.

Los participantes, en su mayoría, también consideraron que los diagramas asociados a la descripción de las actividades eran claros y representaban adecuadamente al flujo de trabajo establecido por la estrategia (M=1.6, DE=0.5). Sin embargo, tres de los profesionales que participaron el estudio no respondieron afirmativamente argumentando que las actividades descritas no se alienan con sus funciones y áreas de experticia. Como dato adicional, se trató de profesionales con formación de Electrónica y Mecatrónica que realizan las pruebas de regresión.

El 89.0% de los participantes consideró que los activos de la PAL fueron definidos de manera clara facilitando su uso en la organización (M=0.9, DE=0.5). Sin embargo, el resto de los participantes no compartió este punto de vista por dos razones principales: (1) la dificultad de asegurar un entendimiento de la estrategia basándose únicamente en material descriptivo, sin ejemplos prácticos, y (2) la falta de validación mediante una aplicación real que demuestre la utilidad de los activos. En la entrevista se corroboró que los perfiles de estos profesionales están más orientados a la realización de pruebas sobre sistemas de hardware y embebidos. No obstante, esta información resalta la importancia de complementar los activos con casos prácticos o capacitaciones específicas para asegurar una adopción uniforme en equipos multidisciplinarios.

De manera similar, el 100% de los encuestados coincidió en que la cantidad de activos era suficiente para lograr la implementación del proceso (M=2.0, DE=0.1), mientras que el 98% consideró que las guías de llenado que acompañan a estos activos eran claras y fáciles de seguir (M=1.9, DE=0.3), lo que demuestra que estos materiales complementarios logran su objetivo de orientar de manera efectiva a los profesionales en la implementación del proceso de pruebas de regresión.

Continuando con el análisis, el 88.0% de los participantes consideró que su organización posee las habilidades técnicas (duras) y personales (suaves) necesarias para implementar con éxito el proceso de pruebas de regresión descrito en la estrategia (M=1.5, DE=0.7). En contraste, el 12.0% que manifestó inseguridad en su respuesta argumentó durante la entrevista que requerirían mayor capacitación en aspectos específicos del flujo de trabajo. En conjunto, los resultados indican que la organización cuenta con una base sólida de competencias, aunque con margen para fortalecer ciertos aspectos que impulsen una implementación más robusta e inclusiva.

Los resultados revelaron un desafío importante para lograr la comprensión de las guías de adaptación para implementar el proceso de pruebas de regresión en diferentes proyectos. Mientras que solo el 4.0% de los participantes afirmó entenderlas fácilmente, el 62.0% expresó dificultades claras

y el 34% permaneció indeciso (M=-0.7, DE=0.8). Durante las entrevistas se confirmó que, aunque la estrategia puede ser clara, las guías específicas para adaptarla a contextos diversos no están logrando transmitir su propósito de manera efectiva. Los altos porcentajes de respuestas negativas y neutras revelan que las instrucciones podrían carecer de ejemplos específicos, facilidad para distintos tipos de proyectos o claridad. Por lo que será necesario que, una vez que se cuente con una herramienta computacional, se capacite a este tipo de profesionistas para que, a través de la práctica, comprendan la capacidad de adaptación de la estrategia.

Un número notable de participantes (91.0%) argumentó sentirse totalmente convencidos de que, con la orientación adecuada, podrían seguir de manera puntual la estrategia presentada en este estudio (M=1.9, DE=0.3). Este consenso muestra que los profesionales experimentaron confianza tanto del proceso descrito en la estrategia como de la capacidad del personal de su organización para implementarlo con el apoyo necesario.

Finalmente, la totalidad de los participantes coincidió en que una herramienta computacional facilitaría significativamente la implementación de la estrategia de pruebas de regresión en su organización (M=1.7, DE=0.5). En este sentido, los participantes consideraron que es necesario automatizar la estrategia propuesta con el fin de agilizar la adopción de tareas como la identificación de cambios en el código, la priorización de pruebas o la documentación de los resultados. Aunado a lo anterior, la mayoría de los profesionales argumentaron que una limitante en la mejora de los procesos es la definición de métodos manuales puesto que dificulta la adopción de nuevas prácticas. Por lo tanto, estas opiniones indican un sólido respaldo para considerar el desarrollo o la adquisición de herramientas computacionales como paso siguiente en la optimización del proceso de pruebas de regresión.

La información recogida con estos hallazgos cualitativos permitió la mejora de la estrategia a nivel de fases, actividades, activos y medidas y actualmente se está trabajando en el desarrollo de la herramienta computacional que será evaluada empíricamente en las pequeñas organizaciones que deseen seguir siendo parte del estudio.

Por último, vale la pena mencionar que este estudio presenta algunas limitaciones propias del enfoque cualitativo, como el número reducido de participantes y la subjetividad implícita de sus respuestas al cuestionario. En este sentido, si bien los resultados ofrecen las percepciones exactas de los profesionales de diversas pequeñas organizaciones desarrolladoras de software sobre la estrategia propuesta, las conclusiones no pueden extrapolarse a otras organizaciones o entornos pequeños con culturas diferentes de trabajo. No obstante, se intentó compensar el efecto de estas restricciones mediante el uso de la triangulación metodológica y la revisión minuciosa de las transcripciones y los análisis realizados.

La información recogida condujo a la obtención de las siguientes conclusiones:

- El grado de adherencia de la estrategia es apropiado para realizar una implementación práctica. Si bien existen algunos temores en los profesionales, debido principalmente por su formación e incertidumbre si una pequeña organización estaría dispuesta a cambiar, también existe disposición e interés por mejorar la forma de trabajo.
- Las percepciones de los profesionales han facilitado la identificación de las adecuaciones necesarias para diseñar una herramienta computacional que implemente una versión refinada de la estrategia.
- Existe un interés latente de las organizaciones participantes en el estudio para involucrarse en la evaluación empírica de la herramienta.

La información recogida con estos hallazgos cualitativos permitió la mejora de la estrategia a nivel de fases, actividades, activos y medidas con el fin de diseñar y desarrollar una herramienta computacional que facilitara su implementación en, al menos, una pequeña organización desarrolladora de software. Las generalidades de dicha herramienta son presentadas en la siguiente sección de la tesis.

3.7. Aspectos generales del diseño e implementación de la herramienta Kuali

Como parte de esta investigación, se consideró automatizar la implementación de la estrategia presentada en este capítulo de la tesis con el objetivo de facilitar su entendimiento y uso y, como consecuencia, mejorar la efectividad de las pruebas de regresión en organizaciones reales de desarrollo de software. Al tomar esta decisión se abordaron algunas de las inquietudes de las y los participantes, las cuales se hicieron presentes durante la evaluación de adherencia. Básicamente las y los participantes indicaron que la aplicación manual de dicha estrategia presentaría desafíos significativos, como la tendencia a errores humanos, la dificultad para gestionar la información y una curva de aprendizaje elevada.

Por lo tanto, para asegurar que la estrategia se aplicara de manera sistemática y eficiente, se optó por el desarrollo de una herramienta denominada Kuali. El objetivo principal de esta herramienta es servir como un vehículo de implementación que guíe al usuario a través de los pasos definidos por la estrategia y centralice los resultados, facilitando así su adopción y validando su efectividad práctica. Con este objetivo en mente, se adoptó la metodología ágil Scrum para gestionar el desarrollo de dicha herramienta. Esta elección se justifica por la naturaleza de la tesis, la cual requiere flexibilidad para adaptar los requisitos a medida que se valida el desarrollo de la herramienta. Así pues, Scrum permitió la construcción incremental e iterativa de la herramienta, asegurando así la entrega de valor continuo. Considerando este enfoque, el proceso de desarrollo se organizó en 24 *sprints*, que representaron ciclos de trabajo con una duración fija de una semana. De esta manera, se siguió el proceso representado en la Figura 3.24, cuyas fases se describen a continuación:

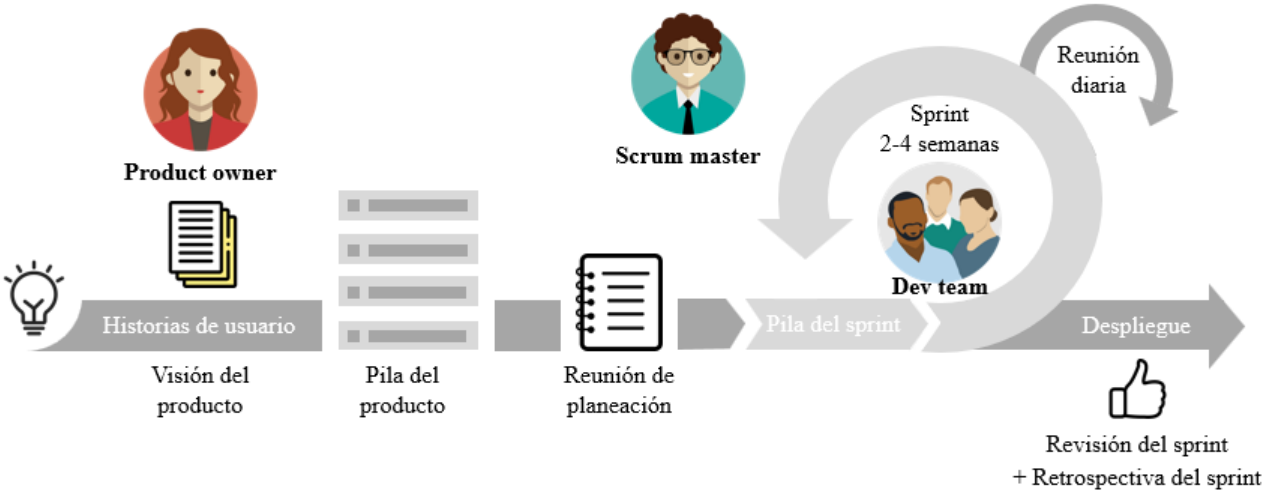


Figura 3.24. Proceso y fases de Scrum consideradas para el desarrollo de Kuali (creada a partir de Akhtar et al., (2022))

- Pila de producto: Inicialmente, se creó un listado priorizado de todas las funcionalidades (i.e., historias de usuario) que la herramienta debía tener. Estas historias se derivaron directamente de los pasos y requisitos de la estrategia diseñada en la tesis y se complementaron que los

deseos y necesidades que expresaron algunos de las y los participantes del estudio de evaluación de adherencia.

- Reunión de planeación: Al inicio de cada *sprint*, se seleccionó un conjunto de historias de la pila del producto para conformar la pila del *sprint* (i.e., el trabajo a realizar durante esa iteración). La prioridad se definió de acuerdo con la necesidad de validar las partes más críticas de la estrategia para gestionar las pruebas de regresión.
- El *sprint* (i.e., iteración de trabajo): Durante las 24 semanas del *sprint*, el esfuerzo se concentró exclusivamente en el desarrollo de las funcionalidades comprometidas (i.e., codificación de *backend*, *frontend* y pruebas).
- Reunión diaria: Aunque en este proyecto de tesis el equipo principal (*Dev team*) fue el propio tesista, siempre se mantuvo un seguimiento diario (similar a una reunión diaria) con el director de tesis (*Product owner*) para revisar el progreso, identificar impedimentos (e.g., un problema en la conexión a la base de datos, un error en *React*) y ajustar el plan del día.
- Revisión del *sprint*: Al finalizar el *sprint*, se generó el "incremento del producto" (i.e., las funcionalidades completadas y funcionales). Esta revisión permitió validar que la herramienta se alineaba con los objetivos de la estrategia y, en ocasiones, se mostraba al director de tesis para obtener retroalimentación temprana.
- Retrospectiva del *sprint*: Tras cada revisión se realizó un análisis crítico sobre el proceso de trabajo, la cual incluía qué salió bien, qué se podía mejorar y qué acciones se implementarían en el siguiente *sprint* para optimizar el flujo de desarrollo.

Este ciclo iterativo permitió construir una herramienta adaptada a los hallazgos de la investigación y centrada en el usuario final.

Por otro lado, para materializar la herramienta se seleccionó un conjunto de tecnologías modernas, eficientes y de código abierto, estructuradas en una arquitectura de tres capas (cliente, servidor y base de datos) que se describe de la siguiente manera:

- Base de Datos: PostgreSQL - Se eligió PostgreSQL en su versión v15.1 como el sistema gestor de base de datos relacional. Su robustez, capacidad para manejar consultas complejas y su estricto cumplimiento del estándar SQL, que se refleja en la integridad de los datos, fueron esenciales para almacenar de forma segura y fiable, por ejemplo, el inicio de sesión, los resultados históricos y los perfiles de usuario.
- *Backend*: FastAPI (Python) - El servidor o *backend* se desarrolló utilizando FastAPI, un framework web moderno de Python. La elección de FastAPI se basó en su rendimiento (gracias a su naturaleza asíncrona) y su capacidad para crear APIs RESTful de forma extremadamente rápida. Toda la lógica de negocio central, incluyendo las validaciones de datos, se implementó en el *backend*. FastAPI también genera automáticamente la documentación interactiva de la API (con *Swagger* UI), lo cual facilitó las pruebas y la conexión con el *frontend*.
- *Frontend*: *React* y Material-UI (MUI) - La interfaz de usuario (i.e., *frontend*), con la que las o los usuarios interactúan directamente, se construyó con *React*. Esta biblioteca de JavaScript permite crear interfaces de usuario dinámicas y reactivas basadas en componentes reutilizables. Para acelerar el desarrollo y garantizar una experiencia de usuario coherente, limpia e intuitiva, se utilizó la biblioteca de componentes Material-UI (MUI). MUI provee un conjunto completo de elementos (i.e., botones, formularios, tablas, *grids*) que implementan los

principios de *Material Design* de Google, permitiendo construir una interfaz profesional y responsiva (i.e., adaptable a móviles y escritorio) en menor tiempo.

Tomando en cuenta lo anterior, la Figura 3.25 representa la arquitectura cliente-servidor en que se basa Kuali, donde el navegador (*frontend*) y el servidor (*backend*) se comunican constantemente.

Figura 3.25. Ciclo de interacción de Kuali

A continuación, se proporciona una descripción del ciclo representado en la figura:

- Solicitud de usuario (zona naranja):
  - La o el usuario ingresan a Kuali a través del navegador de su preferencia (e.g., Chrome, Edge, Firefox) y realiza una acción.
  - En Kuali esta acción ocurre cuando el usuario hace clic en un botón como, por ejemplo, “Registrarse”, “Iniciar Sesión”, “Guardar”, “Buscar” o rellenar un formulario.
  - Ejemplo: Un usuario selecciona los activos para un proyecto y presiona “Guardar”.
- Solicitud API (zona amarilla):
  - La interfaz visual (*frontend*) no realiza las peticiones ni guarda los datos por sí misma; necesita pedirle ayuda al servidor (*backend*) para que empaquete la información del usuario y la envíe a través de Internet.
  - En Kuali, el navegador envía una petición HTTP (como un POST o GET) al servidor. Es una “llamada silenciosa” que el usuario no ve, pero que transporta los datos ingresados hacia el “cerebro” de la aplicación.
- PAL-RT (zona verde):

  - Aquí es donde reside el mayor valor de Kuali. El *backend* recibe la solicitud y decide qué hacer con ella basándose en las reglas específicas que se definieron como parte de la estrategia presentada a partir de la Sección 3.5 de este capítulo de la tesis.
  - En Kuali lo que ocurre es que el código en el servidor (Python) ejecuta los algoritmos, validaciones o modelos definidos previamente durante el desarrollo de la misma. A diferencia de una página web genérica, aquí se aplican las reglas exclusivas que hacen que Kuali cumpla con su propósito de gestión.
- Consulta de la base de datos (zona turquesa):
  - A menudo, la lógica necesita información previa o necesita guardar resultados nuevos. En otras palabras, el servidor "habla" con la base de datos.
  - En Kuali está acción se da cuando, por ejemplo, el *backend* ejecuta una consulta SQL (relacional) para: guardar el nuevo registro que creó el usuario o buscar información histórica necesaria para completar las peticiones del usuario.
- Procesamiento de datos (zona azul):
  - Los datos crudos que vienen de la base de datos o los resultados directos de la lógica no siempre están listos para enviarse al usuario, necesitan "limpiarse" o formatearse.
  - En Kuali, el *backend* toma la información recuperada, la organiza en un formato estándar (i.e., JSON), filtra datos sensibles y prepara el paquete final de respuesta a cada petición solicitada.
- Respuesta API (zona violeta):
  - Llegado este punto, el servidor ha terminado su trabajo y envía la respuesta de vuelta al *frontend* del usuario.
  - En Kuali esta acción se refleja cuando el *backend* envía un código de estado (e.g., como 200 – informando una respuesta positiva sin errores) junto con los datos procesados. En otras palabras, le informa al *frontend*: "Aquí tienes los resultados de la petición solicitada".
- Actualización de la interfaz (zona roja):
  - El ciclo se cierra. Es decir, el *frontend* recibe los datos y actualiza lo que el usuario ve en la pantalla, sin necesidad de recargar toda la página.
  - En Kuali la o el usuario visualizará el resultado de la petición a través de, por ejemplo, información sobre el proyecto o una notificación de "Guardado con éxito". El sistema queda listo para una nueva solicitud del usuario, reiniciando el ciclo.

Por lo tanto, el resultado de este proceso de desarrollo fue una herramienta web plenamente funcional que traduce la estrategia teórica desarrollada en esta tesis en una aplicación práctica y accesible (véase Figura 3.25). Como se mencionó anteriormente, dicha herramienta recibió el nombre de Kuali, que es una palabra en Náhuatl que significa "bueno" o "buena calidad", con lo que se pretende promulgar la idea que, a través de una gestión eficiente de las pruebas de regresión, se puede contribuir a mejorar la calidad del producto de software. Dicha herramienta no solo valida la viabilidad de la estrategia, sino que también proporciona un medio eficaz para su implementación, asegurando la consistencia y facilitando su uso por parte de los interesados.

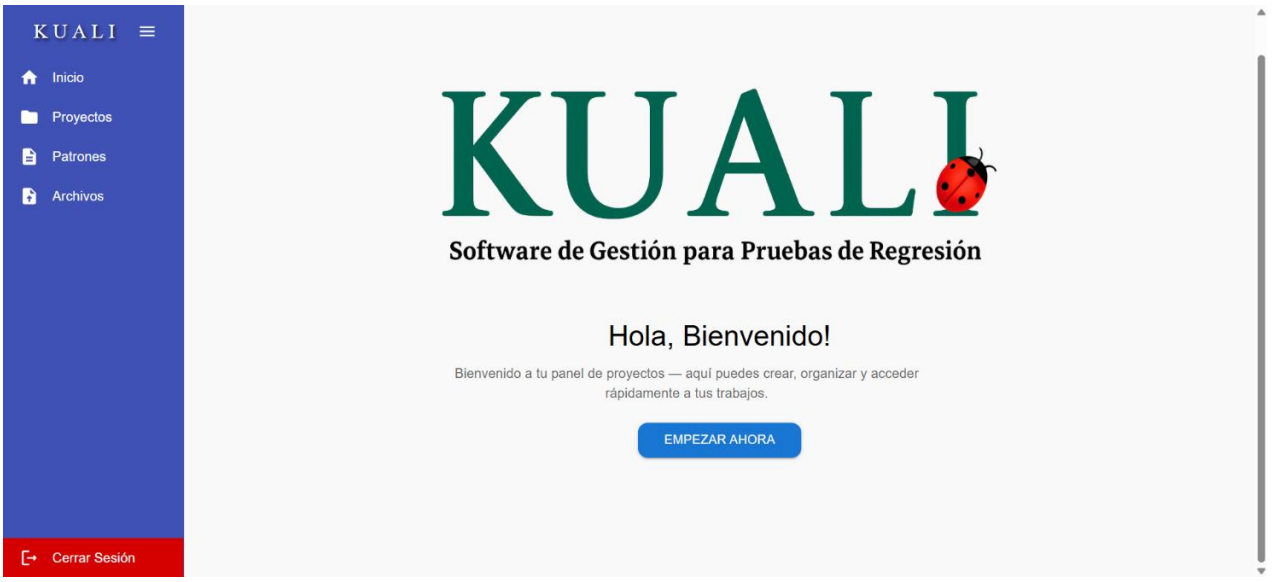


Figura 3.26. Pantalla de inicio de Kuali

Una vez que la herramienta fue terminada, se procedió a diseñar y realizar un caso de estudio con la intención de recoger información que permitiera validar la hipótesis establecida para esta tesis. En este sentido, en el siguiente capítulo se presenta toda la información relacionada con la evaluación empírica de Kuali.

4. Evaluación empírica

El Capítulo 4 de esta tesis expone los resultados obtenidos a través del diseño y conducción de un caso de estudio para evaluar empíricamente la implementación de la estrategia de gestión de las pruebas de regresión, presentada en el capítulo anterior. Por lo tanto, a continuación se presentan los hallazgos de usar dicha estrategia para definir e implementar un proceso de pruebas de regresión sobre proyectos piloto de tres pequeñas organizaciones desarrolladoras de software.

4.1. Contexto de la evaluación

Los resultados de la experimentación en las investigaciones relacionadas con la Ingeniería de Software que buscan la implementación de metodologías, modelos, técnicas y/o tecnologías aún son impredecibles. No obstante, Erthal et al. (2023) argumentó que desde hace poco más de tres décadas el proceso experimental se ha ido formalizando para aplicarse en la variedad de investigaciones que se realizan sobre las diversas áreas de la Ingeniería de Software, lo que ha conducido a la obtención de resultados prometedores. En este sentido, la experimentación ha probado ser efectiva para mejorar el entendimiento que se tiene sobre la Ingeniería de Software y, como consecuencia, mejorarla a través del conocimiento que se obtiene de los procesos y productos. De acuerdo con la investigación de Falessi et al. (2018), el proceso de experimentación en el contexto de la Ingeniería de Software se refiere a generar información sólida para demostrar, de manera irrefutable, las suposiciones e hipótesis que se establecen en las investigaciones que buscan mejorar la construcción y uso del software. En este sentido, se ha establecido que toda investigación que se realiza sobre la Ingeniería de Software se base en técnicas de experimentación que generen evidencia empírica para demostrar la utilidad y validez de las propuestas hechas. Así pues, la experimentación en la Ingeniería de Software proporciona, en palabras de Guevara-Vega et al., (2021), una mejor comprensión de lo que hace bueno al software, en términos de calidad, y cómo puede mejorarse su construcción. Es decir, se busca que el desarrollo de software sea una actividad científicamente predecible a través del análisis formal del conocimiento existente que se genera a través de las relaciones entre los procesos y los productos que se obtienen (Oliveira et al., 2021). Considerando lo anterior, la experimentación en la Ingeniería de Software se ha enfocado en darle formalidad a las investigaciones en el área a través de la introducción del paradigma experimental, lo que implica la definición y uso de estrategias empíricas de evaluación.

En este contexto, Wohlin et al., (2024) establecen que las estrategias empíricas para el desarrollo de experimentos en la Ingeniería de Software incluyen a los casos de estudio y experimentos. Un caso de estudio se centra en realizar un análisis profundo de una unidad específica (e.g., persona, organización, evento) para comprender un fenómeno más amplio, en lugar de buscar una generalización estadística. Por lo tanto, el investigador recopila información detallada sobre, por ejemplo, un único proyecto durante un período de tiempo. Aunado a esto, Baltes y Ralph (2022)

afirman que es posible aplicar una variedad de instrumentos para la recopilación de los datos durante la realización del caso de estudio. De hecho, Wohlin (2021) argumenta que si se desea comparar dos métodos, puede diseñarse un caso de estudio sobre un proyecto piloto para evaluar los efectos de un cambio con respecto a alguna línea base. Por lo tanto, los casos de estudio son adecuados para la evaluación de métodos y herramientas producto de investigaciones en el área de la Ingeniería de Software porque pueden evitar problemas de escalamiento. El estudio de Wohlin y Rainer (2022) argumentó que un caso de estudio es una investigación empírica que se hace sobre un caso (e.g., un grupo de personas, una organización, un fenómeno) a través del uso de múltiples métodos de recolección de datos, con el fin de estudiar un fenómeno contemporáneo en su contexto real, y sin que las o los investigadores participen activamente en el caso investigado. La diferencia entre los casos de estudio y los experimentos es que estos últimos muestrean las variables que están siendo manipuladas, mientras que los casos de estudio muestrean las variables que representan una situación típica. Sin embargo, una ventaja de los casos de estudio es que son más fáciles de planificar, pero la principal desventaja es que los resultados son difíciles de generalizar y más difíciles de interpretar; es decir, es posible mostrar los efectos en una situación típica, pero no puede generalizarse a cualquier situación (Martinsuo y Huemann, 2021). No obstante, en el contexto de esta tesis el caso de estudio pretendió precisamente demostrar que la solución propuesta generaba resultados positivos, o no, en situaciones y contextos específicos.

Considerando todo lo anterior, es evidente que se escogió un caso de estudio como la estrategia empírica experimental a seguir en la validación de la hipótesis establecida en el Capítulo 1 de esta tesis. Por lo tanto, de acuerdo con los argumentos de Wohlin et al. (2024), existen tres diferentes técnicas para desarrollar un caso de estudio:

1. Comparar los resultados obtenidos de una nueva propuesta contra una línea base.
2. Desarrollar dos proyectos en paralelo (llamados “proyectos hermanos”), eligiendo uno de ellos como base.
3. Aplicar la nueva propuesta sobre algunos componentes seleccionados y comparar los resultados que se obtengan con los de los componentes en los cuales no se haya aplicado.

Por lo tanto, como se indicó anteriormente en el Capítulo 1, se planteó el diseño y realización de un caso de estudio relacionado con el desarrollo de dos proyectos que se realizaron en paralelo dentro de tres pequeñas organizaciones desarrolladoras de software. En las siguientes secciones se presenta la información detallada de este proceso.

4.2. Definición del caso de estudio

El caso de estudio se enfocó en la implementación de la estrategia presentada en el Capítulo 3 de la tesis en tres pequeñas organizaciones desarrolladoras de software. Si bien se pudo haber implementado esta estrategia a través del seguimiento del proceso descrito en la sección 3.5.2, se consideró la información proporcionada por las empresas que participaron en la evaluación cualitativa de la estrategia propuesta, en el sentido en que utilizó la herramienta computacional para automatizar la monitorización y control de las pruebas de regresión a través de los activos creados. La invitación de participación se hizo a las 34 organizaciones que se involucraron en el estudio de adherencia, presentado en el capítulo anterior, y la selección de las tres participantes correspondió básicamente con tres criterios: (1) evidencia documental de que en realidad se llevara a cabo un proceso para gestionar las pruebas de regresión, (2) el interés de la organización por probar la estrategia en un proyecto piloto y (3) la disponibilidad de personal para integrar dos equipos de participantes de, al

menos, cuatro integrantes. Se propuso recoger tanto información cuantitativa como cualitativa, por lo que el caso de estudio siguió un enfoque mixto. Las características de las organizaciones que cumplieron con estos criterios, que en lo subsecuente serán llamadas O1, O2 y O3 por razones de confidencialidad, se presentan en la Tabla 33.

Tabla 33. Características de las organizaciones incluidas en el caso de estudio

Organización	Total de empleados	Ubicación	Actividad principal
O1	23	Oaxaca	Desarrollo de software a la medida y aplicaciones móviles
O2	19	Tlaxcala	Desarrollo de software contable (productos empaquetados)
O3	25	Puebla	Desarrollo de software a la medida (principalmente desarrollo web)

Básicamente las tres organizaciones comparten la misma estructura organizacional dado su tamaño, la cual está conformada básicamente por tres grandes áreas:

- Gerencia administrativa y técnica: Se encarga de dos funciones elementales: la gerencia administrativa es la cabeza de la organización y es de donde emanan las decisiones de negocio; mientras que la gerencia técnica es responsable de tomar las decisiones técnicas que se relacionan con cada producto de software que es desarrollado en la organización.
- Departamento contable. Se encarga de realizar todas las gestiones de índole contable con los clientes (e.g., apertura y cierre de contratos, control y rastreo de transferencias bancarias, pago de viáticos) y con los mismos trabajadores (e.g., pago de nómina, cursos, incentivos). Este departamento es controlado por uno o más contadores relacionados con la gestión administrativa.
- Área de desarrollo. Se encarga de desarrollar los proyectos software solicitados por los clientes. Dentro de este personal se encuentran los jefes de proyecto, analistas, programadores y *testers* que participan en el caso de estudio.

Con relación a la experiencia previa de estas organizaciones, se recogió la siguiente información:

- Las tres organizaciones se han enfocado entre tres y cinco años al desarrollo de software, con sus respectivas variantes, con el objetivo de proporcionar soluciones tecnológicas al mercado local. Argumentaron que regularmente los problemas en la gestión les impiden tomar una mayor cantidad de proyectos. No hicieron alusión a tener problemas con el proceso de pruebas, mucho menos con la gestión de las pruebas de regresión.
- O1 tiene un conocimiento limitado sobre algunas prácticas relacionadas con el proceso de pruebas pero argumenta que éste es llevado a cabo en la forma actual de trabajo, aunque se argumenta no tener conocimiento sobre si se realiza de forma institucionalizada. Aunado a esto, se argumenta usar un repositorio de Jira para gestionar los proyectos bajo un enfoque ágil y promover el seguimiento de incidencias, al mismo tiempo que se registran mediciones para gestionar su proceso. Además, se combina Jira con Appium para gestionar tanto las pruebas de aplicaciones móviles como de APIs. Por otro lado, O2 argumentó seguir el método Scrum y utilizar Trello para crear tableros Kanban. Con relación a la gestión de las pruebas de

software, la organización afirmó conducir un proceso a través de los *testers* que utilizan la herramienta de código abierto llamada TestLink. O3 también implementa el método Scrum y utiliza Jira y Kanban Tool para mejorar la gestión de los proyectos. Para gestionar el proceso de pruebas se utiliza la versión gratuita de Selenium, únicamente para desarrollos web, y Cucumber (de forma limitada).

4.2.1. Participantes

Como se mencionó, cada organización que participó en el estudio contribuyó con ocho personas que integraron dos equipos de trabajo conformados por un jefe de proyectos, un analista, un programador y un tester. Todas y todos los participantes tenían una experiencia mínima de ocho años trabajando en la industria de software. Estos equipos fueron escogidos por las organizaciones, sin intervención del tesista, director y codirectora de tesis, por lo que se desconocen las razones que condujeron a la integración de estos equipos. Un equipo de trabajo recibió la designación de "grupo de control" y el otro la de "grupo experimental". Siguiendo las recomendaciones de Wohlin et al. (2024), ambos grupos desarrollarían un mismo proyecto con la diferencia de que el grupo de control realizaría la gestión de las pruebas de regresión de forma tradicional (i.e., el proceso que habitualmente se sigue en la organización), mientras que el grupo experimental lo haría utilizando la estratega de gestión propuesta en esta tesis. En concreto, O1, O2 y O3 participaron cada una con dos equipo (i.e., un grupo de control y otro experimental) integrados por 8 participantes en total.

Además, la participación del tesista en la evaluación empírica realizada fue crucial desde el inicio, puesto que brindó capacitación sobre el uso de las funcionalidades de la herramienta computacional, el uso de activos y la gestión del proceso de pruebas. El soporte durante el desarrollo de los proyectos "hermanos" también fue parte de sus responsabilidades.

Finalmente, el director y la codirectora de la tesis apoyaron en el análisis e interpretación de la información recogida a través de la evaluación empírica.

4.2.2. Instrumentos

De acuerdo con lo establecido anteriormente, la investigación siguió un enfoque mixto, por lo que los datos fueron recolectados utilizando tanto herramientas cualitativas como cuantitativas. Considerando un enfoque cualitativo, se diseñó un cuestionario de evaluación siguiendo las recomendaciones de Engström y Runeson (2010) y Hynninen y Jantunen (2022) con el fin de determinar si, de acuerdo con las percepciones de las y los participantes del grupo experimental, la estrategia propuesta contribuía a mejorar el proceso relacionado con la gestión de las pruebas de regresión (véase Tabla 34). Dicho cuestionario busca evaluar la opinión de estas y estos participantes sobre la efectividad de las pruebas de regresión a través de preguntas que se dividen en cuatro categorías: la automatización de la estrategia, la cobertura de las pruebas, la detección de los fallos, y el tiempo de ejecución. Las preguntas se relacionan directamente con la estrategia propuesta y las métricas clave para asegurar que los cambios en el código no afecten negativamente la funcionalidad existente del software. Dicho instrumento se basó en una escala Likert de 5 puntos con las siguientes alternativas de respuesta: "Totalmente de acuerdo" (TDA=2), "De acuerdo" (DA=1), "Neutral" (N=0), "En desacuerdo" (ED=-1) y "Totalmente en desacuerdo" (TED=-2), que se convirtieron a un valor numérico para obtener una medida cuantitativa. Aunado a esto, se calcularon la mediana (M) y la desviación estándar (DE) de las respuestas proporcionadas por todas y todos los participantes en la evaluación.

Tabla 34. Cuestionario diseñado para recoger las percepciones de las y los participantes después de la evaluación empírica sobre la estrategia propuesta

#	Ítem	Respuestas					M	DE
		TDA	DA	N	ED	TED		
Metodología y estrategia	I1. La estrategia seguida es clara para definir y gestionar los casos de prueba de regresión.							
	I2. La estrategia facilita la gestión tanto de las pruebas manuales como las automatizadas.							
	I3. La estrategia se adecua a la frecuencia con la que se realizan las pruebas de regresión en mi organización.							
	I4. La estrategia me da el soporte adecuado para decidir qué casos de prueba debo re-ejecutar después de un cambio en el código.							
Cobertura	I5. Considero que la estrategia propicia una buena cobertura de código a través de la gestión de las pruebas de regresión.							
	I6. La estrategia facilita la cobertura de los requisitos funcionales a través de su trazabilidad.							
	I7. La estrategia contribuye a la evaluación de la cobertura que se tiene sobre los cambios a los requisitos y los casos de prueba.							
Detección y seguimiento de fallos	I8. Estoy conforme con la cantidad de fallos que fueron detectados en producción después de la ejecución de las pruebas de regresión.							
	I9. Considero que la tasa de detección de fallos durante las pruebas de regresión es apropiada.							
	I10. La estrategia contribuye a rastrear y priorizar los fallos detectados.							
	I11. La estrategia contribuye a identificar fallos críticos durante la ejecución de las pruebas de regresión.							
Automatización y mejora	I12. La estrategia puede alinearse con las herramientas usadas para automatizar las pruebas de regresión.							
	I13. La estrategia facilita la automatización de los casos de prueba de regresión diseñados.							
	I14. Las métricas definidas para medir la efectividad de las pruebas de regresión son apropiadas.							
	I15. La estrategia facilita la obtención de un análisis post-mortem de los ciclos de prueba de regresión para identificar oportunidades de mejora.							

Por otro lado, cuando se requirió, las discrepancias entre las respuestas de las y los participantes fueron abordas en entrevistas semiestructuradas que se realizaron de forma virtual, dada su ubicación y disponibilidad de horario. Cada entrevista tuvo una duración aproximada de 10 a 15

minutos y fue grabada en audio, contando con la previa autorización de las y los entrevistados. Cabe recalcar que, al igual que como se consideró en la evaluación de adherencia presentada en el capítulo anterior, se aseguró el cumplimiento de los principios éticos de la investigación mediante la protección de la confidencialidad y el anonimato de las organizaciones y las y los participantes. En este sentido, cada participante firmó un documento de consentimiento informado donde se establecieron los objetivos del estudio y el tratamiento que se daría a los datos.

Con relación al enfoque cuantitativo de la investigación, se decidió utilizar tanto la conducción de experimentos como el análisis de datos como principales técnicas cuantitativas para recoger datos que permitieran validar la hipótesis establecida en el capítulo inicial de esta tesis. La recogida de esta información, y parte del análisis posterior, se realizó a través de la herramienta computacional que automatiza la implementación de la estrategia propuesta para gestionar las pruebas de regresión. En este sentido, en la hipótesis se estableció que, a través de dicha implementación, podría mejorarse la "efectividad" de las pruebas de regresión. Por lo tanto, con el fin de establecer qué información objetiva debería recogerse durante la evaluación empírica, se consideraron las definiciones de Wong et al. (1997), Do et al., (2005) y Elsner et al. (2021), quienes argumentaron que la efectividad de las pruebas de regresión se puede definir como la capacidad de las pruebas para garantizar que los cambios o actualizaciones recientes no afecten negativamente las funcionalidades ya existentes y previamente operativas del software. En términos concretos, estos investigadores establecieron que se busca determinar si las pruebas de regresión tienen la capacidad de identificar fallos o efectos secundarios no deseados a causa de realizar modificaciones en el código, como alteraciones o adiciones de los requisitos. Así pues, considerando estas investigaciones, se decidió que esta efectividad se midiera mediante cuatro métricas clave:

- Cobertura de los casos de prueba: El porcentaje de funcionalidades existentes que están cubiertas por los casos de prueba de regresión ejecutados.
- Tasa de detección de fallos: La proporción de fallos de regresión encontrados por las pruebas con respecto al total de fallos de regresión que existen en el software.
- Tiempo de ejecución vs cobertura: La evaluación del tiempo invertido en las pruebas de regresión con respecto a su eficiencia para el nivel de cobertura y detección de errores que proporcionan.
- Número de inconsistencias en la trazabilidad: La cantidad de discrepancias que existen entre la matriz de trazabilidad considerando el número de requisitos agregados y modificados y su relación con las regresiones.

4.2.3. Diseño experimental y proceso

Es evidente que la evaluación empírica siguió un diseño cuasi-experimental, puesto que se requirió que las y los participantes tuvieran perfiles específicos, dejando a un lado la selección aleatoria de personas. El proceso seguido puede resumirse a través de los siguientes pasos:

1. Se inició con la capacitación a distancia de las y los participantes de los grupos experimentales con la finalidad de que se relacionaran con la estrategia de gestión propuesta en la tesis. Este proceso tuvo una duración de dos semanas, incluyendo la explicación de las funcionalidades de la herramienta computacional.
2. Posteriormente, se definieron las características de un proyecto con el fin de que fuera realizado por los grupos (i.e., control y experimental) integrados en las tres organizaciones. Esto requirió que se generaran credenciales para las y los participantes asignados a los grupos experimentales con el fin de que pudieran llevar a cabo la gestión de las pruebas de regresión.

3. Considerando las investigaciones de Madeyski y Kawalerowicz (2016) y Ghazi et al., (2018), se seleccionó un proyecto pequeño con una duración máxima de tres meses puesto que la experimentación con proyectos de pequeña escala en un entorno real de Ingeniería de Software incrementa la posibilidad de éxito al no agobiar al personal participante y, por ende, evita impactos negativos en los grupos de trabajo, pero, al mismo tiempo, facilita la recopilación de datos para realizar análisis cuantitativos suficientemente sólidos. En este sentido, se creó la descripción de un proyecto sencillo que fue entregado a los seis equipos para crear un software que abordara las funcionalidades descritas en la Tabla 35.

Tabla 35. Resumen de funcionalidades del proyecto desarrollado para la evaluación empírica

Grupos de funcionalidades		
Ventas, reembolsos y gestión de efectivo	**Gestión de datos maestros**	**Reportes y gráficas**
Permitir la búsqueda de productos	Gestionar productos, categorías y precios	Manejar múltiples parámetros de selección para cada reporte
Permitir el cálculo y la gestión del pago anual de impuestos	Calcular impuestos	Generar vista previa de reportes y gráficas
Integrar lector de código de barras para venta, cancelación y devolución de productos	Gestionar almacén (alta, baja, modificación de productos)	Exportar reportes a los formatos HTML, PDF, XML y CSV
Imprimir recibos	Gestionar usuarios y roles	Generar reportes de seguridad (accesos)
Generar resúmenes de informes sobre operaciones diarias, venta de productos e impuestos	Gestionar recursos	Generar reporte de estadísticas de productos más vendidos
Establecer seguridad basada en roles	Gestionar sucursales	Generar reportes de ventas diarias, semanales y mensuales
Realizar ventas a clientes	Gestionar promociones en productos	Generar reporte de pagos a proveedores
Realizar devoluciones a clientes	Integrar lector de código de barras para hacer alta, baja o modificación de productos	Generar reportes de ganancias y pérdidas
Realizar cancelaciones a clientes	Gestionar datos de productos, sucursales e inventarios	Generar reportes sobre stock de productos y niveles de inventario
Generar gráficos de ventas	Generar órdenes de proveedores	Generar reporte de productos con bajo stock (alerta de aviso)
Incorporar el corte de caja diario	Limitar accesos para acciones sensibles a operaciones de venta	
Presentar opciones de menú de acuerdo con el rol		

4. Una vez que las funcionalidades generales del proyecto piloto fueron definidas, se convocó a una reunión en línea con las y los participantes de cada organización para iniciar su desarrollo, estableciendo las siguientes condiciones: la duración del proyecto sería de, máximo, tres meses para los seis equipos; la cantidad de incrementos o iteraciones para desarrollar el proyecto en su totalidad quedaba a consideración de cada equipo; con la intención de propiciar las

regresiones, el tesista realizaría con frecuencia solicitudes de cambios para agregar o modificar funcionalidades; y, en dado caso de que el proyecto no pudiera concluirse, se cerraría la evaluación y se procedería a analizar la información generada de forma automática por la herramienta computacional. Es importante resaltar que, considerando este límite de tres meses de tiempo, cada equipo entregó un cronograma de trabajo que permitiera establecer una línea base de planificación inicial. Como se mencionó anteriormente, esto implica que se realizó la gestión de las pruebas de regresión en el proyecto ficticio considerando a los dos equipos de cada organización: el de control (que usó el método tradicional para realizar dicha gestión) y el experimental (que usó la herramienta computacional resultante de esta investigación).

5. Ya que los equipos terminaron el proyecto, el tesista procedió a analizar la información cuantitativa generada por la herramienta para confirmar si la estrategia propuesta mejoró o no la efectividad de los grupos experimentales en la gestión de las pruebas de regresión. En este sentido, se procedió a analizar con cuidado el tiempo que cada equipo tardó en terminar el proyecto; el número de regresiones consideradas durante todo el proyecto; el número de fallos detectados en las regresiones después de las modificaciones requeridas por el tesista; la tasa de fallos detectados en la regresión vs el número de fallos al final de cada incremento; el número de inconsistencias en la matriz de trazabilidad al final de cada incremento (relacionadas con las modificaciones requeridas para los requisitos funcionales y las regresiones registradas en la herramienta de gestión); el número de modificaciones realizadas por cada equipo a su forma de trabajo; la cantidad de activos creados o modificados para ajustar la estrategia; y la cantidad de retrasos (en días) que afectaron al proyecto considerando la planificación inicial que cada equipo propuso al iniciar el proyecto. A través de este análisis numérico se generaron resultados directamente comparables entre los proyectos hermanos para determinar la efectividad de la estrategia propuesta.
6. Finalmente, se convocó a los participantes a una reunión de cierre y se aplicó un cuestionario de preguntas cerradas para determinar las percepciones de los equipos involucrados en los grupos experimentales sobre la implementación de la estrategia de gestión. A diferencia de la evaluación de adherencia presentada en el capítulo anterior, esta vez se deseaba conocer las opiniones de las y los participantes después de haber puesto en práctica la estrategia a través de la herramienta computacional. Es decir, a pesar de que es evidente que no se generarían datos directamente comparables con ambas evaluaciones, resultaba interesante determinar si la opinión de estas personas cambiaba respecto a la inicial.

4.3. Resultados cuantitativos

Considerando el proceso anterior, los equipos experimentales accedieron a Kuali para crear el proyecto piloto que se desarrollaría tomando en cuenta el conjunto de funcionalidades resumidas en la Tabla 35 (véase Figura 4.1). Como se indicó anteriormente, por cuestiones de confidencialidad los datos personales de las y los participantes, así como de las organizaciones, han sido eliminados de las imágenes mostradas en este capítulo con la intención de proteger su privacidad.

Por otro lado, debido a la disponibilidad limitada de las y los participantes, se acordó que los proyectos iniciaran entre los meses de julio y agosto del 2025. Sin embargo, no hubo manera de que los grupos de control de O2 y O3 pudieran iniciar el proyecto al mismo tiempo que sus compañeras y compañeros de los grupos experimentales. Por lo tanto, se decidió que cada organización llevara a cabo el proyecto bajo la metodología que deseara y que, para facilitar el análisis de la información recogida durante y después del experimento, mencionaran en la reunión de lanzamiento cómo se planeaba trabajar. En este sentido, es importante aclarar que tanto el tesista como el director y la

codirectora no intervinieron en la elección de la metodología ni asesoraron a los equipos durante el tiempo que les llevó entregar el producto terminado. Las únicas funciones que desempeñó el tesista fue brindar un soporte de tiempo completo para resolver dudas sobre alguna cuestión especial de la herramienta, aclarar la modificación y/o adición de una funcionalidad, y analizar continuamente el desempeño de las y los participantes a través de Kuali (i.e., como administrador) y a lo largo del proyecto.

Figura 4.1. Proyectos registrados en Kuali por las y los jefes de proyectos de las organizaciones participantes (vista de administrador)

En contraste, con la intención de no sesgar la información recogida de los equipos de control, se pidió a las y los participantes que, una vez que el proyecto estuviera terminado, se realizara una reunión para compartir información puntual de sus actividades para gestionar las pruebas de regresión durante el experimento. De hecho, el tesista pudo analizar esta información en compañía de los grupos de control de las tres organizaciones. En las siguientes secciones se presenta la información recogida y analizada.

4.3.1. Cobertura de los casos de prueba

La cobertura de los casos de prueba en las pruebas de regresión se define como el grado en que el conjunto de pruebas de regresión existente verifica las funcionalidades, requisitos y especificaciones del sistema para asegurar que los cambios recientes en el código no hayan introducido nuevos defectos ni afectado negativamente las funcionalidades existentes. En el contexto de la experimentación, la determinación de dicha cobertura se basó en dos métricas fundamentales: la propia Cobertura de los Casos de Prueba (CCP), que permite medir la eficiencia del proceso de pruebas de regresión, y la Cobertura de los Requisitos Funcionales (CRF), que determina la cantidad de funcionalidades que están cubiertas por los casos de prueba de regresión. Rosenblum y Weyuker (1997) y Butool et al., (2019) definieron dichas métricas en términos de las siguientes fórmulas:

$$CCP = \left(\frac{Número\ total\ de\ casos\ de\ prueba\ de\ regresión}{Número\ total\ de\ casos\ de\ prueba}\right) \times 100 \qquad (11)$$

$$CRF = \left(\frac{Número\ de\ requisitos\ cubiertos\ por\ los\ casos\ de\ prueba\ de\ regresión}{Número\ total\ de\ requisitos}\right) \times 100 \qquad (12)$$

De acuerdo con estas investigaciones, un porcentaje de cobertura del 80% en ambas métricas se considera generalmente un buen objetivo para las pruebas de regresión, aunque este valor puede variar según el proyecto y los requisitos específicos. En este sentido, la Tabla 36 resume tanto la información recogida de los grupos experimentales a través de Kuali como la compartida por los grupos de control, la cual también fue contrastada a través de los documentos generados y las herramientas utilizadas.

Tabla 36. Información recogida para evaluar la cobertura de los casos de prueba

	Organizaciones participantes					
	O1		**O2**		**O3**	
	Grupo de control	**Grupo experimental**	**Grupo de control**	**Grupo experimental**	**Grupo de control**	**Grupo experimental**
# total de requisitos funcionales	79	83	53	69	63	71
# total de casos de prueba	168	206	81	132	130	164
# de requisitos cubiertos por los casos de prueba de regresión	26	71	16	36	39	59
# total de casos de prueba de regresión	35	160	12	45	56	139

Una de las principales cuestiones resaltables de esta tabla es que se percibió una mejora entre los datos reportados por los grupos de control (celdas resaltadas en color amarillo en la Tabla 36) y los generados por los grupos experimentales (celdas resaltadas en color verde en la Tabla 36). En este sentido, la mejora más significativa se dio en O3 y la menos significativa en O2. Ahora bien, con esta información se procedió a calcular las métricas considerando las fórmulas (1) y (2). Los resultados se muestran en la Tabla 37.

Tabla 37. Valores de las métricas de cobertura de los casos de prueba y cobertura de los requisitos funcionales

Métrica	**Organizaciones participantes**					
	O1		**O2**		**O3**	
	Grupo de control	**Grupo experimental**	**Grupo de control**	**Grupo experimental**	**Grupo de control**	**Grupo experimental**
CCP	21%	78%	15%	34%	62%	85%
CRF	33%	85%	30%	52%	43%	83%

Si bien, como se mencionó anteriormente, se logró determinar una mejora a favor del grupo experimental en las tres organizaciones respecto a la medición de la cobertura de los casos de prueba, el grupo experimental de O3 es el único que logro un porcentaje de cobertura mayor al 80% (celdas resaltadas en color verde en la Tabla 37), que se indica en la literatura como un valor ideal. De hecho, no se pensaba, al momento de diseñar el caso de estudio, que se obtendrían resultados similares en las organizaciones participantes. Cuando se indagó al respecto por medio de las entrevistas, se descubrió que el grupo experimental de O3 hizo un rastreo impecable de los requisitos funcionales a través del activo MATRAZ, lo que le permitió identificar tempranamente qué requisitos estaban involucrados en cada caso de prueba de regresión y, como consecuencia, se generaron casos de prueba de regresión que involucraban a funcionalidades que habían sido probadas en incrementos anteriores pero tenían que volverse a probar a causa de las modificaciones o adiciones de funcionalidades (véase Figura 4.2).

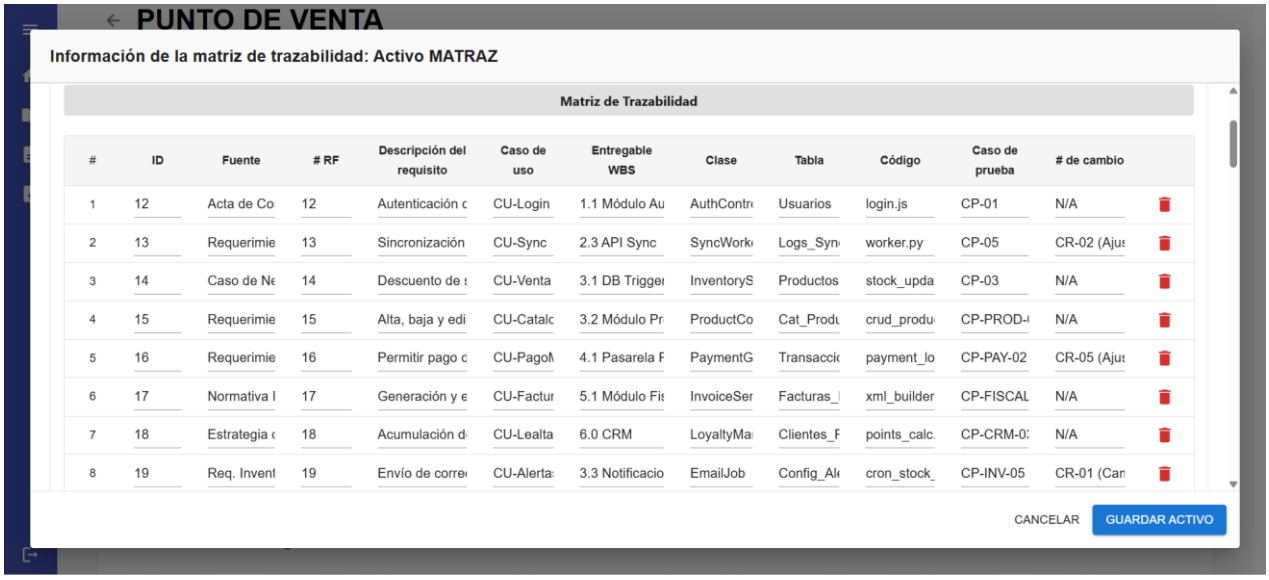


Figura 4.2. Matriz de trazabilidad creada por el grupo experimental de O3 (vista de jefe de proyectos)

Aunado a esto, la evidencia analizada en Kuali para el grupo experimental de O1 mostró un rastreo de la traza de los requisitos funcionales bastante bueno, lo que, de manera similar, condujo a un diseño apropiado de los casos de prueba a través del activo DOCP (véase Figura 4.3).

En contraste, el grupo de control de O2 fue el que tuvo la cobertura más baja en el experimento (celdas resaltadas en color rojo en la Tabla 37). Al entrevistar a los participantes se argumentó que en la organización si bien se suele generar una matriz de trazabilidad con Trello, ésta no se gestiona eficientemente para relacionar los casos de prueba de regresión con las funcionalidades identificadas. Por otro lado, como se indicó antes, la organización utiliza TestLink para gestionar la realización de las pruebas de software y, de acuerdo con la literatura especializada, ambas no son decisiones acertadas dadas dos situaciones importantes: (1) si bien es cierto que Trello puede utilizarse para gestionar la trazabilidad de los requisitos mediante la configuración de tableros que visualizan el flujo de trabajo de los requisitos, desde su ideación hasta su implementación, no es una herramienta nativa de gestión de requisitos, y (2) es un hecho que TestLink es una herramienta popular y confiable para la gestión manual de los casos de prueba; sin embargo, su interfaz de usuario está desactualizada, las funcionalidades de informes son básicas y el soporte nativo para la integración con herramientas de automatización modernas es limitado.

Por lo tanto, se concluyó que la evidencia indica que, para lograr una mejor cobertura de los casos de prueba, se requieren tres factores importantes: un alto compromiso por parte de los equipos

de desarrollo para realizar puntualmente las actividades y utilizar los activos que se recomiendan en la estrategia; las y los integrantes de los equipos deben poseer conocimientos suficientes como para realizar correctamente la gestión de un proceso; y es fundamental utilizar el soporte tecnológico apropiado.

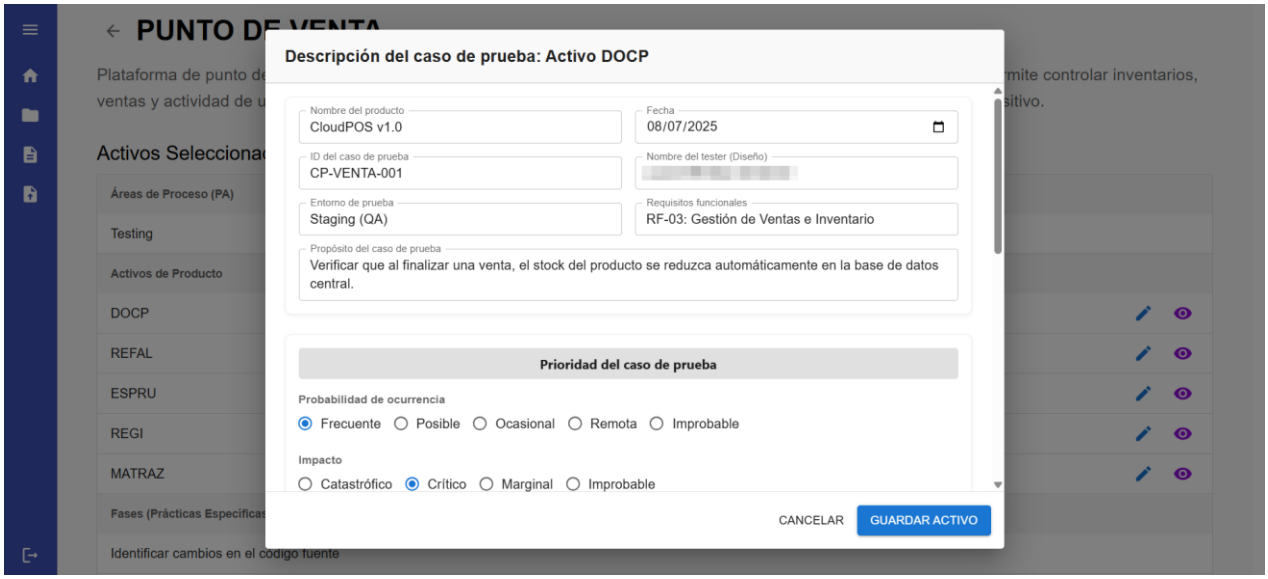


Figura 4.3. Ejemplo de caso de prueba de regresión creado por el grupo experimental de O1 (vista de tester)

4.3.2. Tasa de detección de fallos

De acuerdo con Nayar et al. (2022), la tasa de detección de fallos en las pruebas de regresión es una métrica que mide la efectividad de estas pruebas al comparar tanto el número de fallos descubiertos con las pruebas de regresión como los casos de prueba que fallaron. Esta métrica se puede determinar comúnmente a través de la Tasa de Fallos en los Casos de Prueba (TFCP) y del Porcentaje de Detección de Fallos (PDF). Estos valores se pueden cuantificar a través de las siguientes fórmulas:

$$TFCP = \left(\frac{Número\ de\ casos\ de\ prueba\ que\ fallaron\ por\ las\ regresiones}{Número\ total\ de\ casos\ de\ prueba}\right) \times 100 \quad (13)$$

$$PDF = \left(\frac{Número\ de\ fallos\ encontrados\ durante\ las\ pruebas\ de\ regresión}{Número\ total\ de\ fallos}\right) \times 100 \quad (14)$$

La investigación de Rosero et al. (2016) argumenta que una alta TFCP indica que las pruebas de regresión son eficientes para identificar problemas antes del lanzamiento. Por otro lado, el PDF puede variar ampliamente en las pruebas de regresión y no existe un número estándar universal para la industria, ya que depende en gran medida de la complejidad del software, la estrategia de pruebas y la madurez del proceso de desarrollo. Sin embargo, se argumenta que los procesos de prueba efectivos generalmente buscan lograr una eficiencia de aproximadamente el 90% o superior, lo que implica una fuga de defectos (i.e., defectos que pueden llegar a producción como fallos) de solo el 10 al 12%.

La Tabla 38 resume la información recogida en las tres organizaciones. De igual manera que con la métrica anterior, la información de los grupos experimentales se obtuvo directamente de Kuali; mientras que la información de los grupos de control se analizó con la documentación compartida y los informes generados por algunas de las herramientas utilizadas regularmente en las organizaciones.

Tabla 38. Información recogida para evaluar la tasa de detección de fallos

	Organizaciones participantes					
	O1		**O2**		**O3**	
	Grupo de control	**Grupo experimental**	**Grupo de control**	**Grupo experimental**	**Grupo de control**	**Grupo experimental**
# total de casos de prueba	168	206	81	132	130	164
# total de fallos encontrados durante todo el ciclo de pruebas	52	65	62	70	37	63
# de casos de prueba que fallaron por las regresiones	41	53	24	47	57	83
# de fallos encontrados durante las pruebas de regresión	15	25	14	28	19	39

Como se observa en la Tabla 38, el patrón de mejora que se observó para la métrica de cobertura se mantuvo para la tasa de detección de fallos. De hecho, los datos reportados por los grupos de control (celdas resaltas en color amarillo en la Tabla 38) y los generados por los grupos experimentales (celdas resaltadas en color verde en la Tabla 38) muestran un cambio importante en la forma de trabajo. En congruencia con la métrica anterior, dada su relación con los casos de prueba, el grupo experimental de O3 destacó por un pequeño margen de la labor realizada por sus contrapartes de O2. Los resultados de aplicar las fórmulas (3) y (4) para evaluar la tasa de detección de fallos se muestran en la Tabla 39.

Tabla 39. Valores de las métricas de tasa de fallos en los casos de prueba y porcentaje de detección de fallos

Métrica	**Organizaciones participantes**					
	O1		**O2**		**O3**	
	Grupo de control	**Grupo experimental**	**Grupo de control**	**Grupo experimental**	**Grupo de control**	**Grupo experimental**
TFCP	24%	26%	30%	35%	43%	50%
PDF	28%	38%	23%	40%	51%	62%

Al analizar los datos obtenidos después del uso de Kuali, se observó que el grupo experimental obtuvo una TFCP del 50%, lo cual indica que los casos de prueba de regresión diseñados en la

herramienta condujeron a encontrar una buena cantidad de fallos antes de liberar el producto (véase Figura 4.4). Además, para esta misma organización, el PDF fue del 62% que, de acuerdo con la definición de la métrica, indica que el conjunto de casos de prueba de regresión que se diseñó funciona bien para identificar nuevos problemas generados con la modificación y/o adición de funcionalidades (celdas resaltadas en color verde en la Tabla 39). Se considera que estos valores podrían mejorarse considerablemente con el uso prolongado de Kuali. Por lo contrario, el grupo de control de O1 obtuvo la tasa más baja, lo que indica que el proceso tradicional para gestionar las pruebas de regresión no está siendo efectivo para detectar fallos en el producto antes de su entrega (celdas resaltadas en color rojo en la Tabla 39). No obstante, se puede observar que, para casos como éste, Kuali puede ayudar a mejorar considerablemente el rendimiento de los equipos de trabajo.

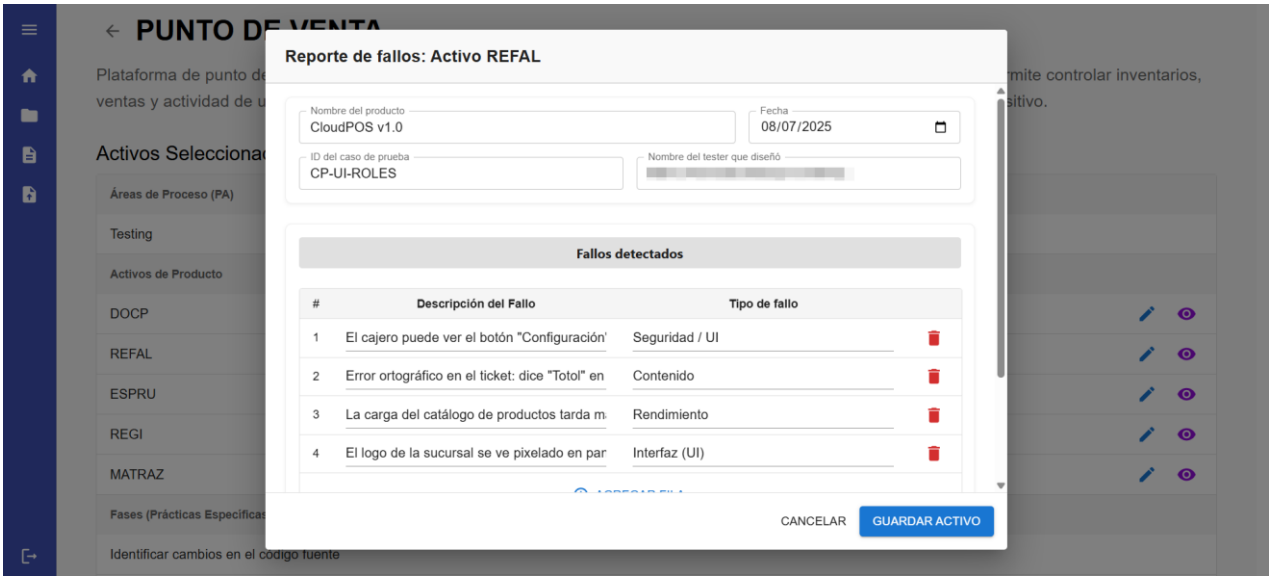


Figura 4.4. Ejemplo de reporte de fallos generado por el grupo experimental de O2 (vista de jefe de proyectos)

En este sentido, se concluyó que si bien la tasa de detección de fallos para el caso de las tres organizaciones presentó un aumento significativo, se requeriría del uso prolongado e institucionalizado de una nueva estrategia para gestionar las pruebas de regresión para asegurar que el producto sea liberado con el número mínimo de fallos.

4.3.3. Tiempo de ejecución vs cobertura

De acuerdo con Dias et al. (2023), asignar suficiente tiempo a las pruebas de regresión es crucial puesto que se estaría garantizando la estabilidad y calidad del software a lo largo del tiempo, previniendo la introducción de nuevos errores en funcionalidades existentes tras realizar cambios. De esta manera, detectar errores en las primeras etapas del desarrollo es significativamente menos costoso que corregirlos después del despliegue. En consecuencia, un tiempo de prueba suficiente minimiza el riesgo de fallos del sistema o filtraciones de datos que pueden tener impactos financieros y reputacionales significativos. En este escenario, el tiempo de ejecución y la cobertura en las pruebas de regresión son dos métricas interrelacionadas pero distintas. Por un lado, el tiempo de ejecución se refiere a la duración de las pruebas (i.e., tiempo tomado del plan original para la ejecución de las pruebas, incluidas las de regresión); mientras que, por otro lado, la cobertura, como ya se mencionó, mide el alcance de las pruebas de regresión sobre los requisitos modificados. Es decir, se trata de la evaluación del tiempo invertido en las pruebas de regresión con respecto a su eficiencia para la CRF

y el PDF que proporcionan. Por lo tanto, en el contexto de esta experimentación se buscó comprobar si existía una relación positiva entre el tiempo destinado a las pruebas y los valores recogidos para dos de las métricas anteriores, CRF y PDF. Mejor aún, se intentó determinar si Kuali ayudaba de alguna manera a facilitar la gestión de las pruebas en términos del tiempo usado. La Tabla 40 muestra la información recogida a través de Kuali, en el caso de los grupos experimentales, y con la planeación del proyecto entregada por los grupos de control.

Tabla 40. Valores analizados para evaluar la correspondencia entre el tiempo de ejecución y la cobertura

	Organizaciones participantes					
	O1		**O2**		**O3**	
	Grupo de control	**Grupo experimental**	**Grupo de control**	**Grupo experimental**	**Grupo de control**	**Grupo experimental**
Tiempo total destinado al proyecto (en semanas)	14	12	9	11	12	13
% de tiempo destinado a pruebas (incluidas las de regresión)	25%	35%	20%	32%	30%	37%
CRF	33%	85%	30%	52%	43%	83%
PDF	28%	38%	23%	40%	51%	62%

Se podrá observar que, en efecto, los datos recogidos con la experimentación muestran una relación directa entre el tiempo, la cobertura, y la tasa de detección puesto que mientras menos tiempo se destine a probar las modificaciones que se realicen a los requisitos o funcionalidades del software menores serán la CRF y el PDF. De hecho, en congruencia con las métricas anteriores, se observa que el grupo experimental de O3 obtuvo las mejores mediciones de las tres organizaciones participantes al haber destinado más tiempo de planificación a la gestión y realización de las pruebas (celdas resaltadas en color verde en la Tabla 40). Por el contrario, O2 reportó valores más bajos para la CRF y el PDF considerando que destinó menos tiempo para la realización y gestión de las pruebas. De acuerdo con las entrevistas realizadas, las y los integrantes de los tres grupos experimentales coincidieron en que Kuali facilitó el rastreo de los requisitos que se veían involucrados con las regresiones. Sin embargo, no se puede proporcionar información concluyente cuando la evaluación empírica solamente incluyó a tres organizaciones y seis equipos pequeños de trabajo.

4.3.4. Número de inconsistencias en trazabilidad

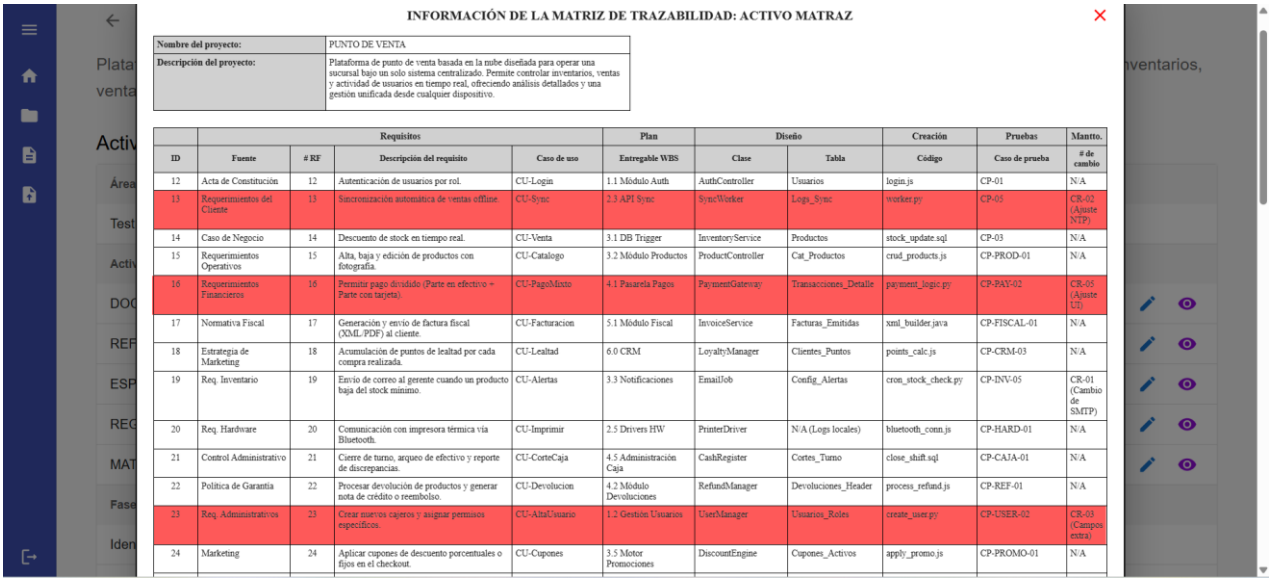
Las investigaciones de Rempel y Mäder (2016) y Charalampidou et al. (2023) argumentan que el número de inconsistencias en la trazabilidad de los requisitos durante las pruebas de regresión se refiere a la cantidad de fallos o discrepancias identificadas en dicha matriz, que indican que los cambios de código han afectado la funcionalidad existente o han introducido nuevos errores no cubiertos por los casos de prueba asociados. Por lo tanto, es crucial determinar la cantidad de discrepancias que existen entre la matriz de trazabilidad considerando el número de requisitos agregados y/o modificados y su relación con las regresiones. En este sentido, se analizó la información relacionada con las matrices de trazabilidad generadas por las y los participantes de los grupos en cada organización con el objetivo de determinar las inconsistencias reportadas por los equipos de trabajo.

Es importante mencionar que, en el caso del grupo de control O2, la "matriz de trazabilidad" que se creó en Trello no les permitió identificar las discrepancias que fueron generadas entre los requisitos a causa de las múltiples regresiones y, como consecuencia, fue imposible determinar un valor cuantitativo exacto. Como consecuencia, y a falta de evidencia documental, se decidió no incluir estos valores con el objetivo de no sesgar los resultados. Las discrepancias identificadas se muestran en la Tabla 41. Como se indicó anteriormente, con la intención de propiciar diferentes regresiones, el tesista envió solicitudes de cambios a través de la plataforma que regularmente involucraron grupos de modificaciones y/o adiciones de al menos cinco requisitos funcionales a lo largo del tiempo que duró el proyecto. Dichas modificaciones y/o adiciones se realizaron específicamente en cuatro diferentes momentos (regresiones en el caso de la Tabla 41) para cada equipo de trabajo.

Tabla 41. Número de incidencias reportadas a causa de las regresiones

	Organizaciones participantes					
	O1		**O2**		**O3**	
	Grupo de control	**Grupo experimental**	**Grupo de control**	**Grupo experimental**	**Grupo de control**	**Grupo experimental**
1ª regresión	13	9	-	7	19	6
2ª regresión	10	5	-	9	4	3
3ª regresión	8	6	-	3	12	5
4ª regresión	9	3	-	2	5	1
Cierre del proyecto	3	0	-	1	2	0

Considerando la evidencia recogida por Kuali, en el caso de los grupos experimentales de O1 y O3 se observó un registro apropiado de las discrepancias generadas con la modificación y/o adición de las funcionalidades registradas por la o el analista en la matriz, después de considerar las modificaciones y/o adiciones requeridas por el tesista (véase Figura 4.5).



INFORMACIÓN DE LA MATRIZ DE TRAZABILIDAD: ACTIVO MATRAZ

Nombre del proyecto:	PUNTO DE VENTA
Descripción del proyecto:	Plataforma de punto de venta basada en la nube diseñada para operar una sucursal bajo un solo sistema centralizado. Permite controlar inventarios, ventas y actividad de usuarios en tiempo real, ofreciendo análisis detallados y una gestión unificada desde cualquier dispositivo.

	Requisitos				Plan	Diseño		Creación	Pruebas	Mantto.
ID	Fuente	# RF	Descripción del requisito	Caso de uso	Entregable WBS	Clase	Tabla	Código	Caso de prueba	# de cambio
12	Acta de Constitución	12	Autenticación de usuarios por rol.	CU-Login	1.1 Módulo Auth	AuthController	Usuarios	login.js	CP-01	N/A
13	Requerimientos del Cliente	13	Sincronización automática de ventas offline.	CU-Sync	2.3 API Sync	SyncWorker	Logs_Sync	worker.py	CP-05	CR-02 (Ajuste NTP)
14	Caso de Negocio	14	Descuento de stock en tiempo real.	CU-Venta	3.1 DB Trigger	InventoryService	Productos	stock_update.sql	CP-03	N/A
15	Requerimientos Operativos	15	Alta, baja y edición de productos con fotografía.	CU-Catalogo	3.2 Módulo Productos	ProductController	Cat_Productos	crud_products.js	CP-PROD-01	N/A
16	Requerimientos Financieros	16	Permitir pago dividido (Parte en efectivo + Parte con tarjeta).	CU-PagoMixto	4.1 Pasarela Pagos	PaymentGateway	Transacciones_Detalle	payment_logic.py	CP-PAY-02	CR-05 (Ajuste UI)
17	Normativa Fiscal	17	Generación y envío de factura fiscal (XML/PDF) al cliente.	CU-Facturacion	5.1 Módulo Fiscal	InvoiceService	Facturas_Emitidas	xml_builder.java	CP-FISCAL-01	N/A
18	Estrategia de Marketing	18	Acumulación de puntos de lealtad por cada compra realizada.	CU-Lealtad	6.0 CRM	LoyaltyManager	Clientes_Puntos	points_calc.js	CP-CRM-03	N/A
19	Req. Inventario	19	Envío de correo al gerente cuando un producto baja del stock mínimo.	CU-Alertas	3.3 Notificaciones	EmailJob	Config_Alertas	cron_stock_check.py	CP-INV-05	CR-01 (Cambio de SMTP)
20	Req. Hardware	20	Comunicación con impresora térmica vía Bluetooth.	CU-Imprimir	2.5 Drivers HW	PrinterDriver	N/A (Logs locales)	bluetooth_conn.js	CP-HARD-01	N/A
21	Control Administrativo	21	Cierre de turno, arqueo de efectivo y reporte de discrepancias.	CU-CorteCaja	4.5 Administración Caja	CashRegister	Cortes_Turno	close_shift.sql	CP-CAJA-01	N/A
22	Política de Garantía	22	Procesar devolución de productos y generar nota de crédito o reembolso.	CU-Devolucion	4.2 Módulo Devoluciones	RefundManager	Devoluciones_Header	process_refund.js	CP-REF-01	N/A
23	Req. Administrativos	23	Crear nuevos cajeros y asignar permisos específicos.	CU-AltaUsuario	1.2 Gestión Usuarios	UserManager	Usuarios_Roles	create_user.py	CP-USER-02	CR-03 (Campos extra)
24	Marketing	24	Aplicar cupones de descuento porcentuales o fijos en el checkout.	CU-Cupones	3.5 Motor Promociones	DiscountEngine	Cupones_Activos	apply_promo.js	CP-PROMO-01	N/A

Figura 4.5. Reporte de inconsistencias generado por Kuali para el grupo experimental de O2 (vista de jefe de proyectos)

Este registro se genera considerando las dependencias existentes entre los requisitos, que se establecen al momento de darlos de alta en el conjunto de requisitos, y se identifican con el color que escoja el jefe de proyectos en la configuración de la herramienta. Aunado a esto, si se desea, el reporte puede generarse en formato .PDF, .XML, o .CSV para promover la interoperabilidad, la integridad de los datos y la flexibilidad. En resumen, los datos recogidos de los grupos experimentales de O1 y O3 demuestran que Kuali realizó una buena gestión de las incidencias, a tal grado que, al cierre del proyecto, no quedaba ninguna abierta. Sin embargo, vale la pena aclarar que una incidencia no es lo mismo que un fallo en el software. Sillito y Kutomi (2020) argumentan al respecto que una incidencia es un evento no planificado que causa una interrupción o reducción en la calidad del servicio de software, mientras que un fallo es la manifestación de un defecto que puede llevar a un incidente. Por lo tanto, Kuali se enfoca en promover una gestión eficiente de incidentes que promueve la resolución rápida del evento actual y puede ser que, como consecuencia, se facilite la gestión de los fallos. A modo de ejemplo, la Figura 4.6 muestra una imagen correspondiente al producto construido por el grupo experimental de O3 con el soporte para la gestión de las pruebas de regresión de Kuali. Se decidió mostrar puesto que, cómo se podrá observar, presenta tanto defectos de interfaz (i.e., palabras escritas incorrectamente) como fallos (i.e., carencia del cobro de IVA), evidenciando que, en la actualidad, ningún producto de software está exento de defectos y/o fallos).

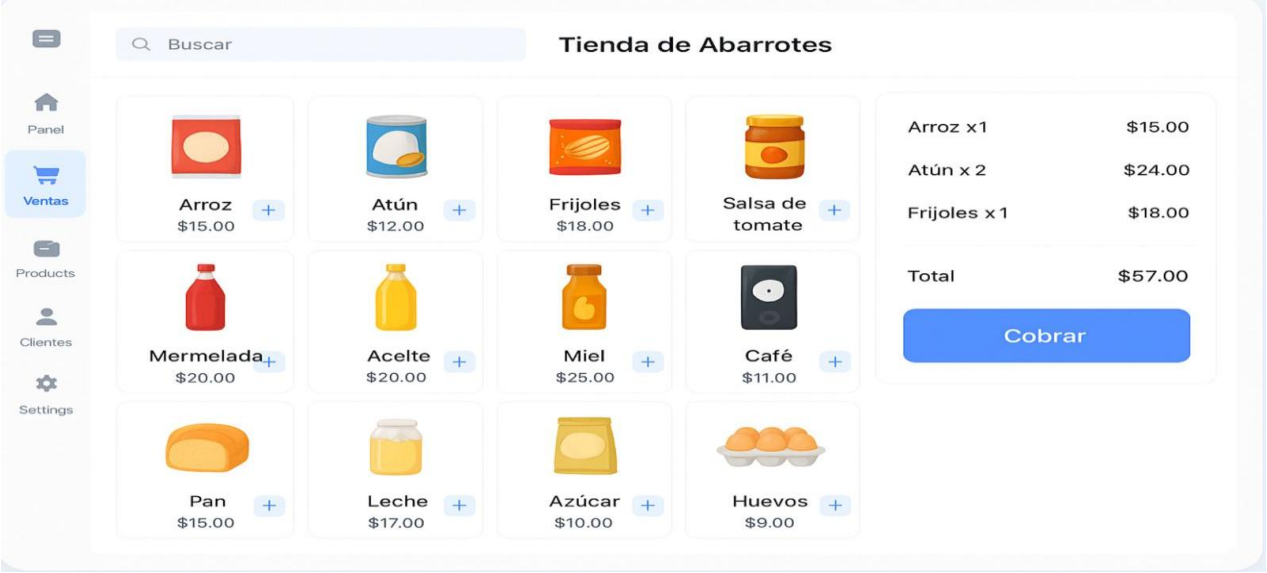


Figura 4.6. Imagen del software creado por el grupo experimental de O3 como parte de la evaluación empírica

Una vez que se cerraron los proyectos, se procedió a que las y los participantes de los grupos experimentales respondieran el cuestionario mostrado en la Tabla 34, con el objetivo de recoger información relacionada con sus percepciones sobre la efectividad de Kuali en sus contextos de trabajo. En la siguiente sección se presentan los hallazgos alcanzados.

4.4. Resultados cualitativos

El objetivo de la evaluación cualitativa fue recoger información que permitiera analizar si se generó, o no, una percepción positiva en las y los participantes que utilizaron Kuali durante el desarrollo de los proyectos (i.e., un total de 12 participantes). Por lo tanto, carecía de todo sentido incluir a las y los participantes del grupo de control. Así pues, el cuestionario mostrado en la Tabla 42 recogió las percepciones de las y los integrantes de los grupos experimentales de las tres

organizaciones sobre la estrategia que estaba detrás de Kuali, dado que es lo fundamental para este tema de tesis.

Tabla 42. Percepciones de las y los participantes después de la evaluación empírica sobre la estrategia propuesta

	Ítem	Respuestas					M	DE
		TDA	DA	N	ED	TED		
Metodología y estrategia	I1. La estrategia seguida es clara para definir y gestionar los casos de prueba de regresión.	9.0	3.0	0.0	0.0	0.0	1.8	0.4
	I2. La estrategia facilita la gestión tanto de las pruebas manuales como las automatizadas.	0.0	7.0	5.0	0.0	0.0	0.6	0.5
	I3. La estrategia se adecua a la frecuencia con la que se realizan las pruebas de regresión en mi organización.	4.0	8.0	0.0	0.0	0.0	1.3	0.5
	I4. La estrategia me da el soporte adecuado para decidir qué casos de prueba debo re-ejecutar después de un cambio en el código.	0.0	12.0	0.0	0.0	0.0	1.0	0.0
Cobertura	I5. Considero que la estrategia propicia una buena cobertura de código a través de la gestión de las pruebas de regresión.	5.0	7.0	0.0	0.0	0.0	1.4	0.5
	I6. La estrategia facilita la cobertura de los requisitos funcionales a través de su trazabilidad.	8.0	4.0	0.0	0.0	0.0	1.7	0.5
	I7. La estrategia contribuye a la evaluación de la cobertura que se tiene sobre los cambios a los requisitos y los casos de prueba.	11.0	1.0	0.0	0.0	0.0	1.9	0.3
Detección y seguimiento de fallos	I8. Estoy conforme con la cantidad de fallos que fueron detectados en producción después de la ejecución de las pruebas de regresión.	7.0	5.0	0.0	0.0	0.0	1.6	0.5
	I9. Considero que la tasa de detección de fallos durante las pruebas de regresión es apropiada.	12.0	0.0	0.0	0.0	0.0	2.0	0.0
	I10. La estrategia contribuye a rastrear y priorizar los fallos detectados.	2.0	10.0	0.0	0.0	0.0	1.2	0.4
	I11. La estrategia contribuye a identificar fallos críticos durante la ejecución de las pruebas de regresión.	7.0	5.0	0.0	0.0	0.0	1.6	0.5
Automatización y mejora	I12. La estrategia puede alinearse con las herramientas usadas para automatizar las pruebas de regresión.	0.0	1.0	8.0	2.0	1.0	-0.3	0.7
	I13. La estrategia facilita la automatización de los casos de prueba de regresión diseñados.	0.0	0.0	0.0	10.0	2.0	-1.2	0.4
	I14. Las métricas definidas para medir la efectividad de las pruebas de regresión son apropiadas.	1.0	11.0	0.0	0.0	0.0	1.1	0.3
	I15. La estrategia facilita la obtención de un análisis post-mortem de los ciclos de prueba de regresión para identificar oportunidades de mejora.	8.0	4.0	0.0	0.0	0.0	1.7	0.5

En el contexto de los casos de estudio que se desarrollan como parte de las investigaciones sobre Ingeniería de Software, las percepciones personales son cruciales porque proporcionan una

comprensión profunda e individualizada del fenómeno estudiado, lo cual es un objetivo central de la investigación cualitativa (Falessi et al., 2018). Por lo tanto, las percepciones individuales de las y los participantes ofrecen una visión de su realidad vivida, que los datos puramente objetivos o cuantitativos podrían pasar por alto. La Tabla 42 recoge estas impresiones. La interpretación de los valores mostrados en dicha tabla es la siguiente: si el valor de M se encuentra entre los valores de 1.0 y 2.0 (i.e., dados los valores cuantitativos de la escala de Likert usada, que van de -2.0 a 2.0) más positiva es la percepción del encuestado. En este sentido, si el valor de M es negativo se trata evidentemente de que las respuestas promedio tienden hacia el extremo negativo o de desacuerdo de la escala, indicando una percepción o actitud general desfavorable hacia el tema evaluado. En el caso de la DE, si el valor se aleja de manera significativa de 0.0 se refleja una alta discrepancia entre las respuestas proporcionadas por las y los participantes. Para este caso, y por definición de la fórmula, no existe la posibilidad de que este valor sea negativo.

Consecuentemente, considerando los ítems I1, I3 e I4 de la categoría de "Metodología y estrategia", por ejemplo, las percepciones de las y los participantes fueron positivas manteniéndose entre los rangos de 1.0 a 1.8 para la M y de 0.0 a 0.5 para la DE (véase Figura 4.7). De hecho, el ítem "*I4. La estrategia me da el soporte adecuado para decidir qué casos de prueba debo re-ejecutar después de un cambio en el código*" destacó por alcanzar un total acuerdo entre los encuestados (M=1.0, DE=0.0). Sin embargo, para el ítem "*I2. La estrategia facilita la gestión tanto de las pruebas manuales como las automatizadas*" se registraron las peores percepciones de la categoría (M=0.6, DE=0.5), puesto que no hubo acuerdo entre las percepciones de las y los participantes encuestados. Una realidad es que esta versión inicial de Kuali no contiene funcionalidades que faciliten la conexión directa con herramientas de código abierto, principalmente, que permitan la automatización de las pruebas de software, de tal forma que se pudiera facilitar su gestión. Por lo tanto, se asume que esta es una limitante de Kuali que fue correctamente identificada por las y los participantes y que, además, se podría abordar más adelante con el objetivo de mejorar la implementación de la estrategia propuesta.

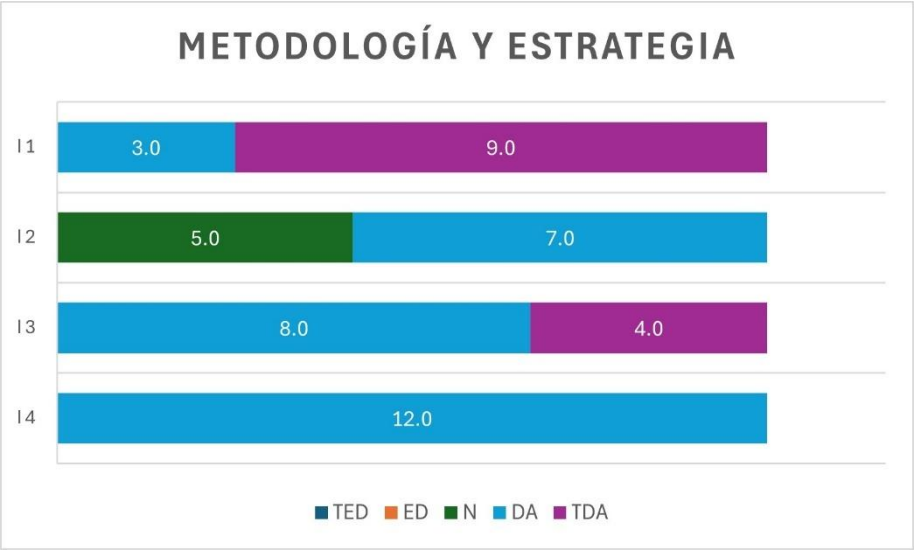


Figura 4.7. Percepciones de los grupos experimentales sobre la categoría de metodología y estrategia

Con relación a la categoría de "Cobertura", todas las percepciones de las y los participantes se mantuvieron del lado positivo de la escala, demostrando coherencia con los resultados cuantitativos mostrados como resultado de la experimentación (véase Figura 4.8). En consecuencia, el único aspecto a destacar para esta categoría fue el alto nivel de consenso que se obtuvo de las y los

participantes al considerar que la estrategia propicia una buena cobertura de código, requisitos funcionales y casos de prueba cuando se generan regresiones.

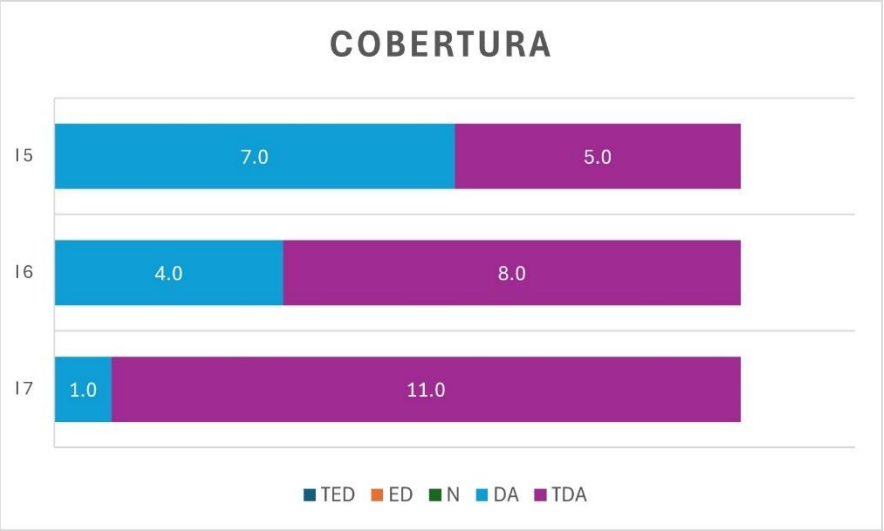


Figura 4.8. Percepciones de los grupos experimentales sobre la categoría de cobertura

Ahora bien, en la categoría de "Detección y seguimiento de fallos" destacó el ítem "*I9. Considero que la tasa de detección de fallos durante las pruebas de regresión es apropiada*" puesto que, de hecho, obtuvo el valor más alto de todo el cuestionario (M=2.0, DE=0.0). Una vez más, las valoraciones de los ítems de esta categoría confirmaron los resultados experimentales que se obtuvieron con la experimentación, puesto que se relacionan directamente con la gestión de la matriz de trazabilidad y el registro y seguimiento de los fallos que promueven la estrategia para mejorar la detección de fallos (i.e., métricas de TFCP y PDF). La Figura 4.9 resume todas las percepciones de las y los participantes sobre esta categoría.

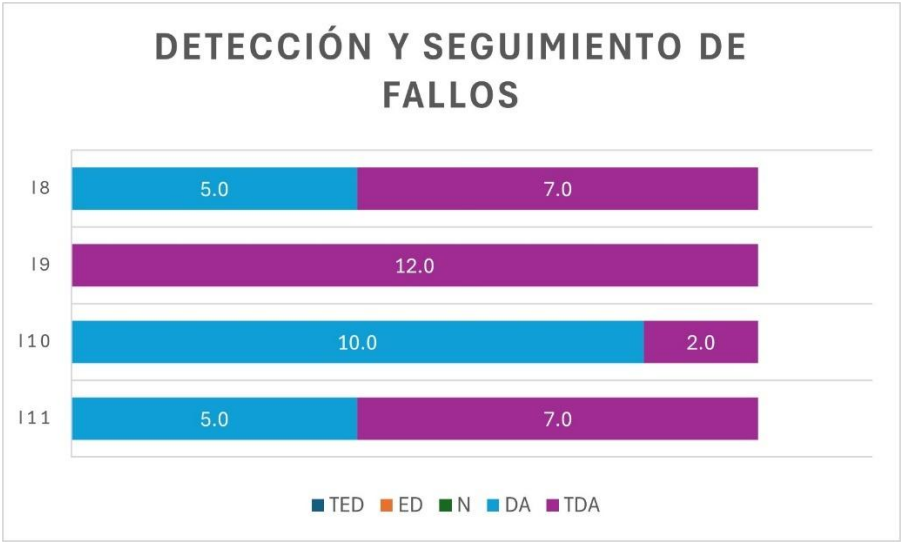


Figura 4.9. Percepciones de los grupos experimentales sobre la categoría de detección y seguimiento de fallos

Finalmente, las percepciones más negativas se dieron para la categoría de "Automatización y mejora". De acuerdo con la clasificación que Engström y Runeson (2010) y Hynninen y Jantunen (2022) hacen para los ítems de esta categoría, se debe demostrar que la estrategia, proceso, marco de

trabajo y/o herramienta que se implemente para eficientar las pruebas de regresión facilite la ejecución de las prácticas relacionadas con el aseguramiento de la calidad, y de las pruebas en sí, a través de la automatización. Desafortunadamente, la estrategia propuesta no satisface los dos ítems de esta categoría que se relacionan con la automatización; de hecho, son los que tuvieron las valoraciones más bajas (véase Figura 4.10). Para el ítem "*I12. Las estrategia puede alinearse con las herramientas usadas para automatizar las pruebas de regresión*", por ejemplo, tres de las y los participantes se mantuvieron en el lado negativo de la escala, ocho se mantuvieron neutrales, y solamente uno estuvo de acuerdo. Al indagar sobre estas respuestas, se comprobó que las y los participantes percibieron, con toda razón, que esta versión de Kuali no fue diseñada para integrarse con otras herramientas de código abierto. No obstante, si bien esto representa una desventaja de la herramienta, la estrategia que se propuso en esta tesis propone ayudar a todas las pequeñas organizaciones desarrolladoras de software, incluidas las que, por falta de conocimiento y/o experiencia, desconocen qué herramientas computacionales podrían integrar para mejorar la gestión de las pruebas de regresión o, incluso, aquellas que no realicen las pruebas de regresión pero que desean iniciar con la definición de un proceso para tal fin. De manera similar, para el ítem "*I13. La estrategia facilita la automatización de los casos de prueba de regresión diseñados*" se observó mayor consenso entre los participantes, puesto que el 83% de estas y estos no considera que Kuali facilite la ejecución automática de los casos de prueba regresión diseñados. Desafortunadamente, esto también es verdad, pero se pretende generar a futuro una versión mejorada de la herramienta que explore la automatización de esta actividad y, como consecuencia, mejore la efectividad de la estrategia de gestión.

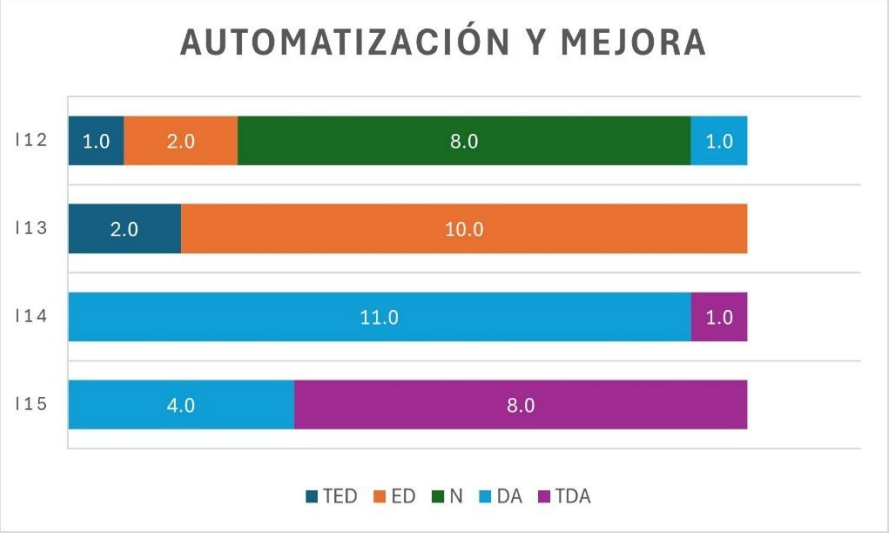


Figura 4.10. Percepciones de los grupos experimentales sobre la categoría de automatización y mejora

4.5. Comprobación de la hipótesis de la tesis

Considerando la evidencia mostrada en las secciones 4.3 y 4.4, principalmente considerando los resultados cuantitativos, se concluye que la hipótesis establecida al inicio de esta tesis de la siguiente forma:

Hi: *"La implementación de una estrategia de gestión de las pruebas de regresión, en una pequeña organización desarrolladora de software, permitirá mejorar su efectividad".*

puede ser aceptada considerando los siguientes argumentos:

1. La información cuantitativa recogida a través de la evaluación empírica, relacionada con cuatro de las métricas definidas por Wong et al. (1997), Do et al., (2005) y Elsner et al. (2021), muestra que, si bien no se alcanza la cobertura ideal establecida por la literatura, la estrategia propuesta sí mejoró la efectividad de las pruebas de regresión en las tres organizaciones participantes. En este sentido, se puede asumir que, el uso prolongado y controlado de dicha estrategia, podría contribuir a mejorar sustancialmente esta efectividad.
2. La información cualitativa recogida a través de la aplicación del cuestionario de evaluación, relacionado con las categorías establecidas por Engström y Runeson (2010) y Hynninen y Jantunen (2022), muestra que las percepciones de las y los participantes de los grupos experimentales son positivas en el 80% de los ítems del cuestionario de evaluación. Este hecho es fundamental en la evaluación empírica realizada, puesto que proporciona una comprensión profunda y holística sobre la efectividad de la estrategia propuesta, revelando el significado que está tendría en los contextos reales desde la perspectiva de las y los participantes.

Es de suma importancia aclarar que los resultados mostrados en este capítulo únicamente permiten validar la hipótesis establecida para el contexto de esta tesis de grado. Es decir, de ninguna manera debe entenderse que la efectividad que demostró Kuali para implementar una estrategia que gestiona las pruebas de regresión en tres organizaciones pequeñas, puede generalizarse a organizaciones más grandes, o incluso del mismo tamaño, que desarrollen cualquier tipo de proyectos con equipos de cualquier cantidad de integrantes. Es evidente que afirmaciones de este tipo deben sustentarse con más evaluaciones empíricas en una cantidad más grande de organizaciones participantes y, por ende, quedan fuera de la aportación que se hace con esta tesis de grado.

4.6. Amenazas a la validez de los resultados

Como se pudo observar, se decidió no utilizar una prueba estadística para validar la hipótesis de esta tesis, ya que la evaluación empírica presentada se enfocó únicamente en el análisis del rendimiento de dos grupos (i.e., control y experimental) al desarrollar "proyectos hermanos". Sin embargo, teniendo en cuenta las recomendaciones de Wohlin et al., (2024) para el diseño y conducción de casos de estudios en la Ingeniería de Software, se identificaron las principales amenazas a la validez de los resultados presentados en las secciones anteriores:

- Validez interna. En el contexto de un caso de estudio, los problemas de validez interna se relacionan principalmente con la posibilidad de establecer cuestiones causales entre las variables y los resultados. En la evaluación presentada en esta tesis, se garantizó que ninguno de las y los participantes conocieran previamente la preparación del caso de estudio, con el fin de evitar modificaciones en sus prácticas habituales y prevenir sesgos en la recogida de los datos cuantitativos. En consecuencia, no existieron incentivos internos que pudiesen influir en los resultados. No obstante, no es posible asegurar que las y los integrantes de los grupos experimentales no hayan mostrado más entusiasmo en el uso de Kuali, ni que dichos grupos contaran con integrantes con mayores capacidades cognitivas que las y los de los grupos de control, ni que el uso de los activos facilitara una implementación más eficiente de la estrategia de gestión.
- Validez de constructo. Los problemas de validez de constructo en un caso de estudio surgen cuando existen errores en la medición o en la recolección de datos. En este sentido, en el presente trabajo se proporcionó evidencia de que los datos cuantitativos fueron recopilados

mediante los activos implementados en la herramienta denominada Kuali, mientras que los datos cualitativos se obtuvieron a través de cuestionarios y entrevistas. Esto permitió garantizar que los resultados fueran consistentes independientemente de la persona encargada de analizarlos. Sin embargo, cabe señalar que los datos obtenidos reflejan únicamente el comportamiento de las y los integrantes de los grupos experimentales durante el desarrollo de los "proyectos hermanos"; por ende, no es posible asegurar que los mismos resultados pudiesen ser replicados en otros proyectos ajenos al contexto de esta tesis.

- Validez externa. Finalmente, es posible que hayan surgido limitaciones de validez externa derivadas de las características del caso de estudio, entre ellas: el tamaño de las organizaciones participantes (i.e., pequeñas con no más de 25 colaboradores), el número de participantes involucrados en la evaluación de la estrategia (i.e., cuatro por cada grupo experimental), el tamaño y complejidad del proyecto (i.e., proyecto pequeño con pocas funcionalidades) y el dominio de aplicación (i.e., un punto de venta para una tienda pequeña).

5. Conclusiones

Las pruebas de regresión representan la columna vertebral de la Ingeniería de Software moderna, siendo indispensables en todo el ciclo de vida para garantizar la calidad y estabilidad del producto. Su función trasciende la simple detección de errores, puesto que actúan como un mecanismo de aseguramiento que valida que las modificaciones al código no alteren negativamente las funcionalidades preexistentes ni reintroduzcan fallos previamente corregidos. Desde una perspectiva de gestión, su implementación adecuada optimiza la relación costo-beneficio del desarrollo. Al mitigar el riesgo de comportamientos imprevisibles y asegurar la compatibilidad entre versiones, estas pruebas aceleran los ciclos de liberación y reducen los costos asociados a correcciones tardías. Asimismo, la automatización se presenta como una solución clave para manejar la carga de trabajo en productos con actualizaciones frecuentes.

Sin embargo, el contexto industrial plantea desafíos significativos. El crecimiento acumulativo de las suites de pruebas hace que la estrategia de ejecución total (i.e., *retest-all*) sea económicamente insostenible. Esto ha forzado la adopción de estrategias más inteligentes, como la selección, priorización y reducción de los casos de prueba. A pesar de la existencia de estas técnicas avanzadas, la brecha de recursos financieros y humanos, especialmente en las organizaciones pequeñas, conduce a la omisión de estrategias formales. Se evidencia, por tanto, una necesidad crítica de estrategias de gestión que no solo sean técnicamente robustas, sino viables dentro de las capacidades operativas de este tipo de organizaciones.

En este sentido, la relevancia de las pruebas de regresión se magnifica en las pequeñas organizaciones desarrolladoras de software dado que, en contextos como el de México donde estas organizaciones representan entre el 95% y el 100% del mercado, la gestión de la calidad deja de ser un requisito técnico para convertirse en un factor de supervivencia económica. Es decir, a diferencia de las grandes corporaciones, las organizaciones pequeñas operan bajo estrictas limitaciones presupuestarias y suelen carecer de procesos formales, dependiendo así del esfuerzo individual. La vulnerabilidad de estas organizaciones radica en que cualquier modificación al código conlleva un riesgo latente. Consecuentemente, sin una red de seguridad adecuada, la baja calidad precipita altos índices de fallos, obligando a destinar recursos excesivos al retrabajo. Esta dinámica incrementa los costos operativos, diluye el retorno de inversión e introduce incertidumbre sobre la estabilidad del producto. Paradójicamente, aunque las pruebas de regresión consumen aproximadamente el 30% del esfuerzo total de las pruebas en estas organizaciones, su ejecución suele ser ineficiente. La falta de análisis, planificación y políticas claras deriva en ejecuciones manuales que no siempre aportan valor. Al omitir una estrategia preventiva para evitar retrasos inmediatos, las organizaciones pequeñas terminan asumiendo costos laborales insostenibles a largo plazo y riesgos desproporcionados de liberar software defectuoso.

Por lo tanto, la dificultad de implementar pruebas de regresión efectivas en estas organizaciones es un problema multifactorial derivado de las siguientes limitaciones:

- Restricción de recursos: Es la barrera más crítica. Los presupuestos ajustados que se tienen en las organizaciones pequeñas impiden la adquisición de software especializado, infraestructura robusta o consultoría. La automatización, cuya inversión inicial es alta y de retorno incierto si no se gestiona bien, se vuelve inalcanzable. Como resultado, se depende de pruebas manuales costosas y se carece de personal capacitado en tendencias modernas como Inteligencia Artificial o minería de datos.
- Inmadurez de procesos: La mayoría de las organizaciones pequeñas no cuentan con procesos institucionalizados. Esta falta de madurez lleva a la omisión de estrategias de prueba, resultando en esfuerzos no planificados y una deficiencia en el proceso de Aseguramiento de la Calidad del Software. Aunado a esto, las técnicas avanzadas de la literatura (e.g., minimización o priorización compleja) resultan inaplicables o no escalables en estos entornos.
- Cultura y presión de tiempo: La industria opera con tiempos de entrega limitados que suelen sacrificar la calidad por velocidad. Existe la falsa creencia de que probar retrasa la entrega. Además, la falta de documentación histórica impide la reutilización del conocimiento, fomentando una dependencia del "heroísmo" de las y los empleados y la improvisación.
- Analogía: Esta situación es comparable a intentar construir un rascacielos con un equipo pequeño y sin planos arquitectónicos. El proceso de prueba equivale a verificar la estructura: sin herramientas adecuadas y sin una estrategia clara, cada nuevo piso (i.e., cambio de código) requiere un esfuerzo manual agotador y aumenta el riesgo de que la estructura completa colapse bajo la presión de los plazos de entrega.

Considerando este contexto, la presente tesis ha abordado un escenario que reconoce que las técnicas avanzadas existentes no son factibles para las pequeñas organizaciones desarrolladoras de software debido a la falta de recursos y especialización que prevalece en éstas. En respuesta, se diseñó e implementó una estrategia de gestión que se articula en tres elementos esenciales para promover la calidad del proceso y del producto: gestión del conocimiento, mejora de los procesos de software, y soporte computacional. Estos elementos se materializan en dos componentes principales:

- PAL-RT: Fundamentada en la gestión del conocimiento, esta biblioteca actúa como un repositorio para crear, compartir y retener el saber adquirido, reduciendo la repetición de errores. Su arquitectura conceptual se estructura en tres capas:
    - N1 (Pruebas de regresión): Define un proceso consensuado producto de un análisis profundo de la literatura y necesidades de 135 organizaciones pequeñas.
    - N2 (Desarrollo del producto): Contiene activos reutilizables (e.g., actividades, tareas, plantillas, guías).
    - N3 (Soporte organizacional): Alinea las pruebas con los objetivos de negocio y documenta las lecciones aprendidas.
- Kuali: Esta herramienta facilita la adopción, ejecución y medición de la estrategia de manera no invasiva. Permite una maduración gradual del proceso, agilizando tareas críticas como la priorización de pruebas y la documentación de resultados, tal como validaron las y los participantes del diagnóstico realizado y detallado en los capítulos anteriores.

En esencia, esta tesis ha contribuido con la definición de una estratega de gestión que se enfocó en tres pequeñas organizaciones desarrolladoras de software al proveerles un marco de trabajo estructurado (i.e., PAL-RT) y una tecnología facilitadora (i.e., Kuali), que se ajustaron estrictamente a sus limitaciones operativas. Por ende, la principal contribución de esta investigación es el desarrollo de un camino viable para que las organizaciones pequeñas gestionen sus pruebas de regresión, llenando un vacío en la literatura que históricamente ha favorecido a grandes corporaciones. Dicha contribución se puede distinguir por tres ejes de aportación:

1. Un marco de trabajo estructurado y adaptado: Como ya se indicó, a través de PAL-RT, se formaliza una estrategia compuesta por ocho fases clave que no depende del "heroísmo" individual sino que se fundamenta en actividades, tareas, roles, activos de proceso y métricas que inculcan una cultura de documentación eficiente y facilitan el control de riesgos derivado de las regresiones. Dicha estrategia es explicada detalladamente en el Capítulo 3 de la tesis.
2. Facilidad de adopción (i.e., reducción de intrusión): Reconociendo la resistencia al cambio en las organizaciones, se creó Kuali para facilitar la implementación de la estrategia creada. De esta manera, la organización se puede enfocar en la efectividad, priorizando conjuntos de pruebas basados en severidad (i.e., impacto y probabilidad) para maximizar la detección temprana de fallos críticos, optimizando así el esfuerzo destinado a las pruebas de regresión.
3. Base para la mejora continua: Se ha creado una estrategia que permite a la organización madurar gradualmente. Evidencia de lo anterior es que el Capítulo 4 se ha centrado en presentar puntualmente el diseño y conducción de un caso de estudio con enfoque mixto (i.e., cuantitativo y cualitativo) que facilita la replicabilidad de la investigación. En este sentido, se ha mostrado información sobre la mejora significativa que se presentó en las organizaciones participantes después de implementar la estrategia de gestión y se considera que, de continuar usándola, se lograría mantener un proceso de mejora continua.

En conclusión, esta tesis de grado ha contribuido a la investigación que se hace sobre Ingeniería de Software al introducir una solución sostenible para que el sector mayoritario de la industria (i.e., las organizaciones pequeñas) cuente con una alternativa para definir, controlar y madurar su proceso de gestión de las pruebas de regresión, a pesar de sus limitaciones. Sin embargo, también se reconoce que la estrategia podría mejorarse considerablemente con trabajo futuro, por ejemplo:

- Analizar herramientas de código abierto para generar un *plug-in* que permita la integración de la estrategia de gestión con el fin de facilitar la ejecución automatizada de los casos de prueba de regresión.
- Explorar el uso de algoritmos de la Inteligencia Artificial para priorizar y seleccionar aquellos casos de prueba de regresión que conduzcan a detectar, de maneras más eficiente, fallos generados con las modificaciones continuas al código.
- Aumentar la cantidad de evaluaciones empíricas para generar evidencia más concluyente sobre la estrategia propuesta y sus componentes.

6. Bibliografía

Abdullah, R., Eri, Z. D., & Talib, A. M. (2011, December). A model of knowledge management system in managing knowledge of software testing environment. In 2011 Malaysian Conference in Software Engineering (pp. 229-233). IEEE.

Abdulwareth, A. J., & Al-Shargabi, A. A. (2021). Toward a multi-criteria framework for selecting software testing tools. IEEE Access, 9, 158872-158891.

Aizprua, S., Ortega, A., & Von Chong, L. (2019). Calidad del software una perspectiva continua. Centros: Revista Científica Universitaria, 8(2), 120-134.

Akhtar, A., Bakhtawar, B., & Akhtar, S. (2022). Extreme programming vs scrum: A comparison of agile models. International Journal of Technology Innovation and Management (IJTIM), 2(2), 80-96.

Ali, N. B., Engström, E., Taromirad, M., Mousavi, M. R., Minhas, N. M., Helgesson, D., Kunze, S., & Varshosaz, M. (2019). On the search for industry-relevant regression testing research. Empirical Software Engineering, 24, 2020-2055.

Ali, S., Hafeez, Y., Hussain, S., & Yang, S. (2020). Enhanced regression testing technique for agile software development and continuous integration strategies. Software Quality Journal, 28, 397-423.

Aman, H., Nakano, T., Ogasawara, H., & Kawahara, M. (2018, April). A topic model and test history-based test case recommendation method for regression testing. In 2018 IEEE international conference on software testing, verification and validation workshops (ICSTW) (pp. 392-397). IEEE.

Anand, A., & Uddin, A. (2019). Importance of software testing in the process of software development. International Journal for Scientific Research and Development, 12(6), 141-145.

Aniche, M. (2022) Effective Software Testing: A Developer's Guide. New York, NY: Manning Publications.

Anwar, Z., Afzal, H., Bibi, N., Abbas, H., Mohsin, A., & Arif, O. (2019). Un sistema de inferencia neuro-difuso híbrido-adaptativo para la optimización de conjuntos de pruebas de regresión multiobjetivo. Computación Neuronal y Aplicaciones, 31, 7287-7301.

Arcilla-Cobián, M., San Feliu Gilabert, T., Feliz, A., & Calvo-Manzano Villalón, J. A. (2017). Implementing a process asset library focused on IT service capacity management. International Journal of Information Systems and Software Engineering for Big Companies,(IJISEBC), 4(2), 43-51.

Argüello, M. (2020). Propuesta de gestión para la adaptación de una metodología de grandes empresas para las pruebas de software en PyMEs. Tesis de la Maestría en Ingeniería de gestión empresaria. Universidad Nacional de Rosario.

Azizi, M. (2021, April). A tag-based recommender system for regression test case prioritization. In 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 146-157). IEEE.

Bajaj, A., & Sangwan, O. P. (2020). Nature-inspired approaches to test suite minimization for regression testing. In: Bansal, A., Jain, A., Jain, S., Jain, V., Choudhary, A. (Eds.), Computational Intelligence Techniques and Their Applications to Software Engineering Problems (pp. 99-110). Boca Raton, FL: CRC Press.

Baltes, S., & Ralph, P. (2022). Sampling in software engineering research: A critical review and guidelines. Empirical Software Engineering, 27(4), 94.

Bakhtiary, V., Gandomani, T. J., & Salajegheh, A. (2020). The effectiveness of test-driven development approach on software projects: A multi-case study. Bulletin of Electrical Engineering and Informatics, 9(5), 2030-2037.

Baumgartner, M., Klonk, M., Pichler, H., Seidl, R., & Tanczos, S. (2021). Agile Testing. Cham, Germany: Springer International Publishing.

Bermón-Angarita, L., Amescua-Seco, A., Sánchez-Segura, M. I., & García-Guzmán, J. (2009). Software process asset libraries using knowledge repositories. In: Theng, Y. L., Foo, S., Goh, D., Na, J. C. (Eds.), Handbook of Research on Digital Libraries: Design, Development, and Impact (pp. 465-475). New York, NY: IGI Global.

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., & Wiswedel, B. (2009). KNIME - The Konstanz information miner: version 2.0 and beyond. ACM SIGKDD explorations Newsletter, 11(1), 26-31.

Bertolino, A., Guerriero, A., Miranda, B., Pietrantuono, R., & Russo, S. (2020, June). Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 1-12).

Bird, S. (2006, July). NLTK: The natural language toolkit. In Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions (pp. 69-72).

Bizyukov, P. E., Radchenko, D. S., Tokarev, A. B., Matveev, B. V., Povetko, V. N., & Slinchuk, S. A. (2020, May). Automation of full-scale regression testing of radio monitoring equipment. In IOP Conference Series: Materials Science and Engineering (Vol. 862, No. 5, p. 052018). IOP Publishing.

Blanquicett, L. A., Bonfante, M. C., & Acosta-Solano, J. (2018). Prácticas de pruebas desde la industria de software. La plataforma ASISTO como caso de estudio. Información Tecnológica, 29(1), 11-18.

Braz, L., Fregnan, E., Arora, V., & Bacchelli, A. (2022, September). An exploratory study on regression vulnerabilities. In Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 12-22). IEEE.

Butool, R., Nadeem, A., Sindhu, M., & Uz Zaman, O. (2019, November). Improving requirements coverage in test case prioritization for regression testing. In 2019 22nd International Multitopic Conference (INMIC) (pp. 1-6). IEEE.

Candal, D. M., Gaspar, M. A., Costa, I., de Magalhães, F. L. F., & Ferreira, R. A. (2022). Influencia de las prácticas de gestión del conocimiento aplicadas al desarrollo de software ágil. Revista Ibérica de Sistemas y Tecnologías de Información, (48), 74-88.

Catal, C. (2012). The ten best practices for test case prioritization. In: Skersys, T., Butleris, R., Butkiene, R. (Eds.), Communications in Computer and Information Science (pp. 452-459). Berlin, Heidelberg: Springer Berlin Heidelberg.

Catal, C., & Mishra, D. (2013). Test case prioritization: A systematic mapping study. Software Quality Journal, 21, 445-478.

Charalampidou, S., Ampatzoglou, A., Karountzos, E., & Avgeriou, P. (2021). Empirical studies on software traceability: A mapping study. Journal of Software: Evolution and Process, 33(2), e2294.

Chaudhary, S., Choudhary, A., & Seth, J. (2023, January). Nature inspired approaches for test case selection in regression testing: A review. In 2023 13th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp. 644-649). IEEE.

Chen, Y. (2021, May). NodeSRT: A selective regression testing tool for Node.js application. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) (pp. 126-128). IEEE.

Chen, C. Y., & Lee, J. C. (2022). Comparative effects of knowledge-based antecedents in different realms of CMMI-based software process improvement success. Computer Standards & Interfaces, 81, 103599.

Chernov, V., Dorokhova, L., Dorokhov, O., & Egorovskaya, G. (2019). Decision support system choosing software testing strategy. Bulletin of the Transylvania University of Brasov. Series III: Mathematics and Computer Science, 457-468.

Chopra, R. (2018). Software quality assurance: A self-teaching introduction. Dulles, VA: Mercury Learning and Information.

Chrissis, M. B., Konrad, M., & Shrum, S. (2011). CMMI for Development: Guidelines for Process Integration and Product Improvement. New York, NY: Pearson Education.

Cieza, E., & Tantajulca, N. M. (2023). Desarrollo de un proceso de pruebas basada en estándares para mejorar la eficiencia de la verificación y validación del producto en micro empresas peruanas que desarrollan software. Tesis de licenciatura. Universidad Señor de Sipan.

Colmerauer, A. (1990). An introduction to Prolog III. Communications of the ACM, 33(7), 69-90.

Cornide-Reyes, H., Madrigal, G., Muñoz, G., Duran, C., Jorquera, J., & Morales, J. (2024). Analysis of the use of software process improvement models in agile development. Ingeniare: Revista Chilena de Ingenieria, 32, 1-11.

Costa, D. M., Teixeira, E. N., & Werner, C. M. L. (2020, December). A repository to support software process reuse based on process lines. In Proceedings of the XIX Brazilian Symposium on Software Quality (pp. 1-10).

Coviello, C., Romano, S., & Scanniello, G. (2018, October). An empirical study of inadequate and adequate test suite reduction approaches. In Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement (pp. 1-10). IEEE.

Creswell, J. W. (2021). A concise introduction to mixed methods research. Boston, MA: SAGE Publications.

Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M., & Zupan, B. (2013). Orange: Data mining toolbox in Python. The Journal of machine Learning research, 14(1), 2349-2353.

Dias, T., Batista, A., Maia, E., & Praça, I. (2023, July). Testlab: An intelligent automated software testing framework. In International Symposium on Distributed Computing and Artificial Intelligence (pp. 355-364). Cham: Springer Nature Switzerland.

Diepenbeck, M., & Drechsler, R. (2015). Behavior driven development for tests and verification. In: Drechsler, R., Kühne, U. (Eds.), Formal Modeling and Verification of Cyber-Physical Systems (pp. 275-277). Springer Vieweg, Wiesbaden.

Do, H. (2016). Recent advances in regression testing techniques. Advances in computers, 103, 53-77.

Do, H., Elbaum, S., & Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering, 10(4), 405-435.

Dudin, M., Frolova, E., Gryzunova, N., & Shuvalova, E. (2015). The Deming cycle (PDCA) concept as an efficient tool for continuous quality improvement in the agribusiness. Asian Social Science, 11(1), 239-246.

Duraisamy, G., & Atan, R. (2013). Requirement traceability matrix through documentation for Scrum methodology. Journal of Theoretical & Applied Information Technology, 52(2), 154-159.

Durán, D. E. S., Gamboa, A. X. R., & Builes, J. A. J. (2017). Aplicación de la gestión de conocimiento al proceso de pruebas de software. Revista Ingenierías USBMed, 8(2), 6-13.

Elsner, D., Hauer, F., Pretschner, A., & Reimer, S. (2021, July). Empirically evaluating readily available information for regression test optimization in continuous integration. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 491-504).

Engström, E., Feldt, R., & Torkar, R. (2012, September). Indirect effects in evidential assessment: A case study on regression test technology adoption. In Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies (pp. 15-20). ACM.

Engström, E., & Runeson, P. (2010, June). A qualitative survey of regression testing practices. In International Conference on Product Focused Software Process Improvement (pp. 3-16). Berlin, Heidelberg: Springer Berlin Heidelberg.

Engström, E., Runeson, P., & Skoglund, M. (2010). A systematic review on regression test selection techniques. Information and Software Technology, 52(1), 14-30.

Erthal, V. M., de Souza, B. P., dos Santos, P. S. M., & Travassos, G. H. (2023). Characterization of continuous experimentation in software engineering: Expressions, models, and strategies. Science of Computer Programming, 229, 102961.

Espinosa-Curiel, I. E., Rodríguez-Jacobo, J., & Fernández-Zepeda, J. A. (2013). A framework for evaluation and control of the factors that influence the software process improvement in small organizations. Journal of software: Evolution and Process, 25(4), 393-406.

Esquirol-Caussa, J., Aldeguer, J. S., & Santamaria, I. D. (2017). La revisión bibliográfica: Base de la investigación. Actualizaciones en Fisioterapia, 2(12), 34-37.

Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A., & Oivo, M. (2018). Empirical software engineering experts on the use of students and professionals in experiments. Empirical Software Engineering, 23(1), 452-489.

Farley, D. (2021). Modern software engineering: Doing what works to build better software faster. Boston, MA: Addison-Wesley Professional.

Farooq, M. S., Omer, U., Ramzan, A., Rasheed, M. A., & Atal, Z. (2023). Behavior driven development: A systematic literature review. IEEE Access, 11, 88008-88024.

Faustino, I., & Mejía, J. (2020, October). Proposal for a software development framework based on the ISO/IEC 29110 standard: Public organizations. In 2020 9th International Conference On Software Process Improvement (CIMPS) (pp. 132-140). IEEE.

Felderer, M., & Fourneret, E. (2015). A systematic classification of security regression testing approaches. International Journal on Software Tools for Technology Transfer, 17, 305-319.

Fisal, N. R., Hamdy, A., & Rashed, E. A. (2021). Search-based regression testing optimization. International Journal of Open Source Software and Processes, 12(2), 1-20.

Fraser, G., & Arcuri, A. (2012). Whole test suite generation. IEEE Transactions on Software Engineering, 39(2), 276-291.

Galin, D. (2018). Software quality: Concepts and practice. Hoboken, NJ: John Wiley & Sons – IEEE Computer Society.

Gamido, H. V., & Gamido, M. V. (2019). Comparative review of the features of automated software testing tools. International Journal of Electrical and Computer Engineering, 9(5), 4473.

García, A. G., Hernández, M. S., Marcos, S. V., & Dapena, M. D. D. (2023). Buenas prácticas de la Ingeniería de Software: Pruebas de software. Revista Cubana de Transformación Digital, 4(2), 205-1.

Garousi, V., Felderer, M., Kuhrmann, M., Herkiloğlu, K., & Eldh, S. (2020). Exploring the industry's challenges in software testing: An empirical study. Journal of Software: Evolution and Process, 32(8), e2251.

Garousi, V., Rainer, A., Lauvås Jr, P., & Arcuri, A. (2020a). Software-testing education: A systematic literature mapping. Journal of Systems and Software, 165, 110570.

Gasca-Hurtado, G. P., Gómez-Álvarez, M. C., Machuca-Villegas, L., & Muñoz, M. (2021). Design of a gamification strategy to intervene in social and human factors associated with software process improvement change resistance. IET Software, 15(6), 428-442.

Ghazi, A. N., Petersen, K., Reddy, S. S. V. R., & Nekkanti, H. (2018). Survey research in software engineering: Problems and mitigation strategies. IEEE Access, 7, 24703-24718.

Govil, N., & Sharma, A. (2021, October). A game plan to build optimized regression testing in agile methodologies using test prioritization. In 2021 5th International Conference on Information Systems and Computer Networks (ISCON) (pp. 1-4). IEEE.

Greca, R., Miranda, B., & Bertolino, A. (2023). State of practical applicability of regression testing research: A live systematic literature review. ACM Computing Surveys, 55(13s), 1-36.

Guevara-Vega, C., Bernárdez, B., Durán, A., Quina-Mera, A., Cruz, M., & Ruiz-Cortés, A. (2021, March). Empirical strategies in software engineering research: A literature survey. In 2021 2nd International Conference on Information Systems and Software Technologies (ICI2ST) (pp. 120-127). IEEE Computer Society.

Gui, L., Leng, J., Pergola, G., Zhou, Y., Xu, R., & He, Y. (2019, November). Neural topic model with reinforcement learning. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 3478-3483). Association for Computational Linguistics.

Heller, D. (2020). Building a career in software: A comprehensive guide to success in the software industry. Boston, MA: APress.

Hooda, I., & Chhillar, R. S. (2015). Software test process, testing types and techniques. International Journal of Computer Applications, 111(13).

Horcas, J. M., Pinto, M., & Fuentes, L. (2019, September). Software product line engineering: A practical experience. In Proceedings of the 23rd International Systems and Software Product Line Conference – Volume A (pp. 164-176). ACM.

Hynninen, T., & Jantunen, S. (2022, May). Questionnaire approach for assessing software engineering and quality assurance practices. In 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO) (pp. 1301-1306). IEEE.

Ibidunni, A. S., Kolawole, A. I., Olokundun, M. A., & Ogbari, M. E. (2020). Knowledge transfer and innovation performance of small and medium enterprises (SMEs): An informal economy analysis. Heliyon, 6(8).

Ibitowa, F. O., & Akinola, S. O. (2021). Knowledge management in software testing. University of Ibadan Journal of Science and Logics in ICT Research, 7(2), 44-54.

International Organization for Standardization. (2015). ISO 9000:2015. Quality management systems. Fundamentals and vocabulary. Geneva, Switzerland.

International Organization for Standardization/International Electrotechnical Commission. (2021). ISO/IEC 5055:2021. Information technology. Software measurement. Software quality measurement. Geneva, Switzerland.

International Organization for Standardization/International Electrotechnical Commission. (2023). ISO/IEC 25010:2023. Systems and software engineering. Systems and software quality requirements and evaluation (SQuaRE). Product Quality Model. Geneva, Switzerland.

Institute of Electrical and Electronics Engineers. (2014). IEEE 730-2014. IEEE standard for software quality assurance processes. Los Alamitos, CA: IEEE Computer Society.

Institute of Electrical and Electronics Engineers. (2016). IEEE 1012-2016. IEEE standard for system, software, and hardware verification and validation. Los Alamitos, CA: IEEE Computer Society.

Itkonen, J., & Mäntylä, M. V. (2014). Are test cases needed? Replicated comparison between exploratory and test-case-based software testing. Empirical Software Engineering, 19, 303-342.

Jarzębowicz, A., & Weichbroth, P. (2021). A qualitative study on non-functional requirements in agile software development. IEEE Access, 9, 40458-40475.

Ji, T., Chen, L., Mao, X., Yi, X., & Jiang, J. (2022). Automated regression unit test generation for program merges. Science China Information Sciences, 65(9), 199103.

Jeon, T., & von Mayrhauser, A. (1994, December). A knowledge-based approach to regression testing. In Proceedings of 1st Asia-Pacific Software Engineering Conference (pp. 144-153). IEEE.

Jung, P., Kang, S., & Lee, J. (2020). Efficient regression testing of software product lines by reducing redundant test executions. Applied Sciences, 10(23), 8686.

Kandil, P., Moussa, S., & Badr, N. (2015, December). A methodology for regression testing reduction and prioritization of agile releases. In 2015 5th International Conference on Information & Communication Technology and Accessibility (ICTA) (pp. 1-6). IEEE.

Kandukuri, P. (2020). Regression testing: A model driven approach. Boston, MA: Lambert Academic Publishing.

Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software, 172, 110851.

Kazmi, R., Jawawi, D. N., Mohamad, R., & Ghani, I. (2017). Effective regression test case selection: A systematic literature review. ACM Computing Surveys (CSUR), 50(2), 1-32.

Khaleel, S. I., & Anan, R. (2023). A review paper: Optimal test cases for regression testing using artificial intelligent techniques. International Journal of Electrical and Computer Engineering, 13(2), 1803-1816.

Khan, S. U. R., Lee, S. P., Javaid, N., & Abdul, W. (2018). A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines. IEEE Access, 6, 11816-11841.

Khari, M., Sinha, A., Verdu, E., & Crespo, R. G. (2020). Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization. Soft Computing, 24(12), 9143-9160.

Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., & Tumeng, R. (2018). Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, 93, 74-93.

Kulkarni, V. (2021). Regression test optimization and automation in agile framework: A review. Turkish Journal of Computer and Mathematics Education, 12(12), 2852-2856.

Kumar, S. (2023). Reviewing software testing models and optimization techniques: An analysis of efficiency and advancement needs. Journal of Computers, Mechanical and Management, 2(1), 32-46.

Labuschagne, A., Inozemtseva, L., & Holmes, R. (2017, August). Measuring the cost of regression testing in practice: A study of Java projects using continuous integration. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 821-830). ACM.

Levi, A., & Sarimurat, S. (2017). Utilizing hash graphs for key distribution for mobile and replaceable interconnected sensors in the IoT context. Ad Hoc Networks, 57, 3-18.

Lin, G., Wen, S., Han, Q. L., Zhang, J., & Xiang, Y. (2020). Software vulnerability detection using deep neural networks: A survey. Proceedings of the IEEE, 108(10), 1825-1848.

Liu, K., Wang, S., Koyuncu, A., Kim, K., Bissyandé, T. F., Kim, D., Wu, P., Klein, J., Mao, X., & Traon, Y. L. (2020, June). On the efficiency of test suite based program repair: A systematic assessment of 16 automated repair systems for java programs. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 615-627). IEEE.

Liu, Y., Wu, J., Liu, X., & Gu, G. (2009, July). Investigation of knowledge management methods in software testing process. In 2009 International Conference on Information Technology and Computer Science (Vol. 2, pp. 90-94). IEEE.

Ma, W., Papadakis, M., Tsakmalis, A., Cordy, M., & Traon, Y. L. (2021). Test selection for deep learning systems. ACM Transactions on Software Engineering and Methodology (TOSEM), 30(2), 1-22.

Ma, X. X., & Wang, J. S. (2018). Optimized parameter settings of binary bat algorithm for solving function optimization problems. Journal of Electrical and Computer Engineering, 22(2), 120-129.

Maciel, C. P. C., de Souza, É. F., Vijaykumar, N. L., de Almeida Falbo, R., Meinerz, G. V., & Felizardo, K. R. (2018, April). An empirical study on the knowledge management practice in software

testing. In XXI Ibero-American Conference on Software Engineering (pp. 29-42). Universidad de los Andes.

Madeyski, L., & Kawalerowicz, M. (2016, August). Software engineering needs agile experimentation: A new practice and supporting tool. In Results of the XVIII KKIO 2016 Software Engineering Conference (pp. 149-162). Cham: Springer International Publishing.

Marenbach, R., & Albert, M. (2018). Regression test approach for testing of protection IEDs to improve field testing quality and support knowledge management. The Journal of Engineering, 2018(15), 1023-1026.

Marin Diaz, A., Trujillo Casañola, Y., & Buedo Hidalgo, D. (2020). Estrategia de pruebas para organizaciones desarrolladoras de software. Revista Cubana de Ciencias Informáticas, 14(3), 83-104.

Marijan, D. (2023). Comparative study of machine learning test case prioritization for continuous integration testing. Software Quality Journal, 31(4), 1415-1438.

Martínez-Ruiz, T., García, F., & Piattini, M. (2009, May). Process institutionalization using software process lines. In International Conference on Enterprise Information Systems (359-362). SCITEPRESS.

Martinsuo, M., & Huemann, M. (2021). Designing case study research. International Journal of Project Management, 39(5), 417-421.

Maulud, D., & Abdulazeez, A. M. (2020). A review on linear regression comprehensive in machine learning. Journal of Applied Science and Technology Trends, 1(2), 140-147.

Maxim, B. R., & Kessentini, M. (2016). An introduction to modern software quality assurance. In: Mistrik, I., Soley, R., Ali, N., Grundy, J., Tekinerdogan, B. (Eds.), Software Quality Assurance (pp. 19-46). London, UK: Morgan Kaufmann.

Medina-Domínguez, F., Saldaña-Ramos, J., Mora-Soto, A., Sanz-Esteban, A., & Sanchez-Segura, M. I. (2008, July). A collaborative framework to support software process improvement based on the reuse of process assets. In International Conference on Software and Data Technologies (Vol. 2, pp. 283-289). SCITEPRESS.

Mejía, J., Arroyo-Morales, L. A., & Tablada-Domínguez, A. (2022, October). Improvement findings in the implementation of software tests based on the ISO/IEC 29110 standard: Case study. In 2022 11th International Conference On Software Process Improvement (CIMPS) (pp. 202-211). IEEE.

Mejía, J., Rodríguez-Maldonado, I., Girón-Bobadilla, H., Muñoz, M. (2019, June). Knowledge management in software process improvement: a systematic literature review. In 2019 14th Iberian Conference on Information Systems and Technologies (CISTI) (pp. 1-7). IEEE.

Mendonça, W. D., Assunção, W. K., & Vergilio, S. R. (2022). Software product line regression testing: A research roadmap. In Proceedings of the 24th International Conference on Enterprise Information Systems (pp. 81-89). SCITEPRESS.

Meyer, M. L. B., Waeselynck, H., & Cuesta, F. (2023, October). A case study on the "jungle" search for industry-relevant regression testing. In 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS) (pp. 382-393). IEEE.

Micheli, J., & Oliver, R. (2017). Empresas de software en México y sus vínculos de desarrollo local. Problemas del desarrollo, 48(190), 37-59.

Middelveen, T. F. (1998). A knowledge based tool for regression testing. Thesis of the Master of Engineering Program. Royal Military College of Canada.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). YALE: Rapid prototyping for complex data mining tasks. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 935-940). ACM.

Minhas, N. M., Petersen, K., Börstler, J., & Wnuk, K. (2020). Regression testing for large-scale embedded software development–Exploring the state of practice. Information and software technology, 120, 106254.

Minhas, N. M. (2022). Understanding and improving regression testing practice. Doctoral dissertation. Blekinge Institute of Technology.

Minhas, N. M., Börstler, J., & Petersen, K. (2023). Checklists to support decision-making in regression testing. Journal of Systems and Software, 202, 111697.

Mishra, A. (2017). IOS code testing: Test-driven development and behavior-driven development with Swift. New York, NY, USA: APress.

Morales-Aguiar, N., & Vega-Zepeda, V. (2018). Factores humanos y la mejora de procesos de software. Propuesta inicial de un catálogo que guíe su gestión. Revista Ibérica de Sistemas y Tecnologías de la Información, (29), 30-42.

Morrison, P., Smith, B. H., & Williams, L. (2017, April). Surveying security practice adherence in software development. In Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp (pp. 85-94).

Moyano-Hernández, F. A., & Sandoval, D. C. V. (2021). Análisis del ciclo PHVA en la gestión de proyectos, una revisión documental. Revista Politécnica, 17(34), 55-69.

Mukherjee, R., & Patnaik, K. S. (2021). A survey on different approaches for software test case prioritization. Journal of King Saud University-Computer and Information Sciences, 33(9), 1041-1054.

Myers, G. J., Badgett, T., & Sandler, C. (2023). The art of software testing. Boston, MA: Wiley-Blackwell.

Nonaka, I., & Takeuchi, H. (2009). The knowledge-creating company. In Siesfeld, T., Cefola, J., Neef, D. (Eds.), The Economic Impact of Knowledge (pp. 175-187). New York, NY: Routledge.

Ndukwe, I. G., Licorish, S. A., Tahir, A., & MacDonell, S. G. (2023). How have views on software quality differed over time? Research and practice viewpoints. Journal of Systems and Software, 195, 111524.

Ocaña-Fernández, Y., & Fuster-Guillén, D. (2021). La revisión bibliográfica como metodología de investigación. Revista Tempos y Espacios en Educación, 14(33), e15614.

Oliveira. Jr, E., Furtado, V., Vignando, H., Luz, C., Cordeiro, A., Steinmacher, I., & Zorzo, A. (2021, September). Towards improving experimentation in software engineering. In Proceedings of the XXXV Brazilian Symposium on Software Engineering (pp. 335-340). ACM.

Palomo, S. R. G., & Gil, E. M. (2020). Aproximación a la Ingeniería del Software. España, Madrid: Editorial Centro de Estudios Ramon Areces S.A.

Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: a systematic literature review. Empirical Software Engineering, 27(2), 29.

Panda, N., Acharya, A. A., & Mohapatra, D. P. (2020). Regression testing of object-oriented systems using UML state machine diagram and sequence diagram. International Journal of Computing Science and Mathematics, 12(2), 132-146.

Pardo, A. C. (2021). Un mecanismo de administración de información histórica de proyectos de software que facilite la estimación de los desarrollos. Tesis de Magister en Tecnologías de la Información. Facultad de Ciencias Físicas y Matemáticas. Departamento de Ciencias de la Computación. Universidad de Chile.

Parsons, D., Susnjak, T., & Lange, M. (2014). Influences on regression testing strategies in agile software development environments. Software Quality Journal, 22, 717-739.

Pérez, J., Arenas, M., & Gutierrez, C. (2009). Semantics and complexity of SPARQL. ACM Transactions on Database Systems (TODS), 34(3), 1-45.

Pesado, P. M., Bertone, R. A., Esponda, S., Pasini, A. C., Boracchia, M., Martorelli, S. L., & Swaels, M. (2013, June). Mejora de procesos en el desarrollo de sistemas de software y en procesos de gestión. In XV Workshop de Investigadores en Ciencias de la Computación (pp. 581-585). Red de Universidades con Carreras en Informática.

Pintelas, E., Livieris, I. E., & Pintelas, P. (2020). Un modelo de conjunto de caja gris que aprovecha la precisión de la caja negra y la interpretabilidad intrínseca de la caja blanca. Algoritmos, 13(1), 17.

Ponzio, P., Aguirre, N., Frias, M. F., & Visser, W. (2016, November). Field-exhaustive testing. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 908-919). ACM.

Porter, M. E. (1997). Competitive strategy. Measuring Business Excellence, 1(2), 12-17.

Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A practitioner's approach. New York, NY: McGraw Hill.

Qasim, M., Bibi, A., Hussain, S. J., Jhanjhi, N. Z., Humayun, M., & Sama, N. U. (2021). Test case prioritization techniques in software regression testing: An overview. International Journal of Advanced and Applied Sciences, 8(5), 107-121.

Qiu, D., Li, B., Ji, S., & Leung, H. (2014). Regression testing of web service: A systematic mapping study. ACM Computing Surveys (CSUR), 47(2), 1-46.

Ragkhitwetsagul, C., Krinke, J., Choetkiertikul, M., Sunetnanta, T., & Sarro, F. (2022, March). Identifying software engineering challenges in software SMEs: A case study in Thailand. In 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (pp. 218-222). IEEE.

Ramachandran, M. (Ed.). (2011). Knowledge engineering for software development life cycles: Support technologies and applications. IGI Global.

Rehan, M., Senan, N., Aamir, M., Samad, A., Husnain, M., Ibrahim, N., Ali, S., & Khatak, H. (2021). A systematic analysis of regression test case selection: A multi-criteria-based approach. Security and Communication Networks, 12(3), 1-11.

Rempel, P., & Mäder, P. (2016). Preventing defects: The impact of requirements traceability completeness on software quality. IEEE Transactions on Software Engineering, 43(8), 777-797.

Riehle, D., & Züllighoven, H. (1996). Understanding and using patterns in software development. Tapos, 2(1), 3-13.

Rhodes, L., & Dawson, R. (2013). Lessons learned from lessons learned. Knowledge and process management, 20(3), 154-160.

Rojas-Montes, M. L., Pino-Correa, F. J., & Martínez, J. M. (2015). Proceso de pruebas para pequeñas organizaciones desarrolladoras de software. Revista Facultad de Ingeniería, 24(39), 55-70.

Romano, S., Scanniello, G., Antoniol, G., & Marchetto, A. (2018). Spiritus: A simple information retrieval regression test selection approach. Information and Software Technology, 99, 62-80.

Rosenblum, D. S., & Weyuker, E. J. (1997). Using coverage information to predict the cost-effectiveness of regression testing strategies. IEEE Transactions on Software Engineering, 23(3), 146-156.

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 years of software regression testing techniques—A survey. International Journal of Software Engineering and Knowledge Engineering, 26(05), 675-689.

Rosero, R. H., Gómez, O. S., Villa, E. R., Aguilar, R. A., & Pardo, C. J. (2022). Software regression testing in industrial settings: Preliminary findings from a literature review. In The International Conference on Advances in Emerging Trends and Technologies (pp. 227-237). Springer, Cham.

Runeson, P., Ohlsson, M. C., & Wohlin, C. (2001). A classification scheme for studies on fault-prone components. In Product Focused Software Process Improvement: Third International Conference, PROFES 2001 Kaiserslautern, Germany, September 10–13, 2001 Proceedings 3 (pp. 341-355). Springer Berlin Heidelberg.

Saifan, A. A., Alsukhni, E., Alawneh, H., & Sbaih, A. A. (2016). Test case reduction using data mining technique. International Journal of Software Innovation (IJSI), 4(4), 56-70.

Salahat, M., Said, R. A., Hamid, K., Haseeb, U., Ghani, E. A. M. A., Abualkishik, A., Iqbal, M. W., & Inairat, M. (2023, March). Software testing issues improvement in quality assurance. In 2023 International Conference on Business Analytics for Technology and Security (ICBATS) (pp. 1-6). IEEE.

Salgado, P. A. (2020). Mejora de la estrategia de testing en una empresa de desarrollo de software, alineado con algunas áreas de CMMI, usando la metodología DMAIC. Tesis de Maestría en Gestión y Desarrollo de Proyectos de Software. Universidad Autónoma de Manizales.

Samad, A., Mahdin, H., Kazmi, R., & Ibrahim, R. (2021). Regression test case prioritization: A systematic literature review. International Journal of Advanced Computer Science and Applications, 12(2), 435-442.

Sharif, A., Marijan, D., & Liaaen, M. (2021, September). Deeporder: Deep learning for test case prioritization in continuous integration testing. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 525-534). IEEE.

Sharma, S., & Kumar, D. (2022). Towards a shift in regression test suite development approach in agile. Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science), 15(5), 668-675.

Shi, A., Zhao, P., & Marinov, D. (2019, October). Understanding and improving regression test selection in continuous integration. In 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE) (pp. 228-238). IEEE.

Shikta, S., Shahriyar, H. M., Das, S. K., Mahal, S. N., Al Jannat, K. B., & Alam, S. (2021, June). Quality assurance guidelines for successful startups. In 2021 IEEE/ACIS 19th International Conference on Software Engineering Research, Management and Applications (SERA) (pp. 81-85). IEEE.

Shin, S. Y., Nejati, S., Sabetzadeh, M., Briand, L. C., & Zimmer, F. (2018, July). Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based

approach. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 49-60). ACM.

Sidhu, A. K., & Sehra, S. K. (2022). Use of software metrics to improve the quality of software projects using regression testing. In: Khosrow-Pour, M., Clarke, S., Jennex, M. E., Anttiroiko, A. V. (Eds.), Research Anthology on Agile Software, Software Development, and Testing (pp. 399-411). Boston, MA: IGI Global.

Sillito, J., & Kutomi, E. (2020, September). Failures and fixes: A study of software system incident response. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 185-195). IEEE.

Singh, A., Singh, K., & Sharma, N. (2015). Agile in global software engineering: An exploratory experience. International Journal of Agile Systems and Management, 8(1), 23-38.

Sivaji, U., & Rao, P. S. (2021). Improving regression testing query replying procedure using secure optimized graph walk scheme. Journal of Theoretical and Applied Information Technology, 99.

Singhal, S., Jatana, N., Suri, B., Misra, S., & Fernandez-Sanz, L. (2021). Systematic literature review on test case selection and prioritization: A tertiary study. Applied Sciences, 11(24), 12121.

Smart, J. F., & Molak, J. (2023). BDD in action: Behavior-driven development for the whole software lifecycle. Shelter Island, NY: Manning Publications Co.

Sommerville, I. (2021). Engineering software products: An introduction to modern software engineering. New York, NY: Pearson.

Song, X., Lin, Y., Ng, S. H., Wu, Y., Peng, X., Dong, J. S., & Mei, H. (2022, July). RegMiner: Towards constructing a large regression dataset from code evolution history. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 314-326). ACM.

Soto, D. E., Reyes, A. X., & Jiménez, J. (2017). Application of knowledge management to the software testing process. Revista ESPACIOS, 38(43).

de Souza, É. F., de Almeida Falbo, R., Vijaykumar, N. L., Felizardo, K. R., Meinerz, G. V., Specimille, M. S., & Coelho, A. G. (2021). Development of an ontology-based approach for knowledge management in software testing: An experience report. Journal of Software Engineering Research and Development, 9, 12-1.

Srikanth, M., & Murugan, P. (2020). A comparative study of regression testing techniques for software maintenance. In 2020 11th International Conference on Computing, Communication and Networking Technologies (pp. 1-6). IEEE.

Sutar, S., Kumar, R., Pai, S., & Shwetha, B. R. (2020, June). Regression test cases selection using natural language processing. In 2020 International Conference on Intelligent Engineering and Management (ICIEM) (pp. 301-305). IEEE.

Taipale, O., Karhu, K., & Smolander, K. (2007, September). Observing software testing practice from the viewpoint of organizations and knowledge management. In First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007) (pp. 21-30). IEEE.

Thota, M. K., Shajin, F. H., & Rajesh, P. (2020). Survey on software defect prediction techniques. International Journal of Applied Science and Engineering, 17(4), 331-344.

Toapanta, H. M. C., Sinchiguano, B. E. O., & Jimenez, E. M. O. (2017). Mejora de procesos de software. Revista publicando, 4(12 (1)), 68-88.

Tuape, M., & Ayalew, Y. (2019, May). Factors affecting development process in small software companies. In 2019 IEEE/ACM Symposium on Software Engineering in Africa (pp. 16-23). IEEE.

Tuape, M., Hasheela-Mufeti, V. T., Kayanda, A., Porras, J., & Kasurinen, J. (2021). Software Engineering in small software companies: Consolidating and integrating empirical literature into a process tool adoption framework. IEEE Access, 9, 130366-130388.

Tuape, M., Hasheela-Mufeti, V. T., & Kasurinen, J. (2022). Theory on non-technical characteristics Affecting process adoption in small software companies: A grounded theory study. IEEE Access, 10, 103382-103400.

Van Der Aalst, W. (2012). Process mining: Overview and opportunities. ACM Transactions on Management Information Systems (TMIS), 3(2), 1-17.

Vos, T. E., Marín, B., Escalona, M. J., & Marchetto, A. (2012, August). A methodological framework for evaluating software testing techniques and tools. In 2012 12th International Conference on Quality Software (pp. 230-239). IEEE.

Wang, S., Lian, X., Marinov, D., & Xu, T. (2023, May). Test selection for unified regression testing. In 2023 IEEE/ACM 45th International Conference on Software Engineering (pp. 1687-1699). IEEE.

Wang, Y., Liu, J., Chang, X., Mišić, J., & Mišić, V. B. (2022). IWA: Integrated gradient‐based white‐box attacks for fooling deep neural networks. International Journal of Intelligent Systems, 37(7), 4253-4276.

Wellman, J. (2007). Lessons learned about lessons learned. Organization Development Journal, 25(3), 65.

Wieringa, R., & Daneva, M. (2015). Six strategies for generalizing software engineering theories. Science of Computer Programming, 101, 136-152.

Wiklund, K., Eldh, S., Sundmark, D., & Lundqvist, K. (2017). Impediments for software test automation: A systematic literature review. Software Testing, Verification and Reliability, 27(8), e1639.

Withanawasam, J. (2015). Apache mahout essentials. New York, NY: Packt Publishing Ltd.

Wnuk, K., & Garrepalli, T. (2018). Knowledge management in software testing: A systematic snowball literature review. e-Informatica Software Engineering Journal, 12(1), 51-78.

Wohlin, C. (2021). Case study research in Software Engineering—It is a case, and it is a study, but is it a case study? Information and Software Technology, 133, 106514.

Wohlin, C., & Rainer, A. (2022). Is it a case study?—A critical analysis and guidance. Journal of Systems and Software, 192, 111395.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2024). Experimentation in Software Engineering (2024 edition), Springer.

Wong, W. E., Horgan, J. R., London, S., & Agrawal, H. (1997, November). A study of effective regression testing in practice. In Proceedings of the 8th International Symposium On Software Reliability Engineering (pp. 264-274). IEEE.

Yao, J., Crupi, A., Di Minin, A., & Zhang, X. (2020). Knowledge sharing and technological innovation capabilities of Chinese software SMEs. Journal of Knowledge Management, 24(3), 607-634.

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability, 22(2), 67-120.

Zarrad, A. (2015). A systematic review on regression testing for web-based applications. Journal of Software, 10(8), 971-990.

Zhong, H., Zhang, L., & Khurshid, S. (2019, April). Testsage: Regression test selection for large-scale web service testing. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) (pp. 430-440). IEEE.

Zúñiga, P. I. V., Cedeño, R. J. C., & Palacios, I. A. M. (2023). Metodología de la investigación científica: Guía práctica. Ciencia Latina Revista Científica Multidisciplinar, 7(4), 9723-9762.

7. Anexo A.- Acrónimos

ABBA	Algoritmo Binario de Murciélago Adaptado Multiobjetivo
AGC	Modelo Adicional de Llamada Codiciosa
AL	Hormiga León
ANFIS	Sistema de Inferencia Neuro Difuso Adaptativo
ANN	Redes Neuronales Artificiales
APFD	Porcentaje Promedio de Detección de Fallos (o Porcentaje Promedio de Fallas Detectadas)
AUCA	Área bajo la Curva del Diagrama de Alberg
BDD	Desarrollo Guiado en el Comportamiento
BPM	Gestión de Procesos de Negocio
BPSO	Optimización del Enjambre de Partículas Binarias
CBM	Modelo Basado en la Cobertura
CCP	Cobertura de los Casos de Prueba
CL	Lista de Comprobación
CMMI	Modelo de Madurez y Capacidad Integradas (versión 1.1, versión 3.0)
CRF	Cobertura de los Requisitos Funcionales
CRM	Sistemas de Gestión de Relaciones con los Clientes
DA	De Acuerdo
DOCP	Activo (Documentación del Caso de Prueba)
DOCPR	Activo (Descripción del Caso de Prueba para la Regresión)
ED	En Desacuerdo
ERP	Sistemas de Planificación de Recursos
ESPRU	Activo (Definición del Escenario de Prueba)
EWMA	Media Móvil Ponderada Exponencialmente
FDC	Pérdida de Capacidad de Detección de Fallos
FR	Tasa de Fallos
GLC	Diferencia entre la Ultima Versión Ejecutada y la Actual

GRASP	Procedimiento Adaptativo y Codicioso de Búsqueda
HP	Perfil Horizontal
IA	Inteligencia Artificial
IEEE	Instituto de Ingenieros Eléctricos y Electrónicos
KM	Gestión del Conocimiento
KMS	Sistema de Gestión del Conocimiento
LDA	Asignación Latente de Dirichlet
LECCA	Activo (Documentación de las Lecciones Aprendidas)
MATRAZ	Activo (Matriz de Trazabilidad)
MD	Distancia de Mahalanobis
N	Neutral
NFS	Sistema Neuro Difuso
NLP	Procesamiento del Lenguaje Natural
OBBA	Algoritmo Binario Original de Murciélago
OHGW-ALR	Réplica Optimizada del Recorrido de Hormiga León en Grafos Hash
OOP	Programación Orientada a Objetos
PAL	Repositorio o Biblioteca de Activos del Proceso
PAL-RT	Biblioteca de Activos del Proceso de Pruebas de Regresión
PDF	Porcentaje de Detección de Fallos
PPRUR	Activo (Plan de Pruebas de Regresión)
REFAL	Activo (Registro de Fallos Detectados en los Casos de Prueba/Reporte de Fallos)
REGI	Activo (Registro de Incidencias)
RQ	Preguntas de Investigación
SCF	Factores Críticos de Éxito
SECIHTI	Secretaría de Ciencia, Humanidades, Tecnología e Innovación
SHF	Factores Sociales y Humanos
SOLCA	Activo (Solicitud de Cambios)
SPI	Mejora de los Procesos de Software
SPIRITuS	Enfoque de Selección de Pruebas de Regresión basado en la Recuperación de Información Simple
SQA	Aseguramiento de la Calidad del Software
SRT	Pruebas de Regresión con un Alcance Específico
TDA	Totalmente de Acuerdo
TCP	Priorización de los Casos de Prueba
TCS	Selección de los Casos de Prueba
TED	Totalmente en Desacuerdo
TDD	Desarrollo Guiado por las Pruebas
TFCP	Tasa de Fallos en los Casos de Prueba

TSR	Porcentaje de Reducción del Conjunto de Pruebas
UML	Lenguaje Unificado de Modelado