PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Escuela de Posgrado

Revisión sistemática de métodos de pruebas de software para aseguramiento de la calidad

Trabajo de Investigación para obtener el grado académico de Maestro en Informática con mención en Ingeniería de Software que presenta:

Rolando Yenner Diaz Gamboa

Asesor:

Mg. Eder Ramiro Quispe Vilchez

Lima, 2024

Informe de Similitud

Yo, Eder Ramiro QUISPE VILCHEZ, docente de la Escuela de Posgrado de la Pontificia Universidad Católica del Perú, asesor(a) de el trabajo de investigación titulada(o) Revisión sistemática de métodos de pruebas de software para aseguramiento de la calidad, de el autor Rolando Yenner DÍAZ GAMBOA, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 37%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 21/03/2024.
- He revisado con detalle dicho reporte y la Tesis de investigación, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:

San Miguel, 21 de Marzo de 2024.

Apellidos y nombres del asesor / de la asesora: QUISPE VILCHEZ, EDER RAMIRO	
DNI: 42264307	Firma
ORCID: 0000-0003-1639-5134	

Resumen

El objetivo de un software es satisfacer la necesidad del usuario final, por ello es indispensable que el proceso de desarrollo de software cuente con actividades que permitan el aseguramiento de la calidad del software. Las pruebas de software ayudan en la prevención de errores en un sistema. Además, se utiliza para analizar el software en busca de otros aspectos del software, como usabilidad, compatibilidad, confiabilidad, integridad, eficiencia, seguridad, capacidad, portabilidad, mantenibilidad y otros.

Se requieren pruebas de software para verificar y validar que el software ha sido construido de acuerdo con sus especificaciones. Por consiguiente, el propósito de esta revisión sistemática es identificar investigaciones en entornos académicos y/o industriales que describan los métodos de pruebas de desarrollo de software y su impacto en el aseguramiento de la calidad, sin importar la metodología o marco de desarrollo de software empleado.

Se han identificado 55 artículos relevantes que permiten concluir que los métodos de pruebas de software se aplican en diversas etapas del ciclo de vida del desarrollo de software, contribuyendo al aseguramiento de la calidad mediante procesos de validación y verificación en diversos escenarios y condiciones. Además, se ha observado que los métodos de pruebas son determinados por la metodología de desarrollo de software utilizada, y que la exhaustividad de las pruebas de software está influenciada por el contexto de la aplicación. En relación con la calidad del software, se destaca la influencia de dos procesos clave: el Aseguramiento de la Calidad, que proporciona un plan y controles para prevenir defectos, y el Control de la Calidad, que se enfoca en la detección y corrección de errores mediante el uso de métodos de pruebas de software. Ambos procesos aseguran que el software desarrollado cumpla con los estándares de calidad, satisfaciendo así las necesidades y expectativas de los usuarios finales.

Palabras Clave: Calidad, Software, Aseguramiento de calidad (QA), Métodos de Pruebas, Pruebas de Software

Abstract

The objective of software is to satisfy the needs of end users; therefore, it is crucial for the software development process to include activities that guarantee software quality. Software testing is essential in preventing system errors. It is also used to assess software for various aspects, such as usability, compatibility, reliability, integrity, efficiency, security, capability, portability, maintainability, and more.

Software testing is required to ensure and validate that the software has been developed in accordance with its specifications. Thus, the purpose of this systematic review is to identify studies within academic and/or industrial contexts that describe software development testing methods and their impact on quality assurance, regardless of the software development methodology or framework utilized.

Fifty-five relevant articles have been identified, concluding that software testing methods are applied at various stages of the software development lifecycle, enhancing quality assurance through validation and verification processes across different scenarios and conditions.

Additionally, it has been observed that the testing methods are influenced by the software development methodology employed, and the thoroughness of software testing is affected by the application context. In terms of software quality, the significant roles of two key processes are highlighted: Quality Assurance, which provides a plan and controls to prevent defects, and Quality Control, which focuses on detecting and correcting errors through software testing methods. Both processes ensure that the developed software meets quality standards, thereby fulfilling the needs and expectations of end users.

Keywords: Quality, Software, Quality Assurance (QA), Testing Methods, Software Testing.

Índice de Contenido

Índice de Tablas

Índice de Figuras

CAPITULO I: Introducción

Las pruebas de software poseen métodos y técnicas para verificar y validar la calidad del software [1]. La prueba de software es el procedimiento de ejecutar un programa o sistema con la intención de encontrar fallas [2]. Se mide como intensivo en mano de obra y costoso, lo que representa > 50 % del costo total de desarrollo de software [3]. Las pruebas de software son una actividad importante del ciclo de vida de desarrollo de software (SDLC). Ayuda a desarrollar la confianza de un desarrollador de que un programa hace lo que está destinado a hacer.

Se necesitan pruebas de software para verificar y validar que el software que ha sido construido para cumplir con estas especificaciones. Las pruebas de software ayudan en la prevención de errores en un sistema. También se utiliza para analizar el software en busca de otros aspectos del software, como usabilidad, compatibilidad, confiabilidad, integridad, eficiencia, seguridad, capacidad, portabilidad, mantenibilidad, etc. [4]. En palabras simples, la prueba de software es el proceso de localizar errores en el programa. La prueba de software consiste en ejecutar el software para (i) realizar la verificación, (ii) detectar los errores y (iii) lograr la validación [5].

La investigación sobre la calidad del software es tan antigua como la construcción del software y la preocupación por la calidad surge con el diseño de programas libres de errores, así como la eficiencia cuando se utiliza [6]

De acuerdo con el Glosario estándar de terminología de ingeniería de software IEEE [7], la calidad de los productos de software se define como i) el grado en que un sistema, componente o proceso cumple con los requisitos especificados y ii) el grado en que un sistema, componente o proceso satisface las necesidades o expectativas de un usuario.

El presente trabajo busca identificar, en la literatura académica, los trabajos de investigación que describen los métodos utilizados durante las pruebas de software para el aseguramiento de la calidad en los ámbitos académicos y/o industriales independientemente de la metodología de gestión utilizada.

La estructura del trabajo será la siguiente: la sección 2 presenta el marco teórico; la sección 3 desarrolla la revisión sistemática de la literatura; la sección 4 lista las conclusiones y recomendaciones a las que se llegaron; y la sección 5 muestra las referencias usadas en la investigación.

CAPÍTULO II: Marco Teórico

2.1 Pruebas de Software

Las pruebas de software en el ámbito de la ingeniería de software son un componente crucial para garantizar la calidad y fiabilidad de los sistemas informáticos. Estas pruebas son fundamentales en el proceso de desarrollo de software, ya que permiten identificar y corregir errores, verificar si el software cumple con los requisitos especificados y garantizar su funcionamiento adecuado [8].

Se ha destacado la escasez de conocimientos y prácticas formales en pruebas de software, tanto en la industria como en el ámbito académico, lo que subraya la necesidad de incluir la formación en pruebas de software en los programas de ingeniería de software [9]. Asimismo, se ha señalado que las pruebas de software son esenciales para el desarrollo de productos de software complejos, y que su aplicación adecuada contribuye a reducir costos, tiempos y esfuerzos en el desarrollo y mantenimiento de software [10].

2.2 Métodos de Pruebas de Software

Los métodos de prueba de software en la ingeniería de software abarcan una variedad de enfoques y técnicas que se aplican a lo largo del ciclo de vida del desarrollo de software. El análisis estático implica examinar el código sin ejecutarlo, mientras que el análisis dinámico implica evaluar el código durante la ejecución. La combinación de estos métodos proporciona información completa sobre la calidad y seguridad del software [11]. La importancia de estos métodos radica en su capacidad para identificar y corregir errores, verificar el cumplimiento de los requisitos del software y garantizar su funcionamiento adecuado. Además, la aplicación de métodos de prueba adecuados contribuye a reducir costos, tiempos y esfuerzos en el desarrollo y mantenimiento de software.

Las pruebas de caja negra y las pruebas de caja blanca son dos enfoques fundamentales en las pruebas de software. Las pruebas de caja negra se centran en evaluar la funcionalidad de un sistema de software sin examinar su estructura de código interno. Este método incluye varios tipos específicos, como pruebas basadas en modelos, pruebas basadas en escenarios, pruebas basadas en datos, pruebas estadísticas y pruebas aleatorias [12]. Es particularmente útil para probar la funcionalidad del software bajo prueba sin considerar los detalles de su implementación interna o la estructura del código.

Por otro lado, las pruebas de caja blanca, también conocidas como pruebas de caja de vidrio, examinan la estructura interna, el diseño y la implementación del software. Este enfoque se utiliza principalmente para medir la idoneidad de los conjuntos de pruebas obtenidos mediante métodos de caja negra [12]. Las técnicas de prueba de caja blanca incluyen técnicas basadas en gráficos de llamadas y se clasifican como un enfoque de caja blanca, así como fuzzing de caja blanca basado en gramática inspirado en la ejecución simbólica y la generación de pruebas dinámicas [13] [14].

La combinación de estos dos métodos de prueba es esencial para lograr una cobertura de prueba integral. Las pruebas de caja negra son efectivas para validar el comportamiento externo del software, mientras que las pruebas de caja blanca son valiosas para identificar errores en el código y evaluar la lógica interna del programa.

Ambos enfoques son cruciales para garantizar la calidad, confiabilidad y seguridad de los sistemas de software.

La metodología de desarrollo de software también influye en los métodos de prueba utilizados. Los métodos ágiles, como XP, han ganado popularidad y han incorporado pruebas de software como parte integral de su enfoque [15]. Asimismo, se ha destacado la importancia de la secuencia de actividades a seguir para completar el ciclo de vida de desarrollo de un software, lo que resalta la relevancia de seguir metodologías específicas en el proceso de pruebas [16].

2.3 Validación y verificación de Software

Las pruebas de validación y verificación son el proceso de asegurar el resultado de cada fase en el ciclo de vida de desarrollo de software que cumple con los requisitos y especificaciones del usuario. Los procesos de validación y verificación de software son procesos importantes en el ciclo de vida del desarrollo de software junto con tipos de pruebas que garantizan la calidad de los productos [17].

2.4 Aseguramiento de la calidad del software

Para sofocar un malentendido generalizado, la garantía de calidad del software no es una prueba. El aseguramiento de la calidad del software (SQA) es un conjunto de actividades que definen y evalúan la adecuación de los procesos de software para proporcionar evidencia que establezca la confianza de que los procesos de software son apropiados y producen productos de software de calidad adecuada para los propósitos previstos [17].

CAPÍTULO III: Revisión sistemática de la literatura

3.1 Proceso de revisión

Este estudio se apoya en la metodología señalada por B. Kitchenham [18] para la realización de revisiones sistemáticas. Una revisión sistemática es un proceso formal y repetible para identificar, evaluar e interpretar toda la investigación disponible relacionada con una pregunta de investigación.

El proceso de realización de revisión sistemática comprende las siguientes fases y etapas:

a) Planificar la revisión

- Identificación de la necesidad de una revisión
- Encargo de una revisión (opcional)
- Especificación de la(s) pregunta(s) de investigación
- Desarrollo de un protocolo de revisión
- Evaluación del protocolo de revisión (opcional)

b) Realizar la revisión

- Identificación de la investigación
- Selección de estudios primarios
- Evaluación de la calidad de los estudios
- Extracción de datos y monitoreo
- Síntesis de los datos

c) Reportar la revisión

- Especificación de los mecanismos de difusión
- Formateo del reporte principal
- Evaluación del reporte (opcional)

Es importante tomar en cuenta que estas etapas no son completamente secuenciales y que muchas de ellas implican un proceso iterativo de progresiva revisión y refinamiento.

El protocolo de revisión debe especificar el contexto de la revisión, las preguntas de investigación, la estrategia de búsqueda, los criterios de selección de los estudios, los procedimientos para la selección de los estudios, los procedimientos para la evaluación de la calidad de los estudios, la estrategia de extracción de datos y de la síntesis de los datos extraídos, la estrategia de difusión y el cronograma del proyecto de revisión.

Para la estructuración de las preguntas de investigación se recomienda usar los criterios PICOC (*Population, Intervention, Comparison, Outcome, Context*) propuestos por [19].

La estrategia de búsqueda comprende la selección de fuentes y la definición de los términos de búsqueda. En esta etapa se debe elegir las fuentes o bases de datos más relevantes y que contengan los estudios más importantes y de mayor impacto sobre el tema de investigación.

Luego de una búsqueda inicial en las fuentes seleccionadas se puede realizar una verificación y análisis de las referencias contenidas en los estudios seleccionados para validar que no se haya dejado de lado estudios relevantes.

Los criterios de selección tienen como objetivo identificar todos los estudios primarios que contienen evidencia directa acerca de la pregunta de investigación, así como reducir el riesgo de parcialidad en la revisión. Deben estar basados en la pregunta de investigación y pueden ser refinados durante el proceso de búsqueda.

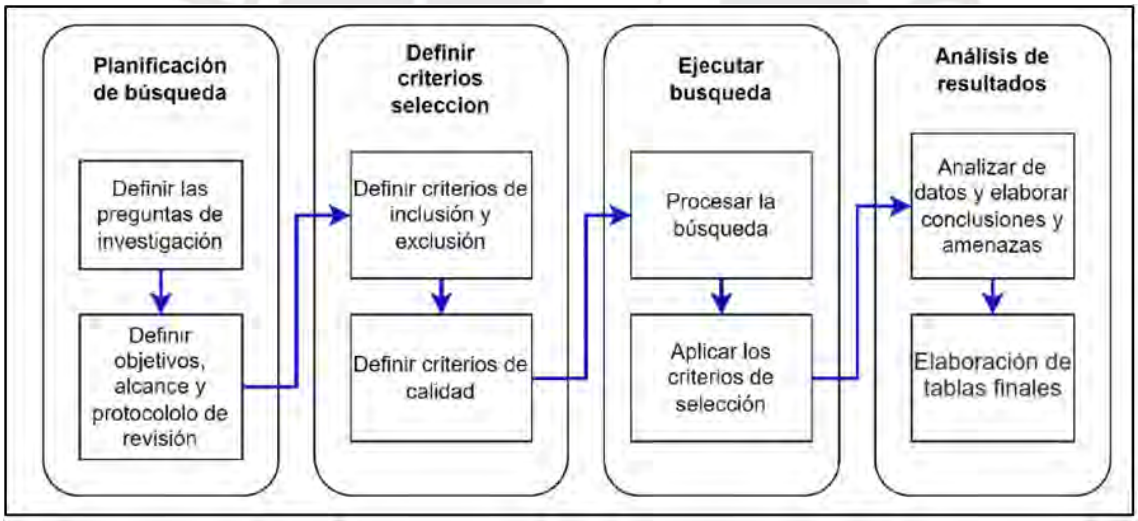
La evaluación de la calidad busca asegurar la credibilidad de los estudios a incluir desde el punto de vista de su imparcialidad y validez interna y externa.

El proceso de extracción de datos tiene como objetivo identificar en cada uno de los estudios los elementos de información que se necesita para responder a las preguntas de investigación.

El proceso de análisis de datos tiene el fin de sintetizar los datos de un modo tal que se responda a las preguntas de investigación.

Se ha elaborado el diagrama de la figura 1 en base a lo recomendado en [20] [21] para representar las etapas a seguir en el proceso de revisión.

Figura 1: Proceso seguido en las revisiones sistemáticas planteadas



3.2 Necesidad de la realización

Una revisión de la literatura sobre pruebas y calidad del software es crucial por varias razones. En primer lugar, proporciona una comprensión profunda del estado actual de la investigación, los métodos, metodologías y las mejores prácticas en pruebas de software y evaluación de calidad. Este conocimiento es esencial para que los investigadores, profesionales y organizaciones se mantengan actualizados con los últimos avances y estándares de la industria en ingeniería de software.

En segundo lugar, una revisión de la literatura ayuda a identificar brechas en la investigación existente, lo que puede guiar direcciones de investigación futuras y

contribuir al desarrollo de nuevas técnicas de prueba, modelos de calidad y metodologías de evaluación. Al comprender las limitaciones y las áreas que requieren mayor exploración, los investigadores pueden centrarse en abordar estas brechas para avanzar en el campo de las pruebas de software y la evaluación de la calidad.

Además, una revisión exhaustiva de la literatura ayuda a evaluar la efectividad y aplicabilidad de diferentes modelos de evaluación de la calidad del software y enfoques de prueba. Este análisis crítico es valioso para profesionales y organizaciones que buscan adoptar o adaptar modelos y metodologías existentes para mejorar sus procesos de desarrollo y prueba de software.

Adicionalmente, una revisión de la literatura puede ayudar a identificar desafíos comunes, mejores prácticas y factores de éxito en las pruebas de software y la evaluación de la calidad. Este conocimiento se puede utilizar para desarrollar pautas, estándares y marcos que promuevan el aseguramiento de la calidad y la excelencia en las pruebas en las prácticas de ingeniería de software.

3.3 Preguntas de investigación

Esta revisión sistemática tuvo como objetivo identificar los métodos de pruebas de software y su apoyo al aseguramiento de la calidad dentro del proceso de desarrollo de software. Según ello, se buscó responder a la siguiente pregunta de investigación principal: *¿De qué manera las pruebas de software contribuyen al aseguramiento de la calidad del proceso de desarrollo de software?*

Para estructurar los elementos de la pregunta de investigación se usó los criterios PICOC propuestos en [18], según se muestra en la tabla 1. Dado que como parte del estudio no se incluye la comparación entre métodos, metodologías, tipos o técnicas de pruebas de software, el elemento “Comparison” de PICOC no aplica.

Tabla 1: Criterios PICOC

Concepto	Ámbito de investigación
Population	Proceso de desarrollo de software
Intervention	Pruebas de software y criterios de aseguramiento de la calidad de software en el ámbito de pruebas
Comparison	no aplica
Outcome	Métodos, metodologías, tipos o técnicas
Context	Aplicaciones en la Academia o Industria de Ingeniería de Software

A partir de la pregunta general de investigación se formuló un conjunto de preguntas específicas para los diversos aspectos de los métodos de pruebas de software y su

contribución al aseguramiento de la calidad del proceso de desarrollo de software que se desea investigar:

Tabla 2:Preguntas de Investigación

Id	Pregunta de Investigación	Motivación
PI-1	¿Existe alguna consideración en los proyectos tradicionales o ágiles que determine el criterio de ejecución de pruebas de software?	Mediante esta pregunta buscamos determinar si la metodología o marco de desarrollo de software influye en la ejecución de los método, tipo o técnica para las pruebas de software.
PI-2	En relación con las pruebas de software ¿Como influyen en el aseguramiento de la calidad del proceso de desarrollo de software?	Al responder esta pregunta buscamos precisar la necesidad, beneficios, principios y métricas de las pruebas de software que contribuyen al aseguramiento de la calidad del proceso de desarrollo de software.
PI-3	En relación a las pruebas de software ¿Existen métodos, tipos o técnicas que permitan verificar y validar el desarrollo de software?	Al responder esta pregunta buscamos corroborar la existencia de métodos, tipos o técnicas de pruebas de software que sirvan de soporte para realizar un correcto desarrollo de software.
PI-4	¿Los casos de estudio encontrados se aplican en entornos académicos y/o industriales?	Mediante esta pregunta se pretende determinar la relevancia y aplicabilidad de los casos de estudio identificados en diferentes contextos. Podrían ser ámbitos académicos o industriales.

Además, se han establecido las siguientes preguntas de investigación bibliométricas asociadas.

Tabla 3: Preguntas bibliométricas

Id	Preguntas bibliométricas	Motivación
PB-1	¿Cuándo han sido publicados?	Mediante esta pregunta buscamos encontrar una tendencia en cuanto al uso de las pruebas de software en el tiempo.
PB-2	¿En qué medios se han publicado?	Mediante esta pregunta se busca indicar cuáles son los medios en los que más se publica este tipo de información.
PB-3	¿En qué países se hacen este tipo de investigaciones?	Al responder esta pregunta lo que se busca es identificar qué países más publican este tipo de información.

3.4 Estrategia de búsqueda

Para la definición de los términos de búsqueda se tomó en cuenta las sugerencias presentadas en [19], en particular las siguientes:

- Descomposición de la pregunta de investigación en sus componentes PICOC.
- Búsquedas preliminares usando varias combinaciones de términos de búsqueda derivadas de la pregunta de investigación.
- Adición de sinónimos, abreviaturas y formas ortográficas alternativas si aplicasen.
- Revisión de títulos, sumarios y palabras clave.
- Uso de operadores lógicos AND y OR.
- Verificación de que las cadenas de búsqueda de prueba recuperen la lista preliminar de estudios primarios ya identificados.

3.4.1 Cadena de búsqueda

En la siguiente tabla hemos definido los términos que formarán parte de la cadena de búsqueda:

Tabla 4: Definición de términos de búsqueda con PICOC

PICOC	Palabras de Búsqueda	Palabras en Inglés y sus Semejantes
Population	Proceso de desarrollo de software	Software development, software construction, software project, software process, software implementation
Intervention	Pruebas de Software y Calidad	Testing, quality, verification, validation
Comparison	No aplica	---------
Outcome	Métodos de Pruebas de Software	Methods, methodologies, types
Context	Aplicados en la Ingeniería de Software	Aplication in software engineering on academia or industry

Para luego obtener la siguiente cadena general de búsqueda:

Cadena: (**P** AND **I** AND **O** AND **C**)

Tabla 5: Cadena de Búsqueda

(("software development" OR "software construction" OR "software project*" OR "software process*" OR "software implementation") AND ("software testing") AND (“quality" OR "verification" OR "validation") AND ("method*" OR "type*") AND ("software engineering"))

3.4.2 Selección de fuentes

La selección de bases de datos fue realizada a partir de revisiones sistemáticas, el criterio de calidad de publicaciones y bases de datos especializadas en ingeniería. Las bases de datos consultadas fueron:

Tabla 6: Fuentes de datos revisadas

FUENTE DE DATOS	URL
Scopus	https://www.scopus.com
IEEE	https://ieeexplore.ieee.org
ACM Digital Library	https://dl.acm.org

3.4.2 Criterios de inclusión y exclusión

Para identificar los estudios más adecuados, se han definido un conjunto de criterios clasificados en inclusivos y exclusivos.

Tabla 7: Criterios de inclusión

Id	Criterios de inclusión	Motivación
CI-1	Que el estudio publicado mencione métodos, tipos o técnicas de prueba de software.	Buscamos centrarnos en los estudios que mencionen métodos, tipos o técnicas de pruebas de software.
CI-2	Que el estudio publicado mencione las palabras clave en la introducción.	Buscamos centrarnos en aquellos estudios etiquetados con las palabras clave que hemos propuesto en la cadena de búsqueda.
CI-3	Que el estudio aborde aspectos de la calidad en las pruebas de software.	El estudio debe hacer referencia a temas de pruebas de software y su relación con la calidad.

Tabla 8: Criterios de exclusión

Id	Criterios de exclusión	Motivación
CE-1	Que el estudio publicado se encuentre en un idioma diferente al inglés o español.	Estudios redactados en un idioma distinto al inglés o español no serán tomados en cuenta.
CE-2	Que el estudio publicado trate temas diferentes a pruebas de software y la calidad.	Estudios que no mencionen temas de pruebas de software y la calidad no serán tomados en cuenta.
CE-3	Que el estudio aborde el mismo tema que uno ya seleccionado, pero con menor detalle.	Se tomará en cuenta solo el estudio que profundice más sobre el tema.
CE-4	Que el estudio no debe pertenecer a una revisión o mapeo sistemático de la literatura.	Los estudios deben ser primarios, el resto no será tomado en cuenta.

3.4.3 Criterios de estimación de calidad

El instrumento de evaluación aplicado está basado en la propuesta de Zarour [22] el cual utiliza una escala de calificación de 3 niveles de cumplimiento. Si el estudio sometido a evaluación **S cumple** con satisfacer la pregunta de aseguramiento de calidad se le asigna 1 punto, si **No cumple** se le asigna un puntaje de 0 y si **cumple Parcialmente** se le asigna 0.5 puntos.

Tabla 9: Criterios de estimación de calidad

Id	Criterios de estimación de calidad
CC-1	¿El objetivo de la investigación es lo suficientemente explicado?
CC-2	¿La idea o el enfoque presentado han sido claramente explicados?
CC-3	¿Se han considerado amenazas a la validez?
CC-4	¿Hay una adecuada descripción del contexto en donde se ha llevado a cabo la investigación?
CC-5	¿Se mencionan de forma clara los hallazgos del estudio?

Todas aquellas publicaciones relevantes con una puntuación total igual o menor a 5.0 y mayor o igual a 2.5 son consideradas como aceptadas; mientras que aquellas con un puntaje menor a 2.5 son descartadas.

Tabla 10: Estimación de calidad

Identificador	CC-1	CC-2	CC-3	CC-4	CC-5	Puntuación total
S01	1	1	1	1	1	5
S02	1	1	0	1	1	4
S03	1	1	0	0,5	0,5	3
S04	1	1	0	0	1	3
S05	1	0,5	0	0,5	1	3
S06	1	0,5	0	0,5	0,5	2,5
S07	1	0,5	0	1	0,5	3
S08	1	0,5	0	1	1	3,5
S09	1	0,5	0	0,5	1	3
S10	1	1	0	1	1	4
S11	1	1	0	1	1	4
S12	1	1	1	0,5	1	4,5
S13	1	1	0	0,5	1	3,5
S14	1	1	0	0,5	1	3,5
S15	1	1	0	0,5	0,5	3
S16	1	1	0	1	1	4
S17	1	1	1	1	1	5
S18	1	0,5	1	0,5	1	4
S19	1	1	1	1	1	5
S20	1	1	1	0,5	1	4,5
S21	1	1	1	1	1	5
S22	1	1	1	1	1	5
S23	1	1	0	0	1	3
S24	1	1	0	0	0,5	2,5
S25	1	1	0	0	1	3
S26	1	1	0	0,5	0,5	3
S27	1	1	1	1	1	5
S28	1	1	0	1	1	4
S29	1	1	1	0	1	4
S30	1	1	1	1	1	5
S31	1	1	1	1	1	5
S32	1	1	0	0	1	3
S33	1	0,5	0	1	1	3,5
S34	1	0,5	0	0,5	1	3
S35	1	1	0	1	1	4
S36	1	1	0	1	1	4
S37	1	1	1	0,5	1	4,5
S38	1	1	0	0,5	1	3,5
S39	1	1	0	0,5	1	3,5
S40	1	1	0	0,5	0,5	3
S41	1	1	0	1	1	4
S42	1	1	1	1	1	5

S43	1	0,5	1	0,5	1	4
S44	1	1	1	1	1	5
S45	1	1	1	0,5	1	4,5
S46	1	1	1	1	1	5
S47	1	1	1	1	1	5
S48	1	1	0	0	1	3
S49	1	1	1	0,5	1	4,5
S50	1	1	0	0,5	1	3,5
S51	1	1	0	0,5	1	3,5
S52	1	1	0	0,5	0,5	3
S53	1	1	0	1	1	4
S54	1	1	0	0	1	3
S55	1	0,5	0	1	1	3,5

Podemos ver que las publicaciones revisadas cumplen con el criterio de calidad que se definió.

3.4.5 Estrategia para la extracción de los datos

La información de los estudios seleccionados fue extraída y registrada en vistas a responder a las preguntas de investigación. A continuación, se describe el formulario y el procedimiento empleados en esta etapa.

a) Formulario de extracción de datos

El formulario de extracción de datos contuvo tanto la información general sobre cada estudio como los detalles completos que permitiesen describir cómo cada uno de ellos respondía a las preguntas de investigación de la revisión sistemática. El formulario constó de los campos indicados en la tabla 11.

b) Procedimiento para la extracción de datos

Para cada uno de los estudios seleccionados, se completó los formularios de extracción indicados en la sección anterior a). Se leyó completamente la introducción y conclusión de los estudios. Se leyó con detenimiento el texto restante en la medida que fue necesario para extraer los datos requeridos.

Tabla 11: Formulario de extracción de datos

Campo	Descripción	Pregunta de investigación
Id	Identificador de la publicación en nuestra clasificación	General
Fecha de extracción	Fecha en la que se ejecutó la búsqueda	General
Titulo	Título del estudio	General
Tipo de fuente	Fuente del estudio identificado	General
Año de publicación	Año de publicación del estudio	General

País	País del autor	General
Metodología de desarrollo de software utilizada	Bajo qué metodología o marco de trabajo fue gestionado el proyecto del caso de estudio.	PI-1
Explicación de la prueba de software utilizada	Descripción de la prueba de software y su ejecución dentro del ciclo de vida de desarrollo de software	PI-1
Necesidad y beneficio de las pruebas de software	Las necesidades y beneficios de las pruebas de software que describen los casos de estudio	PI-2
Principios de la prueba de software utilizado	Los principios de las pruebas de software mencionados en los casos de estudio	PI-2
Métricas relevantes	¿Cómo mide sus resultados?	PI-2
Método, tipo o técnica utilizado como soporte del proceso de pruebas de software	Cuál es el método, tipo o técnica que se utiliza en el caso de estudio	PI-3
Nivel de prueba de software	Niveles de pruebas de software identificados en los casos de estudio.	PI-3
Dominio de la aplicación de los casos	En qué contexto se realizó el estudio, si fue académico o industrial	PI-4

3.4.6 Estrategia para la esis de los datos

Una vez que se extrajo los datos de cada estudio primario, se consolidó y tabuló la información correspondiente a cada una de las cuatro preguntas de investigación. Las tablas obtenidas fueron utilizadas para facilitar el análisis y síntesis de la información en vistas a responder cada una de las preguntas de investigación planteadas.

3.5 Resultados y análisis

3.5.1 Búsqueda y selección de estudios primarios

El proceso de selección de los documentos fue el siguiente:

a) Primer filtro

Se ejecutó la cadena de búsqueda en la base de datos Scopus la cual devolvió 512 estudios (el 06 de septiembre de 2022), luego de una primera revisión inicial en función del tema del estudio, se selecciona 89 estudios.

Por otro lado, luego de ejecutar la cadena de búsqueda en la base de datos IEEE se obtuvieron 425 estudios, de los cuales mediante el primer filtro 45 estudios resultaron relevantes.

Adicionalmente, se ejecutó la cadena de búsqueda en las bases de datos ACM y se obtuvieron 3 estudios, de los cuales no fueron relevantes para el estudio, por tanto, quedaron descartados.

b) Segundo filtro

De los 89 estudios de Scopus, se eliminaron aquellos que no cumplían con los criterios de inclusión. Se selecciona 53 estudios.

Por el lado de IEEE, se selecciona 30 estudios relevantes para nuestra revisión sistemática.

c) Tercer filtro

Finalmente, se revisaron las introducciones y conclusiones de los 53 estudios para ver si respondían las preguntas de investigación propuestas (por lo menos una de ellas), luego de la revisión se selecciona 40 estudios en Scopus.

Del mismo modo, en el caso de IEEE, se realizó la revisión de las introducciones y conclusiones de los 30 artículos, de los cuales, 15 estudios respondían por lo menos una de las preguntas de investigación. Por lo que en total se selecciona 55 estudios.

Tabla 12: Resultados de búsqueda

Bases de Datos	Resultados de búsqueda	Documentos Relevantes	Filtro Inclusión/Exclusión	Filtro preguntas de investigación
Scopus	512	89	53	**40**
IEEE	425	45	30	**15**
ACM	3	0	0	**0**

Total	**940**	**134**	**83**	**55**

En la tabla 12 se sintetiza el resultado final de las búsquedas realizadas en las bases de datos electrónicas.

A continuación, se lista los artículos seleccionados:

Estudio	Titulo	Año
S01	A formal approach to software error removal [23]	1987
S02	Toward a quality inspection method and management [24]	1995
S03	Increasing testing productivity and software quality: A comparison of software testing methodologies within NASA [25]	1996
S04	An overview of software testing [26]	1997
S05	"Continuous verification" in mission critical software development [27]	1997
S06	A practical method for verifying event-driven software [28]	1999
S07	Development of a software security assessment instrument to reduce software security risk [29]	2001
S08	Improving software testing via ODC: Three case studies [30]	2002
S09	On estimating testing effort needed to assure field quality in software development [31]	2002
S10	Model-based testing of object-oriented systems [32]	2003
S11	Test-driven development as a defect-reduction practice [33]	2003
S12	An industrial case study of the verification and validation activities [34]	2003
S13	Effective test driven development for embedded software [35]	2006
S14	The economics of unit testing [36]	2006
S15	On the influence of test-driven development on software design [37]	2006
S16	Defining Agile Software Quality Assurance [38]	2006
S17	Testing Critical Software: A Case Study for an Aerospace Application [39]	2006
S18	Does test-driven development improve the program code? Alarming results from a comparative case study [40]	2008
S19	An approach to testing black-box components using contract-based mutation [41]	2008
S20	Evaluating automated unit testing in Sulu [42]	2008
S21	From waterfall to evolutionary development and test [43]	2009
S22	An Automatic Compliance Checking Approach for Software Processes [44]	2009
S23	What Makes Testing Work: Nine Case Studies of Software Development Teams [45]	2009
S24	Porantim-opt: Optimizing the combined selection of model-based testing techniques [46]	2011
S25	CRANE: Failure prediction, change analysis and test prioritization in practice - Experiences from Windows [47]	2011
S26	Generating test data for black-box testing using genetic algorithms [48]	2012
S27	Software test automation practices in agile development environment: An industry experience report [49]	2012
S28	Hadoopmutator: A cloud-based mutation testing framework [50]	2014
S29	N-Tiered test automation architecture for Agile software systems [51]	2014
S30	Applying black-box testing to UML/OCL database models [52]	2014
S31	A method for the selection of software testing techniques using analytic Hierarchy process [53]	2015
S32	Vision 2020: The future of software quality management and impacts on global user acceptance [54]	2015
S33	Investigating the effect of "defect co-fix" on quality assurance resource allocation: A search-based approach [55]	2015
S34	Using statistical usage testing in conjunction with other black box testing techniques [56]	2015
S35	Evaluating the Effectiveness of BEN in Localizing Different Types of Software Fault [57]	2016

S36	Unit Test Generation during Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins [58]	2016
S37	Model based testing of satellite on-board software - An industrial use case	2016
S38	Agile testing [59]	2016
S39	A comparative study of software testing techniques [60]	2017
S40	Effectiveness assessment of an early testing technique using model-level mutants [61]	2017
S41	Applying usability testing to improving Scrum methodology in develop assistant information system [62]	2017
S42	Regression Testing of Database Applications under an Incremental Software Development Setting [63]	2017
S43	Software quality assurance during implementation: Results of a survey in software houses from Germany, Austria and Switzerland [64]	2017
S44	Aspect oriented software testing [65]	2017
S45	SBSTFrame: A framework to search-based software testing [66]	2017
S46	Implementation of software testing practices in Pakistan's software industry [67]	2018
S47	A Study of Software Testing Practices in Sri Lankan Software Companies [68]	2018
S48	Test-Driven Development in HPC Science: A Case Study [69]	2018
S49	Assessing the Effectiveness of Test-Driven Development and Behavior-Driven Development in an Industry Setting [70]	2019
S50	Framework for collaborative software testing efforts between cross-functional teams aiming at high quality end product [71]	2019
S51	Practices of Software Testing Techniques and Tools in Bangladesh Software Industry [72]	2019
S52	TestSage: Regression Test Selection for Large-Scale Web Service Testing [73]	2019
S53	Mobile Application testing tools and their challenges: A comparative study [74]	2019
S54	An effective approach for context driven testing in practice - A case study [75]	2020
S55	Mutation Testing and Self/Peer Assessment: Analyzing their Effect on Students in a Software Testing Course [76]	2021

3.5.2 Extracción y s ntesis de datos

La búsqueda se realizó el 06 de septiembre de 2022 cuyo resultado fue un conjunto de publicaciones en el rango de fechas entre 1987 y 2022; este resultado obtenido fue nuestra base para posteriormente realizar la revisión sistemática.

Los estudios primarios obtenidos luego fueron leídos completamente a fin de extraer la información necesaria mediante el formulario de extracción de datos.

3.5.3 Visión general de los estudios incluidos

En esta sección buscamos dar respuesta a las preguntas bibliométricas que nos planteamos en un principio:

a) Años de publicación

En la figura 2 se encuentran las publicaciones por año. Podemos ver que la mayor cantidad de publicaciones las tenemos en el año 2009 y en segundo lugar el año 2019, esta diferencia puede reflejar cambios en las prioridades de investigación, la evolución de la tecnología y los enfoques en pruebas de software, así como la

diversificación de intereses dentro de la comunidad académica y la industria de software a lo largo del tiempo.

Figura 2: Publicaciones por año

b) Medios de publicación

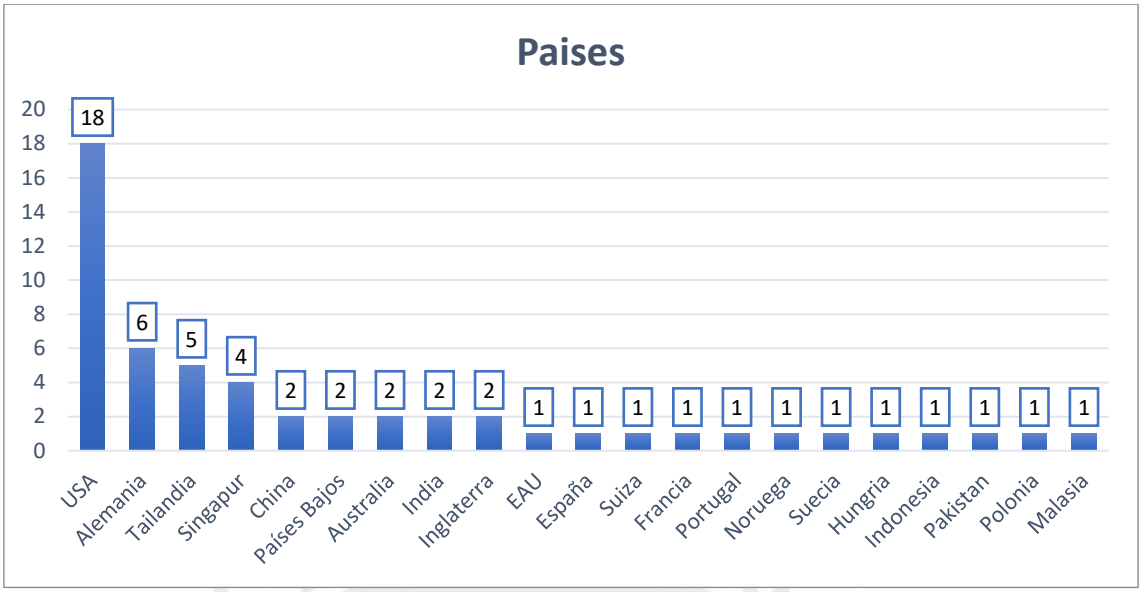
Otra de las preguntas bibliométricas que nos planteamos responder corresponde a los medios de publicación en donde se han presentado los trabajos que seleccionamos, Podemos ver en la tabla 13 que la mayoría de las publicaciones (45 en total) son del tipo “Conference paper”, luego siguen los tipos “Artículos” con 8 publicaciones y 2 publicaciones como capítulo de libro.

Tabla 13: Medios de publicación

Año de Publicación	Article	Book chapter	Conference paper	Total general
1987	1			1
1995			1	1
1996			1	1
1997			2	2
1999			1	1
2001			1	1
2002	1		1	2
2003	1		2	3
2006			5	5
2008	1		2	3
2009		1	2	3
2011			2	2
2012			2	2
2014	1		2	3
2015	1		3	4
2016		1	3	4
2017	1		6	7
2018			3	3
2019			5	5
2020	1			1
2021			1	1
Total general	**8**	**2**	**45**	**55**

c) P donde se realizaron los estudios

Figura 3: Publicaciones por países



Mediante la Figura 3, podemos ver que los países en donde se han realizado más investigaciones sobre el tema de estudio son Estado Unidos de América, Alemania y Tailandia.

3.5.4 Resultados a las preguntas de investigación

3.5.4.1 Pregunta 1:

¿Existe alguna consideración en los proyectos tradicionales o ágiles que determine el criterio de ejecución de pruebas de software?

Mediante esta pregunta buscamos determinar si la metodología o marco de desarrollo de software influye en la ejecución del método, tipo o técnica para las pruebas de software.

Al abordar los criterios de ejecución de las pruebas de software en los proyectos tradicionales frente a los ágiles, existen distintas consideraciones o factores que guían el proceso de pruebas en cada uno de ellos.

La tabla 14, muestra los factores recogidos de los estudios, hallando como factor de mayor relevancia en la ejecución de pruebas de software, la selección de metodología de desarrollo de software tradicional o ágil.

Tabla 14: Factores que influyen en la ejecución de pruebas de software

Factor	Explicación	Estudios
Metodologías de Desarrollo de Software	En los proyectos tradicionales, las pruebas suelen estar más estructuradas y separadas en fases específicas, como pruebas unitarias, de	[23],[24],[25],[26],[27],[28],[30], [31],[32],[33],[34],[35],[36],[37], [38],[39],[40],[41],[42],[43],[44], [45],[49],[50],[51],[52],[54],[56], [59],[60],[62],[63],[64],[65],[66],

	integración y de sistema. En contraste, en los proyectos ágiles, las pruebas son continuas y se integran de manera más fluida en todo el ciclo de desarrollo.	[68],[69],[70],[71],[72],[73],[74], [76]
Priorización de Funcionalidades	Es esencial priorizar las funcionalidades críticas o de alto riesgo para enfocar los esfuerzos de prueba en áreas que tienen un impacto significativo en la calidad del software.	[55],[76],[77],[58],[49],[72],[61], [57]
Ciclos de Desarrollo y Entrega	En proyectos ágiles, donde se realizan entregas incrementales y frecuentes, las pruebas deben adaptarse para garantizar que cada iteración cumpla con los estándares de calidad establecidos.	[60],[27],[63],[53],[39],[59],[37], [40]
Recursos Disponibles	La disponibilidad de recursos humanos, herramientas de prueba y entornos de prueba puede influir en la planificación y ejecución de las pruebas de software en ambos enfoques de desarrollo.	[63],[52],[74],[48],[71],[38],[69], [60],[49]
Restricciones de Tiempo y Presupuesto	Las limitaciones de tiempo y presupuesto son factores críticos que pueden afectar la extensión y profundidad de las pruebas realizadas. Es necesario equilibrar la cobertura de pruebas con los recursos disponibles.	[27],[52],[58],[62],[60]

En los proyectos tradicionales (a menudo en cascada), las pruebas de software son una etapa que suele venir después de las fases de planificación detallada, recopilación de requisitos, diseño del sistema e implementación. En este caso, la fase de pruebas es formal, estructurada y suele producirse tarde en el ciclo del proyecto. Si se identifican defectos, pueden dar lugar a cambios significativos que pueden resultar costosos debido a la naturaleza secuencial del proceso de desarrollo [55].

Los proyectos ágiles, por el contrario, incorporan las pruebas como una actividad continua durante todo el ciclo de desarrollo. Las metodologías ágiles, como Scrum o Kanban, hacen hincapié en la integración continua y, como resultado, las pruebas se realizan de forma incremental a medida que se desarrollan las características. Este enfoque continuo ayuda a los equipos a identificar y abordar los problemas desde el principio, lo que puede suponer un ahorro de tiempo y costes. Además, el enfoque ágil favorece la adaptabilidad, lo que permite a los equipos responder rápidamente a los cambios, incluso en una fase avanzada del proyecto [72].

Mientras que las pruebas tradicionales pueden ser compartimentadas y por fases, los proyectos ágiles emplean prácticas como el desarrollo orientado a pruebas (TDD), en el que las pruebas se escriben antes que el código y la codificación se orienta a hacer que las pruebas pasen, garantizando así una garantía de calidad continua [66], [72].

Además, con la agilidad, a menudo se hace hincapié en las pruebas automatizadas para apoyar las rápidas iteraciones de las versiones de software. Las pruebas continuas, como parte de las canalizaciones de integración continua/despliegue continuo (CI/CD) en una cultura DevOps, mejoran también esta metodología [72].

Las pruebas de software, independientemente de la metodología, siguen un proceso estructurado que consiste generalmente en [53],[72],[59],[63],[55],[44],[76],[74]:

a. Planificación de las pruebas: Determinación del alcance, los recursos y el calendario.
b. Diseño de casos de prueba: Creación de casos basados en la documentación y los requisitos del sistema.
c. Configuración del entorno de pruebas: Preparación del entorno para las pruebas.
d. Ejecución de las pruebas: Ejecución de los casos de prueba y evaluación de los resultados.
e. Notificación de errores: Documentar y notificar los defectos.

En la decisión de realizar pruebas de software también influyen factores como los requisitos de fiabilidad, las limitaciones de tiempo, las consideraciones presupuestarias y la complejidad del sistema que se está desarrollando [55], [66], [61], [65], [46], [31].

Aunque los enfoques de las pruebas pueden diferir en función de la metodología de desarrollo elegida, las pruebas de software en general tratan de garantizar que el producto final cumpla los requisitos del usuario y pueda funcionar en distintos entornos [66], [65], [28].

Asimismo, existen diversas metodologías de desarrollo de software tradicionales o ágiles que se mencionan en diferentes contextos y proyectos. A continuación, se presentan algunas de las metodologías identificadas:

Metodologías de desarrollo de software tradicionales:

- Modelo en cascada [56]
- Metodología RUP (Rational Unified Process) [56]
- Método XP (eXtreme Programming) [56]

Metodologías de desarrollo de software ágil:

- Scrum [56], [31], [47]
- Kanban [31], [47], [72]
- DSDM (Dynamic Systems Development Method) [31]
- SAFe (Scaled Agile Framework) [48], [49], [72]
- Crystal [31]
- FDD (Feature-Driven Development) [31]
- Lean Startup [31]
- Modelo Spotify [49]

En conclusión, para elegir las metodologías de prueba adecuadas y alinearlas con las limitaciones del proyecto, hay que tener en cuenta la naturaleza del proyecto, la familiaridad del equipo con las herramientas necesarias, el dominio de la aplicación y la criticidad del software [28].

La elección de la ejecución de las pruebas en los proyectos de software depende de si se emplea la metodología tradicional o la ágil, impulsada por los hitos específicos y las filosofías que guían cada una. La estrategia de pruebas también debe tener en cuenta la complejidad del proyecto, la criticidad del software y cualquier limitación temporal o presupuestaria.

3.5.4.2 Pregunta 2:

En relación con las pruebas de software ¿Como influyen en el aseguramiento de la calidad del proceso de desarrollo de software?

Al responder esta pregunta buscamos precisar la necesidad, beneficios, principios y métricas de las pruebas de software que contribuyen al aseguramiento de la calidad del proceso de desarrollo de software.

En la Tabla 15, se detalla las múltiples funciones esenciales que cumplen las pruebas de software a la hora de garantizar la calidad del proceso de desarrollo de software.

Tabla 15: Función de las Pruebas en Relación con el Aseguramiento de la Calidad

Función de las Pruebas en Relación con la Calidad	Explicación	Estudios
Verificación y validación	Verifica si el software cumple los requisitos especificados y valida que el producto final se ajusta a las necesidades y expectativas del usuario.	[72],[76],[32],[29], [63],[55],[54],[53]
Mejora de la calidad	Las pruebas mejoran la calidad general del producto de software identificando y rectificando los defectos antes de su despliegue, lo que conduce a un producto más sólido y fiable.	[37],[45],[34],[68], [59],[31],[62]
Reducción de costes	Identificar y abordar los errores en una fase temprana del proceso de desarrollo del software suele ser menos costoso que corregirlos después de la implantación.	[53],[59],[34],[55]
Detección de errores	Las pruebas de software ayudan a detectar errores que podrían haberse introducido durante las fases de desarrollo y diseño, evitando así que los defectos lleguen a los clientes.	[55],[53],[76],[45], [56],[47],[31],[40], [61],[72]
Seguridad y usabilidad	Puede revelar problemas de seguridad y mejorar la usabilidad y la experiencia del usuario, haciendo que el software sea más fácil de usar y seguro.	[72],[29],[49],[63]

Mantenimiento a largo plazo	Las pruebas reducen los costes de mantenimiento a largo plazo al identificar a tiempo problemas cuya solución sería más costosa posteriormente.	[53],[72],[76],[61], [46]
Cumplimiento de los requisitos	Garantiza que el software se adhiere a los requisitos y especificaciones definidos, satisfaciendo así las necesidades del cliente.	[41],[39],[51],[37], [72]
Eficacia y resultados	Se ha observado que las pruebas dinámicas y de caja blanca mejoran la eficiencia de los programadores, sobre todo en contextos de desarrollo ágil, y ayudan a generar confianza en la calidad del producto de software final.	[37],[61],[53],[41], [55],[72]

Las pruebas de software influyen significativamente en el aseguramiento de la calidad del proceso de desarrollo del software. Así mismo, son esenciales en el ciclo de vida del desarrollo de software para garantizar la calidad del producto desarrollado. Su objetivo es identificar errores y defectos en el software antes de que sea lanzado al mercado, lo que puede ahorrar tiempo, esfuerzo y dinero. [45], [73], [30], [60]

La necesidad de las pruebas de software esencialmente radica en garantizar la calidad del software entregado, prevenir errores y corregir defectos en el software antes de su lanzamiento.

Diversos documentos consultados describen la necesidad y los beneficios de las pruebas de software, aportando lo siguiente:

- Garantizar la calidad y la confiabilidad del software, especialmente en términos de los sistemas de software críticos para la misión, donde tanto la seguridad de las personas como equipos costosos pueden estar en juego. [29]
- Afrontar la creciente complejidad del software y la importancia crítica de su fiabilidad y calidad en el uso cotidiano. [38]
- Garantizar la calidad del producto de software entregado y son una parte integral del aseguramiento de la calidad durante la fase de implementación. Las pruebas permiten identificar y corregir errores antes del lanzamiento del producto y, con ello, se pueden prevenir problemas costosos y daños a la reputación de la empresa. [63]
- Necesarias para identificar y corregir errores antes de que el software se despliegue a los usuarios finales, lo que ayuda a evitar costos adicionales y mejora la satisfacción del cliente. [71]
- Necesarias para garantizar que el software funcione correctamente y cumpla con los requisitos especificados, a la vez que mejora la calidad del software y garantiza su fiabilidad y seguridad para los usuarios finales. [47]
- Ayudan a detectar y corregir errores que podrían haberse escapado durante la programación y mejorar la calidad del software entregado. [31], [73]
- Ayudan a prevenir posibles interrupciones de servicios y mejoran la satisfacción del usuario final. [73], [30]
- Permiten tomar mejores decisiones en el proceso de desarrollo del software, donde la información obtenida puede ser utilizada como una medida de calidad general en el ciclo de vida del software. [73]

- Reducen el costo de corrección de los errores, además de mejorar la seguridad y la confiabilidad de los sistemas de software. [31].
- Identificar y corregir defectos antes del lanzamiento del software, lo que puede reducir los costos de mantenimiento y aumentar la satisfacción del usuario final. [70], [60], [44]
- Prevenir defectos en las etapas tempranas del desarrollo y evitar problemas más costosos más adelante. [60], [44]
- Revelar problemas de seguridad y mejorar la usabilidad y la experiencia del usuario.[60]
- Validar la confiabilidad, seguridad y rendimiento del software antes de su despliegue y uso en entornos de producción. [44]
- Contribuir a prevenir fallos críticos en producción. [44]
- Detectar y corregir errores en el software antes de que llegue a los usuarios finales, lo que ayuda a prevenir fallos en el uso del producto. [49]
- Aumentar la calidad general del software, lo que se traduce en una mayor satisfacción del usuario y una reducción de los costos de mantenimiento a largo plazo. [49]
- Corregir errores de manera más económica en las etapas tempranas del desarrollo. [49]
- Ayudar a identificar y corregir errores antes de que el producto sea desplegado, asegurando que las nuevas características o modificaciones no introduzcan fallos en versiones anteriores del producto. [62]
- Reducir los costos asociados con la corrección de fallos, que tienden a ser más altos si se detectan después del lanzamiento del producto. [62]
- Evitar problemas de cumplimiento de normativas y requisitos regulatorios. [73]
- Permitir la toma de decisiones informadas en el proceso de desarrollo del software. [73]
- Proporcionar retroalimentación temprana sobre las funcionalidades y características del software, lo que ayuda a ajustarlas en consecuencia. [73]
- Mejorar el proceso de gestión del proyecto y la capacidad de anticipar y prevenir problemas. [73]
- Promover una cultura de mejora continua en el proceso de desarrollo de software, ayudando a mantener un elevado nivel de calidad. [73]

Los principios de las pruebas de software son fundamentales para garantizar la calidad y eficacia del proceso de pruebas. A partir de los documentos consultados, se pueden mencionar los siguientes principios, donde las pruebas deben:

- Ser sistemáticas y planificadas en un proceso estructurado y planificado para ser efectivas. [62]
- Comenzar temprano en el ciclo de vida del software para identificar y corregir errores más rápidamente y a un menor costo. [62], [32], [56]
- Verificar y validar que el software realiza las funciones para las que fue diseñado, asegurando que la implementación sea consistente con las especificaciones. [32]
- Ser tempranas y continuas, realizándose de manera constante a lo largo del proceso de desarrollo del software. [32], [56]
- Enfocarse en áreas del software que presenten el mayor riesgo de contener defectos, priorizando las pruebas en función de la importancia y probabilidad de fallos. [56]
- Ser independientes de los desarrolladores, lo que ayuda a evitar prejuicios y fomenta la objetividad en el proceso de pruebas. [4], [73]

- Las pruebas automatizadas son cada vez más esenciales en entornos ágiles, ya que aceleran el proceso de aseguramiento de la calidad, documentan el software y reducen costos. [37]
- La repetibilidad de las pruebas es fundamental para garantizar la eficacia del proceso de pruebas. [60], [32]
- Evaluar la funcionalidad, la fiabilidad, el rendimiento, la seguridad y la usabilidad del software. [73],[30]
- Ser diseñadas para buscar no solo casos de éxito, sino también casos de fallo y límites de los sistemas. [73]
- Ser replicable y se debe documentar para que sea posible trazar los resultados y garantizar la trazabilidad. [73], [38]
- Ser realizadas por personas con experiencia y habilidades adecuadas. [31], [30]
- Ser ejecutadas con herramientas, técnicas y metodologías apropiadas para optimizar la eficacia y la eficiencia del proceso de pruebas. [31],[11]
- Tener objetivos claros y medibles, y el resultado de las pruebas debe estar claramente definido. [30],[38]

En relación con las métricas de calidad que se evidencias en los artículos consultados, se mencionan las siguientes métricas relevantes en el contexto de las pruebas de software:

- Cobertura de código: Mide el porcentaje de líneas de código que son ejecutadas durante las pruebas. [63], [32]
- Número de defectos encontrados: Mide la cantidad de errores o defectos encontrados durante el proceso de pruebas. [63], [32]
- Tiempo medio de detección y corrección de defectos: Mide el tiempo medio entre la detección y la corrección de un defecto en el software. [63]
- Tasa de defectos por líneas de código: Mide la cantidad de defectos encontrados por cada 1000 líneas de código. [63], [32]
- Satisfacción del usuario: Mide el grado en que el software cumple con las expectativas y necesidades del usuario final. [63]
- Tasa de errores residuales: Mide la cantidad de errores que aún persisten en el software después del proceso de corrección de defectos. [63]
- Efectividad de las pruebas: Mide la capacidad de las pruebas para detectar defectos en el software. [25]
- Eficiencia de las pruebas: Mide la capacidad de las pruebas para detectar defectos en el software con un mínimo de esfuerzo y recursos. [25], [57]
- Costo de las pruebas: Mide el costo total de realizar pruebas de software durante el ciclo de vida del desarrollo de software. [25], [58]
- Tasa de repetición de pruebas: Mide la cantidad de veces que se deben repetir las pruebas antes de que se alcance un nivel adecuado de calidad del software. [58]
- Tasa de defectos críticos: Mide la cantidad de defectos que representan un riesgo crítico para el desempeño o la seguridad del software. [73]
- Tasa de falsos positivos: Mide la cantidad de pruebas que se identifican como defectos cuando en realidad son un resultado legítimo del software. [73]
- Tiempo de ejecución de las pruebas: Mide el tiempo necesario para completar el conjunto determinado de pruebas. [32]

- Tasa de pruebas completadas con éxito: Mide la cantidad de pruebas que se han completado satisfactoriamente en comparación con el total de pruebas planificadas. [32]
- Tasa de cobertura de requisitos: Mide la cantidad de requisitos funcionales y no funcionales del software que han sido probados. [31]

En conclusión, las pruebas de software son un elemento indispensable de la garantía de calidad en el ciclo de vida del desarrollo de software. Abarca la verificación, la validación, la mejora de la calidad, la reducción de costes, la prevención de defectos y la mejora de la fiabilidad y el rendimiento, todo lo cual contribuye a obtener un producto de software superior listo para el mercado [9], [10].

La eficacia de las pruebas de software reside en su enfoque exhaustivo para validar sistemáticamente diversos aspectos del software, garantizando no sólo la corrección funcional sino también el rendimiento en diferentes condiciones y entornos de usuario. Estas pruebas exhaustivas infunden confianza tanto a los desarrolladores como a los usuarios en cuanto a la calidad del producto de software final [8].

3.5.4.3 Pregunta 3:

En relación a las pruebas de software ¿Existen métodos, tipos o técnicas que permitan verificar y validar el desarrollo de software?

Al responder esta pregunta buscamos corroborar la existencia de métodos, tipos o técnicas de pruebas de software que sirvan de soporte para realizar un correcto desarrollo de software.

En la Tabla 16, se muestran los métodos de pruebas que existen y la ocurrencia encontrada en los estudios revisados.

Tabla 16: Métodos de Pruebas de Software

Método de Prueba	Explicación	Estudios
Estáticas	Se enfocan en la revisión del código fuente y la documentación para encontrar posibles defectos sin necesidad de ejecutar el programa.	[23],[24],[25],[26],[27],[28],[29],[30],[31],[32],[33],[34],[35],[36],[37],[38],[39],[40],[41],[42],[43],[44],[45],[46],[47],[49],[50],[51],[52],[53],[54],[55],[58],[61],[62],[63],[64],[65],[66],[67],[68],[70],[71],[72]
Dinámicas	Las pruebas dinámicas siguen un enfoque de verificación continua y rigurosa, esencial para garantizar la calidad y confiabilidad del software.	[26],[27],[28],[29],[30],[31],[32],[33],[34],[35],[36],[37],[38],[39],[40],[41],[42],[43],[44],[45],[46],[47],[49],[50],[51],[52]

En los artículos se mencionan distintos métodos, tipos y técnicas de pruebas de software que permiten verificar y validar el desarrollo de software.

A continuación, se brinda los tipos de pruebas de software mencionados en los estudios clasificados según el método usado:

Pruebas que usan el método estático:

- Pruebas estáticas (Static testing): Pruebas que se realizan sin ejecutar el software y que pueden incluir revisión de código, análisis estático de código, entre otras técnicas para identificar errores en el software. [74]
- Pruebas exploratorias (Exploratory testing): Técnica de pruebas en la que el probador utiliza su experiencia y conocimiento para descubrir errores en el software. [73]

Pruebas que usan el método dinámico:

- Pruebas de caja negra (Black-box testing): Pruebas basadas en la especificación de lo que el software debe hacer y no en cómo lo hace. [35]
- Pruebas de caja blanca (White-box testing): Pruebas basadas en el conocimiento de la estructura interna del software y su implementación. [35]
- Pruebas de humo (Smoke testing): Pruebas iniciales y superficiales destinadas a comprobar si el software es capaz de realizar funciones básicas y determinar si es apto para continuar con pruebas más exhaustivas. [73]
- Pruebas de interfaz gráfica de usuario (GUI testing): Verifican que la interfaz de usuario cumple con los requisitos de diseño y usabilidad. [69]
- Pruebas de regresión (Regression testing): Verifican que las modificaciones realizadas en el software no introducen nuevos errores en el funcionamiento de componentes ya existentes. [74]
- Pruebas de carga (Load testing): Pruebas destinadas a simular situaciones de alta demanda de procesamiento y verificar que el software sigue funcionando adecuadamente con esas cargas. [74]

Pruebas que combinan los métodos:

- Pruebas de corrección (Correctness testing): Incluyen diversas técnicas de pruebas, como pruebas de caja negra (Black-box testing), pruebas de caja blanca (White-box testing) y pruebas estáticas (Static testing) con el objetivo de detectar defectos en el software. [74]
- Pruebas de automatización (Test Automation): Técnica que permite automatizar los procesos de pruebas para reducir el tiempo de prueba y aumentar la eficiencia del proceso de pruebas. [[37], [37]]
- Pruebas de usabilidad (Usability testing): Técnica de pruebas enfocada en verificar la facilidad de uso del software para los usuarios finales. [30]
- Pruebas de seguridad (Security testing): Técnica de pruebas destinada a verificar que el software es resistente a ataques malintencionados y vulnerabilidades. [9]

En relación con los niveles de pruebas de software, los artículos consultados mencionan los siguientes niveles de pruebas de software [64],[72],[58],[48],[33]:

- Pruebas unitarias: Se centran en la menor unidad de software, como funciones o métodos, para asegurarse de que funcionan correctamente de manera aislada.
- Pruebas de integración: Verifican que los módulos o componentes del software funcionan correctamente cuando se combinan o integran.
- Pruebas de sistema: Evalúan el comportamiento y las capacidades del sistema completo para asegurarse de que cumple con los requisitos especificados.

- Pruebas de aceptación del usuario (User Acceptance Testing, UAT): Pruebas realizadas por usuarios finales para verificar que el software cumple con sus necesidades y expectativas.

En conclusión, mediante el empleo de estos métodos y técnicas, las pruebas de software sirven como un componente crítico en el desarrollo de software de alta calidad, ayudando a minimizar los riesgos y a mejorar la satisfacción del usuario [7].

3.5.4.4 Pregunta 4:

¿Los casos de estudio encontrados se aplican en entornos académicos y/o industriales?

Mediante esta pregunta se pretende determinar la relevancia y aplicabilidad de los casos de estudio identificados en diferentes contextos. Podrían ser ámbitos académicos o industriales.

Los métodos de pruebas de software mencionados en los documentos tienen aplicaciones tanto en entornos académicos como industriales. Algunos de ellos, como las pruebas unitarias, de aceptación y de usabilidad, son empleados tanto en entornos académicos como en proyectos industriales. Por otro lado, otros métodos como las pruebas de caja negra, de regresión, de carga, de corrección y de automatización son utilizados principalmente en el ámbito de la industria de desarrollo de software. [64],[37],[35],[74],[28],[47],[41]

En los artículos se mencionan una variedad de aplicaciones y contextos en los que se utilizan técnicas y metodologías de pruebas de software. Algunos de los contextos más comunes incluyen:

- Industria de software: la mayoría de los estudios y documentos se aplican en el ámbito de la industria de software, en diferentes sectores como la banca, la salud, el comercio, entre otros.
- Entornos académicos: se han llevado a cabo varios estudios en entornos académicos, como en cursos de ingeniería de software y en investigaciones en universidades.

3.5.5 Amenazas de la validez

Al abordar las amenazas a la validez, una preocupación importante es la rigurosidad del proceso de revisión bibliográfica, por tanto, cabe señalar que:

- No se realizó una búsqueda sistemática de referencias dentro de los artículos filtrados. Esta ausencia de una búsqueda exhaustiva de referencias puede dar lugar a un descuido de los estudios pertinentes y posiblemente a un sesgo en la síntesis bibliográfica y las conclusiones extraídas.
- Es prudente afirmar que la búsqueda inicial basada en cadenas que arrojó una gran cantidad de artículos puede, en efecto, haber llevado a la exclusión inadvertida de estudios significativos, especialmente relevantes para la revisión sistemática metodológica.

- Una búsqueda ampliada en bases de datos adicionales podría reducir potencialmente el riesgo de pasar por alto bibliografía pertinente, aumentando así la solidez de las conclusiones del estudio.

Amenazas a la validez externa:

- Los enfoques de las pruebas pueden variar según la metodología de desarrollo elegida, en general, las pruebas de software buscan garantizar que el producto final cumpla con los requisitos del usuario y pueda funcionar en diferentes entornos.

Amenazas a la validez interna:

- La metodología de desarrollo de software utilizada y su impacto en las pruebas de software. Es importante considerar cómo las diferencias en las metodologías de desarrollo pueden influir en la ejecución de las pruebas y en la interpretación de los resultados. Además, la variabilidad en la estructuración y separación de las pruebas en fases específicas, como pruebas unitarias, de integración y de sistema, puede plantear desafíos para la validez interna al afectar la consistencia y fiabilidad de los resultados obtenidos en el estudio.

Amenazas a la validez del constructo:

- Determinada por la variabilidad en la interpretación y medición de los conceptos clave utilizados en las pruebas de software. Es importante considerar cómo la definición y operacionalización de los constructos, como usabilidad, compatibilidad, confiabilidad, integridad, eficiencia, seguridad, capacidad, portabilidad y mantenibilidad, pueden influir en los resultados y conclusiones del estudio. La falta de claridad en la definición de estos constructos o la inconsistencia en su medición pueden plantear desafíos para la validez del constructo en la investigación sobre pruebas de software.

La validez de la presente investigación podría mejorarse asegurándose de que se realiza una búsqueda sistemática y exhaustiva, que posiblemente abarque una gama más amplia de bases de datos. Esta ampliación permite incluir un conjunto más diverso de estudios, que pueden ofrecer una perspectiva más amplia y una base de pruebas más sustancial para los resultados de la investigación.

3.5.6 Lecciones aprendidas

A partir de los documentos consultados, se puede afirmar que las pruebas de software son un elemento clave en el desarrollo de software y su importancia ha ido en aumento en los últimos años. Las pruebas de software se aplican en diferentes contextos, tanto en entornos académicos como industriales, y son esenciales para crear software de alta calidad y mejorar la confiabilidad del mismo.

Algunas lecciones aprendidas de los documentos incluyen:

- Las pruebas de software deben comenzar lo más temprano posible en el ciclo de vida del desarrollo de software y deben realizarse continuamente a lo largo de todo el proceso [32].
- Las pruebas de software deben verificar y validar que el software realiza las funciones para las que fue diseñado, asegurando que la implementación sea consistente con las especificaciones [32].

- La automatización de las pruebas de software es clave para su éxito en entornos ágiles, ya que permite acelerar el proceso de aseguramiento de la calidad, documentar el software y reducir costos [37].
- El conocimiento de las pruebas de software es esencial para el desarrollo de software confiable y de alta calidad [76].
- La educación en pruebas de software es crucial para satisfacer la demanda de graduados con una sólida formación en pruebas [76].
- Los programas que incluyen proyectos prácticos de pruebas de software en la educación han demostrado aumentar significativamente el entusiasmo y el conocimiento de los estudiantes sobre la disciplina, así como su valor [41].

En general, se puede concluir que las pruebas de software son un elemento fundamental para el desarrollo de software confiable y de alta calidad, y su importancia seguirá aumentando a medida que los sistemas de información se vuelven más complejos. Por lo tanto, es importante que tanto en entornos académicos como industriales se dé prioridad a la educación y la utilización de herramientas y técnicas de pruebas de software para garantizar la calidad del mismo.

CAPÍTULO IV: Conclusiones y Trabajo futuro

4.1 Conclusiones

Los artículos consultados enfatizan la importancia de las pruebas de software en el desarrollo de aplicaciones confiables y de alta calidad.

Las pruebas de software son esenciales para garantizar que los programas funcionen según lo previsto y que se ajusten a las expectativas de los usuarios. Además, las pruebas ayudan a identificar y corregir errores de forma temprana, lo que reduce los costos y aumenta la satisfacción del usuario.

Los documentos mencionan diferentes metodologías, herramientas y técnicas utilizadas en las pruebas de software, incluyendo la automatización de pruebas, la verificación y validación, la generación de informes de resultados, la realización de pruebas de regresión y la gestión de errores.

También se destaca la importancia de la educación en pruebas de software, tanto en entornos académicos como en la industria, para crear una base sólida de ingenieros de software que estén capacitados para diseñar y ejecutar pruebas de software efectivas.

En cuanto a las métricas en el ámbito del análisis de fallos, modos y efectos, contribuyen significativamente al proceso de aseguramiento de la calidad al identificar la gravedad y la probabilidad de fallos potenciales que pueden requerir acciones correctivas dentro del proyecto. Esta faceta de las métricas es indispensable, sobre todo en aplicaciones críticas como la aeroespacial, donde la fiabilidad y la seguridad son primordiales.

En general, se puede concluir que las pruebas de software son cruciales para garantizar que las aplicaciones cumplan con los requisitos de los usuarios y los estándares de calidad. Los ingenieros de software deben seguir los principios y las mejores prácticas para diseñar y ejecutar pruebas efectivas y optimizar los recursos disponibles.

4.2 Recomendaciones y Trabajo Futuro

Las recomendaciones generales relacionadas con las pruebas de software:

- Incorporar pruebas tempranas y continuas en el ciclo de vida del software: Las pruebas de software deben comenzar lo más temprano posible en el ciclo de vida del desarrollo de software y deben realizarse de manera continua a lo largo de todo el proceso. Esto ayuda a identificar y corregir errores de manera temprana y reduce los costos de corrección.
- Enfatizar la educación en pruebas de software: Los currículos académicos en ingeniería de software deberían poner más enfoque en la educación de pruebas de software para satisfacer la demanda de graduados con una sólida formación en pruebas. Además, se recomienda que los profesionales de la industria también se mantengan al día en las últimas metodologías, técnicas y herramientas de pruebas de software.
- Utilizar herramientas y técnicas de automatización: La automatización de las pruebas de software puede aumentar la eficiencia y la capacidad de realizar pruebas de regresión frecuentes, especialmente en entornos de desarrollo

iterativo incremental. Esto también ayuda a documentar el software y reducir costos.

- Problemática de la comunicación en la gestión de pruebas: Es importante prestar atención a la comunicación en la gestión de pruebas, incluyendo la comunicación entre los equipos de desarrollo y de pruebas, la selección de herramientas de apoyo a pruebas y la definición de estándares y mejores prácticas de pruebas.
- Considerar pruebas de seguridad: Las pruebas de seguridad son importantes para detectar vulnerabilidades y garantizar la seguridad del software en diferentes entornos.
- Realizar pruebas en múltiples plataformas y sistemas operativos: Es importante probar el software en múltiples plataformas y sistemas operativos para garantizar su funcionalidad en diferentes entornos.
- En general, las pruebas de software son fundamentales para garantizar la calidad y la eficacia del software, y es esencial que los ingenieros de software se mantengan actualizados en las últimas metodologías, técnicas y herramientas de pruebas de software para garantizar su efectividad.
- A partir de los documentos consultados, se pueden identificar algunos temas y áreas de interés para futuras investigaciones en el campo de las pruebas de software.

Algunos posibles trabajos futuros incluyen:

- Uso de inteligencia artificial en pruebas de software: La utilización de técnicas de inteligencia artificial en las pruebas de software puede ayudar a mejorar la velocidad y la eficacia de las pruebas, así como a detectar problemas más complejos y difíciles de identificar manualmente.
- Mejoras en la calidad de la educación en pruebas de software: A pesar de la importancia de las pruebas de software, la educación formal en esta área a menudo se pasa por alto. Los futuros trabajos podrían centrarse en mejorar la capacidad de los programas de ingeniería de software para impartir una formación en pruebas de software adecuada y adecuada para satisfacer las necesidades de una industria en constante evolución.
- Aplicación de pruebas de seguridad en la internet de las cosas (IoT): La creciente presencia de dispositivos IoT en nuestra vida cotidiana presenta nuevos desafíos en términos de seguridad y privacidad. Los futuros trabajos podrían centrarse en el desarrollo de técnicas de pruebas de software altamente especializadas y efectivas para la detección de vulnerabilidades en los dispositivos IoT.
- Mejora en la gestión de pruebas de software: Una parte importante del éxito de las pruebas de software es la gestión responsable de los recursos. Los futuros trabajos podrían abordar la mejora y simplificación del proceso de planificación, diseño y ejecución de pruebas para aumentar la eficacia y reducir los costos.
- Evaluación del valor de pruebas en el contexto del software de código abierto: Con la popularidad creciente del software de código abierto, se necesitan más estudios sobre la efectividad y el valor de las pruebas de software en este contexto particular. Los trabajos futuros podrían investigar la relación entre el nivel de pruebas y la calidad del software de código abierto.

En resumen, hay múltiples áreas donde se puede seguir investigando para mejorar la efectividad y eficiencia de las pruebas de software, y para adaptarse a los cambios y desafíos continuos de la industria de software.

CAPITULO V: Referencias

[1] D. Shao, S. Khurshid, y D. E. Perry, "A Case for White-box Testing Using Declarative Specifications Poster Abstract", en *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*, Windsor, UK: IEEE, sep. 2007, pp. 137–137. doi: 10.1109/TAIC.PART.2007.36.

[2] G. J. Myers, C. Sandler, y T. Badgett, *The art of software testing*, 3rd ed. Hoboken, N.J: John Wiley & Sons, 2012.

[3] M. Sharma y S. C. B., "Automatic Generation of Test Suites from Decision Table - Theory and Implementation", en *2010 Fifth International Conference on Software Engineering Advances*, Nice: IEEE, ago. 2010, pp. 459–464. doi: 10.1109/ICSEA.2010.78.

[4] L. Leopoldo Pauta Ayabaca y S. Moscoso Bernal, "Verificación y validación de software", *tecnica*, vol. 1, núm. 3, p. 25, feb. 2018, doi: 10.26871/killkana_tecnica.v1i3.112.

[5] E. Serna M. y F. Arango I., "Prueba del software: más que una fase en el ciclo de vida", *Revista de Ingeniería*, núm. 35, pp. 34–40, jul. 2011, doi: 10.16924/revinge.35.5.

[6] F. Elberzhager, J. Münch, y V. T. N. Nha, "A systematic mapping study on the combination of static and dynamic quality assurance techniques", *Information and Software Technology*, vol. 54, núm. 1, pp. 1–15, ene. 2012, doi: 10.1016/j.infsof.2011.06.003.

[7] "IEEE Standard Glossary of Software Engineering Terminology", IEEE. doi: 10.1109/IEEESTD.1990.101064.

[8] R. Bierig, S. Brown, E. Galván, y J. Timoney, *Essentials of Software Testing*, 1a ed. Cambridge University Press, 2021. doi: 10.1017/9781108974073.

[9] P. Kamthan, "On Conducting Tests in Software Engineering Courses during the COVID-19 Pandemic (S)", presentado en The 33rd International Conference on Software Engineering and Knowledge Engineering, jul. 2021, pp. 365–368. doi: 10.18293/SEKE2021-025.

[10] M. A. Aucancela Guamán, "La usabilidad en los sistemas de inteligencia de negocios, un caso práctico", *CD*, vol. 3, núm. 3.3, pp. 319–330, sep. 2019, doi: 10.33262/cienciadigital.v3i3.3.824.

[11] M. D. Delgado Dapena, A. Macías Rojas, D. Larrosa Uribazo, S. Verona Marcos, y P. B. Fernández Oliva, "Model for Automatic Generation of Search-Based Early Tests", *CyS*, vol. 21, núm. 3, sep. 2017, doi: 10.13053/cys-21-3-2716.

[12] T. Ostrand, "Black-Box Testing", en *Encyclopedia of Software Engineering*, 1a ed., J. J. Marciniak, Ed., Wiley, 2002. doi: 10.1002/0471028959.sof022.

[13] Q. Luo, K. Moran, y D. Poshyvanyk, "A large-scale empirical comparison of static and dynamic test case prioritization techniques", en *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Seattle WA USA: ACM, nov. 2016, pp. 559–570. doi: 10.1145/2950290.2950344.

[14] S. Lee *et al.*, "A comprehensive security assessment framework for software-defined networks", *Computers & Security*, vol. 91, p. 101720, abr. 2020, doi: 10.1016/j.cose.2020.101720.

[15] P. Arapa Carcasi, "Diseño y análisis de un software para formulación de mezclas alimenticias a base de cultivos andinos", *RECIA*, vol. 7, núm. 1, pp. 30–41, dic. 2019, doi: 10.23850/24220582.2571.

[16] F. Zorzan *et al.*, "Delayed completion of Final Project of the career Computer Analyst: Seeking its causes", en *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*, Medellin, Colombia: IEEE, oct. 2012, pp. 1–8. doi: 10.1109/CLEI.2012.6427138.

[17] P. Bourque y R. E. Fairley, Eds., *SWEBOK: guide to the software engineering body of knowledge*, Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014.

[18] B. A. Kitchenham, "Systematic review in software engineering: where we are and where we should be going", en *Proceedings of the 2nd international workshop on Evidential assessment of software technologies*, Lund Sweden: ACM, sep. 2012, pp. 1–2. doi: 10.1145/2372233.2372235.

[19] A. Beelmann, “Review of Systematic reviews in the social sciences”, *European Psychologist*, vol. 11, núm. 3, pp. 244–245, ene. 2006, doi: 10.1027/1016-9040.11.3.244.
[20] A. Ahmad, P. Jamshidi, y C. Pahl, “Classification and comparison of architecture evolution reuse knowledge—a systematic review”, *J Software Evolu Process*, vol. 26, núm. 7, pp. 654–691, jul. 2014, doi: 10.1002/smr.1643.
[21] A. V. Papadopoulos *et al.*, “Methodological Principles for Reproducible Performance Evaluation in Cloud Computing”, *IIEEE Trans. Software Eng.*, vol. 47, núm. 8, pp. 1528–1543, ago. 2021, doi: 10.1109/TSE.2019.2927908.
[22] M. Zarour, A. Abran, J.-M. Desharnais, y A. Alarifi, “An investigation into the best practices for the successful design and implementation of lightweight software process assessment methods: A systematic literature review”, *Journal of Systems and Software*, vol. 101, pp. 180–192, mar. 2015, doi: 10.1016/j.jss.2014.11.041.
[23] M. Dyer, “A formal approach to software error removal”, *Journal of Systems and Software*, vol. 7, núm. 2, pp. 109–114, jun. 1987, doi: 10.1016/0164-1212(87)90015-X.
[24] A. J. Chruscicki y J. E. Gaffney, “Toward a quality inspection method and management”, en *Proceedings of Software Engineering Standards Symposium*, Montreal, Que., Canada: IEEE Comput. Soc. Press, 1995, pp. 71–78. doi: 10.1109/SESS.1995.525953.
[25] D. W. Sova y C. Smidts, “Increasing testing productivity and software quality: A comparison of software testing methodologies within NASA”, *Empirical Software Engineering*, vol. 1, núm. 2, pp. 165–188, 1996, doi: 10.1007/BF00368703.
[26] J. E. Heiser, “An overview of software testing”, en *1997 IEEE Autotestcon Proceedings AUTOTESTCON ’97. IEEE Systems Readiness Technology Conference. Systems Readiness Supporting Global Needs and Awareness in the 21st Century*, Anaheim, CA, USA: IEEE, 1997, pp. 204–211. doi: 10.1109/AUTEST.1997.633613.
[27] Tien-Fu Chang *et al.*, “‘Continuous verification’ in mission critical software development”, en *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, Wailea, HI, USA: IEEE Comput. Soc. Press, 1997, pp. 273–284. doi: 10.1109/HICSS.1997.663184.
[28] G. J. Holzmann y M. H. Smith, “A practical method for verifying event-driven software”, en *Proceedings of the 21st international conference on Software engineering*, Los Angeles California USA: ACM, may 1999, pp. 597–607. doi: 10.1145/302405.302710.
[29] D. P. Gilliam, J. C. Kelly, J. D. Powell, y M. Bishop, “Development of a software security assessment instrument to reduce software security risk”, en *Proceedings Tenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2001*, Cambridge, MA, USA: IEEE Comput. Soc, 2001, pp. 144–149. doi: 10.1109/ENABL.2001.953404.
[30] M. Butcher, H. Munro, y T. Kratschmer, “Improving software testing via ODC: Three case studies”, *IBM Syst. J.*, vol. 41, núm. 1, pp. 31–44, 2002, doi: 10.1147/sj.411.0031.
[31] O. Mizuno, E. Shigematsu, Y. Takagi, y T. Kikuno, “On estimating testing effort needed to assure field quality in software development”, en *13th International Symposium on Software Reliability Engineering, 2002. Proceedings.*, Annapolis, MD, USA: IEEE Comput. Soc, 2002, pp. 139–146. doi: 10.1109/ISSRE.2002.1173234.
[32] B. Rumpe, “Model-Based Testing of Object-Oriented Systems”, en *Formal Methods for Components and Objects*, vol. 2852, F. S. De Boer, M. M. Bonsangue, S. Graf, y W.-P. De Roever, Eds., en Lecture Notes in Computer Science, vol. 2852. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 380–402. doi: 10.1007/978-3-540-39656-7_16.
[33] L. Williams, E. M. Maximilien, y M. Vouk, “Test-driven development as a defect-reduction practice”, en *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, Denver, Colorado, USA: IEEE, 2003, pp. 34–45. doi: 10.1109/ISSRE.2003.1251029.
[34] T. Berling y T. Thelin, “An industrial case study of the verification and validation activities”, en *Proceedings. 5th International Workshop on Enterprise Networking and Computing in*

Healthcare Industry (IEEE Cat. No.03EX717), Sydney, NSW, Australia: IEEE Comput. Soc, 2003, pp. 226–238. doi: 10.1109/METRIC.2003.1232470.

[35] M. Karlesky, W. Bereza, y C. Erickson, "Effective Test Driven Development for Embedded Software", en *2006 IEEE International Conference on Electro/Information Technology*, East Lansing, MI, USA: IEEE, may 2006, pp. 382–387. doi: 10.1109/EIT.2006.252188.

[36] M. Ellims, J. Bridges, y D. C. Ince, "The Economics of Unit Testing", *Empir Software Eng*, vol. 11, núm. 1, pp. 5–31, mar. 2006, doi: 10.1007/s10664-006-5964-9.

[37] D. S. Janzen y H. Saiedian, "On the Influence of Test-Driven Development on Software Design", en *19th Conference on Software Engineering Education & Training (CSEET'06)*, Turtle Bay, HI, USA: IEEE, 2006, pp. 141–148. doi: 10.1109/CSEET.2006.25.

[38] E. Mnkandla y B. Dwolatzky, "Defining Agile Software Quality Assurance", en *2006 International Conference on Software Engineering Advances (ICSEA'06)*, Tahiti: IEEE, oct. 2006, pp. 36–36. doi: 10.1109/ICSEA.2006.261292.

[39] D. Loubach, J. S. Nobre, A. Da Cunha, L. V. Dias, M. Nascimento, y W. Dos Santos, "Testing Critical Software: A Case Study for an Aerospace Application", en *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*, Portland, OR, USA: IEEE, oct. 2006, pp. 1–9. doi: 10.1109/DASC.2006.313741.

[40] M. Siniaalto y P. Abrahamsson, "Does Test-Driven Development Improve the Program Code? Alarming Results from a Comparative Case Study", en *Balancing Agility and Formalism in Software Engineering*, vol. 5082, B. Meyer, J. R. Nawrocki, y B. Walter, Eds., en Lecture Notes in Computer Science, vol. 5082. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 143–156. doi: 10.1007/978-3-540-85279-7_12.

[41] Y. Jiang, S.-S. Hou, J.-H. Shan, L. Zhang, y B. Xie, "AN APPROACH TO TESTING BLACK-BOX COMPONENTS USING CONTRACT-BASED MUTATION", *Int. J. Soft. Eng. Knowl. Eng.*, vol. 18, núm. 01, pp. 93–117, feb. 2008, doi: 10.1142/S0218194008003556.

[42] R. P. Tan y S. Edwards, "Evaluating Automated Unit Testing in Sulu", en *2008 International Conference on Software Testing, Verification, and Validation*, Lillehammer, Norway: IEEE, abr. 2008, pp. 62–71. doi: 10.1109/ICST.2008.59.

[43] J. Watkins, *Agile testing: how to succeed in an extreme testing environment*. Cambridge: Cambridge University Press, 2009.

[44] X. He, J. Guo, Y. Wang, y Y. Guo, "An Automatic Compliance Checking Approach for Software Processes", en *2009 16th Asia-Pacific Software Engineering Conference*, Batu Ferringhi, Penang, Malaysia: IEEE, dic. 2009, pp. 467–474. doi: 10.1109/APSEC.2009.48.

[45] C. D. Thomson, M. Holcombe, y A. J. H. Simons, "What Makes Testing Work: Nine Case Studies of Software Development Teams", en *2009 Testing: Academic and Industrial Conference - Practice and Research Techniques*, Windsor, United Kingdom: IEEE, 2009, pp. 167–175. doi: 10.1109/TAICPART.2009.12.

[46] A. C. D. Neto, R. De Freitas Rodrigues, y G. H. Travassos, "Porantim-Opt: Optimizing the Combined Selection of Model-Based Testing Techniques", en *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, Berlin, Germany: IEEE, mar. 2011, pp. 174–183. doi: 10.1109/ICSTW.2011.33.

[47] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, y A. Teterev, "CRANE: Failure Prediction, Change Analysis and Test Prioritization in Practice -- Experiences from Windows", en *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, Berlin, Germany: IEEE, mar. 2011, pp. 357–366. doi: 10.1109/ICST.2011.24.

[48] M. Fischer y R. Tonjes, "Generating test data for black-box testing using genetic algorithms", en *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, Krakow, Poland: IEEE, sep. 2012, pp. 1–6. doi: 10.1109/ETFA.2012.6489789.

[49] E. F. Collins y V. F. De Lucena, "Software Test Automation practices in agile development environment: An industry experience report", en *2012 7th International Workshop on*

Automation of Software Test (AST), Zurich, Switzerland: IEEE, jun. 2012, pp. 57–63. doi: 10.1109/IWAST.2012.6228991.

[50] I. Saleh y K. Nagi, "HadoopMutator: A Cloud-Based Mutation Testing Framework", en *Software Reuse for Dynamic Systems in the Cloud and Beyond*, vol. 8919, I. Schaefer y I. Stamelos, Eds., en Lecture Notes in Computer Science, vol. 8919. , Cham: Springer International Publishing, 2014, pp. 172–187. doi: 10.1007/978-3-319-14130-5_13.

[51] P. Day, "n-Tiered Test Automation Architecture for Agile Software Systems", *Procedia Computer Science*, vol. 28, pp. 332–339, 2014, doi: 10.1016/j.procs.2014.03.041.

[52] H. Aljumaily, D. Cuadra, y P. Martínez, "Applying black-box testing to UML/OCL database models", *Software Qual J*, vol. 22, núm. 2, pp. 153–184, jun. 2014, doi: 10.1007/s11219-012-9192-9.

[53] M. Sadiq y S. Sultana, "A Method for the Selection of Software Testing Techniques Using Analytic Hierarchy Process", en *Computational Intelligence in Data Mining - Volume 1*, vol. 31, L. C. Jain, H. S. Behera, J. K. Mandal, y D. P. Mohapatra, Eds., en Smart Innovation, Systems and Technologies, vol. 31. , New Delhi: Springer India, 2015, pp. 213–220. doi: 10.1007/978-81-322-2205-7_20.

[54] R. Poston y A. Calvert, "Vision 2020: The Future of Software Quality Management and Impacts on Global User Acceptance", en *HCI in Business*, vol. 9191, F. Fui-Hoon Nah y C.-H. Tan, Eds., en Lecture Notes in Computer Science, vol. 9191. , Cham: Springer International Publishing, 2015, pp. 748–760. doi: 10.1007/978-3-319-20895-4_70.

[55] H. Hemmati, M. Nagappan, y A. E. Hassan, "Investigating the effect of 'defect co-fix' on quality assurance resource allocation: A search-based approach", *Journal of Systems and Software*, vol. 103, pp. 412–422, may 2015, doi: 10.1016/j.jss.2014.11.040.

[56] S. K. Khatri, K. Kaur, y R. K. Datta, "Using Statistical Usage Testing in Conjunction with Other Black Box Testing Techniques", *Int. J. Rel. Qual. Saf. Eng.*, vol. 22, núm. 01, p. 1550004, feb. 2015, doi: 10.1142/S0218539315500047.

[57] J. Chandrasekaran, L. Sh. Ghandehari, Y. Lei, R. Kacker, y D. R. Kuhn, "Evaluating the Effectiveness of BEN in Localizing Different Types of Software Fault", en *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Chicago, IL, USA: IEEE, abr. 2016, pp. 26–34. doi: 10.1109/ICSTW.2016.44.

[58] A. Arcuri, J. Campos, y G. Fraser, "Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins", en *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Chicago, IL, USA: IEEE, abr. 2016, pp. 401–408. doi: 10.1109/ICST.2016.44.

[59] H. J. Herpel *et al.*, "Model based testing of satellite on-board software — An industrial use case", en *2016 IEEE Aerospace Conference*, Big Sky, MT: IEEE, mar. 2016, pp. 1–9. doi: 10.1109/AERO.2016.7500845.

[60] M. Atifi, A. Mamouni, y A. Marzak, "A Comparative Study of Software Testing Techniques", en *Networked Systems*, vol. 10299, A. El Abbadi y B. Garbinato, Eds., en Lecture Notes in Computer Science, vol. 10299. , Cham: Springer International Publishing, 2017, pp. 373–390. doi: 10.1007/978-3-319-59647-1_27.

[61] M. F. Granda, N. Condori-Fernández, T. E. J. Vos, y O. Pastor, "Effectiveness Assessment of an Early Testing Technique using Model-Level Mutants", en *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, Karlskrona Sweden: ACM, jun. 2017, pp. 98–107. doi: 10.1145/3084226.3084257.

[62] P. Rahayu, D. I. Sensuse, W. R. Fitriani, I. Nurrohmah, R. Mauliadi, y H. N. Rochman, "Applying usability testing to improving Scrum methodology in develop assistant information system", en *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, Bandung - Bali, Indonesia: IEEE, oct. 2017, pp. 1–6. doi: 10.1109/ICITSI.2016.7858222.

[63] R. H. Rosero, O. S. Gomez, y G. Rodriguez, “Regression Testing of Database Applications Under an Incremental Software Development Setting”, *IEEE Access*, vol. 5, pp. 18419–18428, 2017, doi: 10.1109/ACCESS.2017.2749502.

[64] M. Felderer y F. Auer, “Software Quality Assurance During Implementation: Results of a Survey in Software Houses from Germany, Austria and Switzerland”, en *Software Quality. Complexity and Challenges of Software Engineering in Emerging Technologies*, vol. 269, D. Winkler, S. Biffl, y J. Bergsmann, Eds., en Lecture Notes in Business Information Processing, vol. 269. , Cham: Springer International Publishing, 2017, pp. 87–102. doi: 10.1007/978-3-319-49421-0_7.

[65] A. Bajaj y O. P. Sangwan, “Aspect oriented software testing”, en *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, Noida, India: IEEE, ene. 2017, pp. 809–814. doi: 10.1109/CONFLUENCE.2017.7943261.

[66] B. N. Machado, C. G. Camilo-Junior, C. L. Rodrigues, y E. H. D. Quijano, “SBSTFrame: a framework to search-based software testing”, en *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Budapest, Hungary: IEEE, oct. 2017, pp. 004106–004111. doi: 10.1109/SMC.2016.7844875.

[67] B. Maqbool, F. U. Rehman, M. Abbas, y S. Rehman, “Implementation of Software Testing Practices in Pakistan’s Software Industry”, en *Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences*, Wuhan China: ACM, ene. 2018, pp. 147–152. doi: 10.1145/3180374.3181340.

[68] S. Vasanthapriyan, “A Study of Software Testing Practices in Sri Lankan Software Companies”, en *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Lisbon: IEEE, jul. 2018, pp. 339–344. doi: 10.1109/QRS-C.2018.00066.

[69] A. Nanthaamornphong y J. C. Carver, “Test-Driven Development in HPC Science: A Case Study”, *Comput. Sci. Eng.*, vol. 20, núm. 5, pp. 98–113, sep. 2018, doi: 10.1109/MCSE.2018.05329819.

[70] A. S. Dookhun y L. Nagowah, “Assessing The Effectiveness Of Test-Driven Development and Behavior-Driven Development in an Industry Setting”, en *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, Dubai, United Arab Emirates: IEEE, dic. 2019, pp. 365–370. doi: 10.1109/ICCIKE47802.2019.9004328.

[71] P. Mahanta y G. Bischoff, “Framework for Collaborative Software Testing Efforts Between Cross-Functional Teams Aiming at High Quality End Product”, en *On the Move to Meaningful Internet Systems: OTM 2018 Workshops*, vol. 11231, C. Debruyne, H. Panetto, W. Guédria, P. Bollen, I. Ciuciu, y R. Meersman, Eds., en Lecture Notes in Computer Science, vol. 11231. , Cham: Springer International Publishing, 2019, pp. 189–195. doi: 10.1007/978-3-030-11683-5_21.

[72] T. Saha y R. Palit, “Practices of Software Testing Techniques and Tools in Bangladesh Software Industry”, en *2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, Melbourne, Australia: IEEE, dic. 2019, pp. 1–10. doi: 10.1109/CSDE48274.2019.9162355.

[73] H. Zhong, L. Zhang, y S. Khurshid, “TestSage: Regression Test Selection for Large-Scale Web Service Testing”, en *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, Xi’an, China: IEEE, abr. 2019, pp. 430–440. doi: 10.1109/ICST.2019.00052.

[74] K. S. Arif y U. Ali, “Mobile Application testing tools and their challenges: A comparative study”, en *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Sukkur, Pakistan: IEEE, ene. 2019, pp. 1–6. doi: 10.1109/ICOMET.2019.8673505.

[75] Q.-T. Huynh, L.-T. Pham, N.-H. Ha, y D.-M. Nguyen, “An Effective Approach for Context Driven Testing in Practice — A Case Study”, *Int. J. Soft. Eng. Knowl. Eng.*, vol. 30, núm. 09, pp. 1245–1262, sep. 2020, doi: 10.1142/S0218194020500333.

[76] P. Delgado-Perez, I. Medina-Bulo, M. A. Alvarez-Garcia, y K. J. Valle-Gomez, “Mutation Testing and Self/Peer Assessment: Analyzing their Effect on Students in a Software Testing Course”, en *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Madrid, ES: IEEE, may 2021, pp. 231–240. doi: 10.1109/ICSE-SEET52601.2021.00033.