

Santiago de Cali, 20 de 06 de 2023

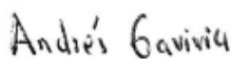
Ingeniero:

Juan Carlos Martínez Arias
Director Posgrados de Ingeniería
Facultad de Ingeniería y Ciencias
Pontificia Universidad Javeriana - Cali

Con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el Trabajo de Grado y posteriormente optar por el título de Magíster en Ingeniería de software, nos permitimos presentar a su consideración el anteproyecto de Trabajo de Grado denominado desarrollo de un prototipo de servicio para probar respuestas de diferentes ambientes en arquitecturas REST, el cual será realizado por el (la) estudiante Andrés Felipe Gaviria Ocampo con código 8973244 perteneciente al énfasis en pruebas de software, bajo la dirección del profesor Mario Julian Mora Cardona.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer la evaluación de este Anteproyecto ante el Tribunal que para el efecto se designe, toda vez que ha revisado cuidadosamente el documento y avala que ya se encuentra listo para ser presentado oficialmente.

Atentamente,



Firma
Andrés Felipe Gaviria Ocampo
C.C. 1144096389 de Cali



Firma
Mario Julian Mora Cardona
C.C. 94400220 de Cali



Firma
Jhon Haide Cano Beltran
C.C. 94531260 de Cali

Ficha Resumen

Anteproyecto de Trabajo de Grado

Desarrollo de un prototipo de servicio para probar respuestas de diferentes ambientes en arquitecturas REST.

1. Área de trabajo: Detección de errores (Prueba de calidad).
2. Tipo de proyecto: Aplicado
3. Estudiante: Andrés Felipe Gaviria Ocampo
4. Correo electrónico: fgaviriaocam@javerianacali.edu.co
5. Dirección y teléfono: Tulipanes del castillo casa 204, 3164394974
6. Director: Mario Julian Mora Cardona
7. Vinculación del director: Magister en ingeniería
8. Correo electrónico del director: mariomora@javerianacali.edu.co
9. Co-Director (Si aplica): Jhon Haide Cano Beltran
10. Grupo o empresa que lo avala (Si aplica):
11. Otros grupos o empresas:
12. Palabras clave(al menos 5): Pruebas, Calidad, Caja negra, regresión, comparador de respuestas.
13. Fecha de inicio: 24 de Julio de 2023
14. Duración estimada: 6 meses
15. Resumen: Desarrollo de un prototipo de servicio web que permita comparar las respuestas de diferentes ambientes antes de los despliegues productivos. El resultado esperado es reducir los errores inyectados en producción.



Desarrollo de un prototipo de servicio para probar respuestas de diferentes ambientes en arquitecturas REST

Andrés Felipe Gaviria Ocampo

Anteproyecto presentada(o) como requisito parcial para optar al título de:
Magister en Ingeniería de Software

Director(a):
MSc. Mario Julian Mora Cardona

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería

Departamento de Electrónica y Ciencias de la Computación
Cali, Colombia
21 de junio de 2023

Quedó
corrido
fuera de la pág.

Índice

1. Introducción	9
2. Definición del problema	10
2.1. Planteamiento del problema	10
2.2. Formulación del problema	12
3. Objetivos del proyecto	13
3.1. Objetivo General	13
3.2. Objetivos específicos	13
3.3. Resultados esperados	13
4. Alcance	14
5. Justificación del trabajo de grado	15
6. Marco teórico de referencia y antecedentes	16
6.1. Bases Teóricas	16
6.2. Estado del Arte	28
7. Metodología de la investigación	32
8. Recursos a emplear	33
9. Cronograma de actividades	34
10. Referencias Bibliográficas	35
11. Glosario de Términos	38

Índice de figuras

1. Costo de arreglar un error en cada fase.	11
---	----

2.	A model of the software testing process	18
----	---	----

Resumen

Los servicios web basados en el arquitectura REST (Representational State Transfer) están ganando popularidad en el ámbito empresarial. Cada empresa tiene un núcleo de negocio basado en los servicios web, la cual son vitales para la ejecución de las operaciones comerciales. La descomposición del negocio en microservicios ha traído la dificultad para evaluarlas, es crucial garantizar la calidad del código y proporcionar un nivel de asegurabilidad que proteja al negocio de posibles errores.

En este contexto, la propuesta de investigación de este proyecto tiene como propósito desarrollar un prototipo de servicio que mejore la observabilidad de posibles errores en el comportamiento del código antes de su implementación en un entorno productivo. Este servicio no reemplazará otras pruebas existentes en la organización, sino que se integrará como un paso adicional en el proceso de aseguramiento de calidad.

Palabras Clave Pruebas de caja negra, comparador, calidad de código, Rest, deployment, protocolo, unit test, pruebas de regresión.

Abstract

Web services based on the REST (Representational State Transfer) architecture are gaining popularity in the business environment. Each company has a core business based on web services, which are vital for carrying out business operations. The decomposition of the business into microservices has brought the challenge of evaluating them. It is crucial to ensure code quality and provide a level of assurance that protects the business from potential errors.

In this context, the research proposal of this project aims to develop a prototype service that enhances the observability of potential errors in code behavior before its implementation in a production environment. This service will not replace other existing tests in the organization but will be integrated as an additional step in the quality assurance process.

Keywords Black box testing, comparator, code quality, REST, deployment, protocol, unit test, regression testing.

1. Introducción

Los servicios REST (Representational State Transfer) y los microservicios son estilos de arquitectura que resuelven problemas específicos. Estos estilos permiten intercambiar mensajes a través de HTTP utilizando una estructura JSON, mediante puntos de entrada conocidos como endpoints. El uso de HTTP implica la gestión de diferentes verbos, como GET, POST, DELETE y PUT, lo que permite especificar el tipo de operación a realizar. Estas arquitecturas están diseñadas para ofrecer flexibilidad en las solicitudes de entrada. No obstante, esta flexibilidad también puede dar lugar a entradas inesperadas, lo que a su vez puede provocar fallas si el desarrollador no ha tenido en cuenta estos casos. En muchos aspectos, estas fallas no se detectan durante el análisis, las inspecciones de código o las pruebas unitarias. Aunque se permita flexibilidad en las entradas del servicio o microservicio, no es aceptable que el servidor genere errores o se caiga cuando se enfrenta a situaciones inesperadas.

En el desarrollo de servicios y microservicios, la garantía de calidad y la asegurabilidad del código son fundamentales para su implementación en el entorno productivo. Para lograr esto, se utilizan herramientas especializadas que permiten identificar posibles fallos dentro de un sistema. Además, existen equipos expertos en la generación de pruebas para las diferentes partes un sistema, quienes llevan a cabo pruebas exhaustivas en un entorno controlado que emula las condiciones del entorno real. En este proceso, se consideran varios factores, como la utilización de datos realistas, el tamaño de las máquinas, las configuraciones, los usuarios y el tráfico.

Sin embargo, a pesar de todos estos esfuerzos para garantizar la uniformidad, no se puede demostrar una igualdad absoluta entre los entornos, ya que es natural que existan pequeñas diferencias entre ellos. Estas diferencias pueden deberse a diversos factores, como las configuraciones específicas de cada ambiente, las variaciones en la carga de trabajo o incluso la infraestructura subyacente utilizada. Por lo tanto, la necesidad de contar con un nuevo instrumento que permita ~~aumentar~~ la disminución de errores en despliegues productivos se vuelve tan relevante dentro de las organizaciones del campo IT.

> Redundante : "disminución de errores es suficiente."

2. Definición del problema

2.1. Planteamiento del problema

En el desarrollo de software, es común contar con diversos entornos, como el ambiente productivo, el ambiente de pruebas y posiblemente otros ambientes, que se utilizan para probar y desplegar aplicaciones. Sin embargo, durante este proceso, pueden surgir errores que se introducen en el código y pasan desapercibidos hasta que se manifiestan en el ambiente productivo. Estos errores pueden generar problemas significativos y afectar la calidad del software en general. De acuerdo con (Mendes et al.) las fallas que se presentan en aplicativos web afectan el rendimiento y la confiabilidad.

En la figura 1, se muestra un gráfico de barras que ilustra el costo de arreglar un error en cada ciclo de desarrollo de software, en el caso de producción puede tener un alto costo, tanto en términos económicos como en la reputación de la empresa. En el artículo publicado por la Nasa (Stecklein et al.), titulado “Error Cost Escalation Through the Project Life Cycle”, se aborda una realidad común en el desarrollo de software. A medida que un proyecto avanza, los errores no detectados o no abordados a tiempo pueden convertirse en desafíos costosos. Esto se debe a factores como la detección tardía, el impacto en múltiples componentes, las complejas dependencias y la necesidad de reestructuración. Para mitigar este problema, es crucial implementar estrategias como la detección temprana de errores y pruebas exhaustivas en diferentes etapas del ciclo de vida del proyecto. Estas prácticas ayudan a minimizar los costos asociados con los errores y aseguran la calidad del software desarrollado.

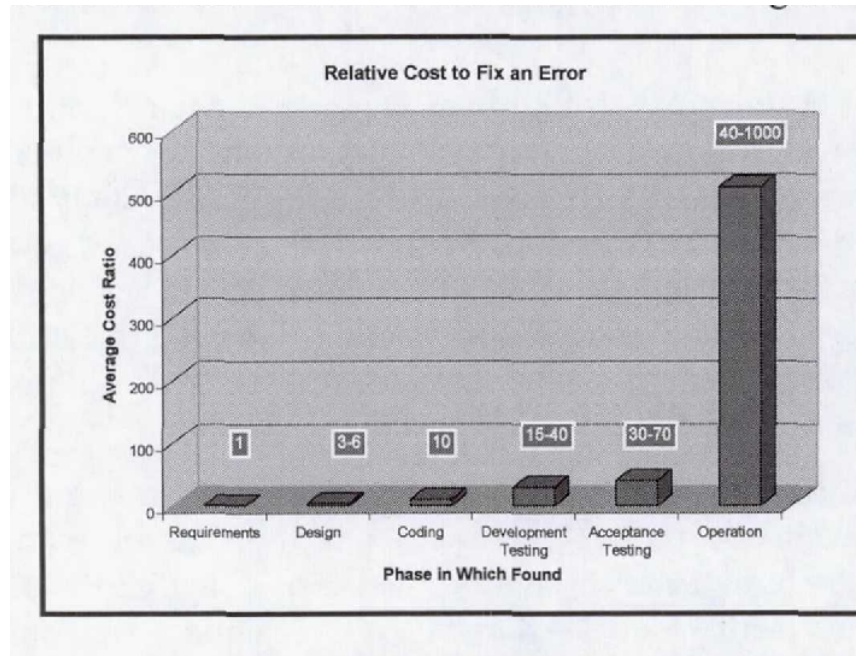


Figura 1: Costo de arreglar un error en cada fase.
(Stecklein et al.)

En (Laranjeiro et al.) menciona la herramienta RapiTest, la cual se destaca por su enfoque en pruebas de caja negra en APIs web RESTful. Esta herramienta permite realizar pruebas automatizadas sin necesidad de acceder al código fuente subyacente de la API. Utiliza técnicas avanzadas de generación de datos y cobertura exhaustiva para garantizar pruebas rigurosas en diversos escenarios. Es importante destacar que todas las pruebas son configurables y se basan en definiciones proporcionadas a través de OpenAPI. Sin embargo, es necesario tener en cuenta que el uso de datos no productivos para las pruebas puede afectar la precisión de los resultados obtenidos.

La probabilidad de introducir errores en producción es una preocupación constante en el desarrollo de software. A pesar de contar con procesos de prueba y controles de calidad rigurosos, siempre existe la posibilidad de que algunos errores pasen desapercibidos y se filtren al entorno productivo. Esto puede ocurrir debido a diferentes factores, como la complejidad de los sistemas, la falta de cobertura en las pruebas,

la interacción entre múltiples componentes, los plazos ajustados o los cambios de último momento. Además, las discrepancias entre los entornos de pruebas y producción, como las configuraciones y los datos reales, pueden contribuir a la aparición de problemas que no se manifestaron durante las pruebas. Por lo tanto, es fundamental implementar estrategias y herramientas que minimicen esta probabilidad, tales como pruebas exhaustivas, revisiones de código, automatización de pruebas y análisis estático, con el fin de garantizar la máxima calidad posible en el software que se despliega en el entorno productivo.

2.2. Formulación del problema

Las empresas de TI se enfrentan al reto de asegurar la calidad del código.

- ¿Cómo diseñar e implementar un mecanismo para detección de errores en el comportamiento de los servicios Rest?

3. Objetivos del proyecto

3.1. Objetivo General

Diseñar e implementar un prototipo de un servicio web que permita realizar comparaciones de comportamientos basado en respuestas REST entre diferentes ambientes, para identificar y mostrar discrepancias.

3.2. Objetivos específicos

- Identificar las consecuencias y repercusiones que los errores puede tener en el funcionamiento del sistema, la calidad del software y la experiencia del usuario.
- Desarrollar un servicio web que permita comparar las respuestas, con el fin de identificar y mostrar las discrepancias en los servicios REST.
- Elaborar una caracterización de las posibles discrepancias a encontrar.
- Realizar pruebas de validación del servicio desarrollado para asegurar su funcionamiento.

3.3. Resultados esperados

- Un prototipo de un servicio web que permita realizar informes de las discrepancias encontradas en las respuestas de dos servicios Rest.
- Análisis de la implementación del prototipo en un contexto real.

aspecto a qui'?
descripción incompleta del objetivo

4. Alcance

En el presente proyecto de investigación, se busca diseñar y desarrollar e implementar un prototipo de servicio web que permita analizar y comparar las respuestas entre dos entornos: el ambiente de producción y el ambiente de pruebas (sandbox). Este análisis y comparación se limitará a transacciones de obtención de datos, excluyendo las operaciones de creación, actualización y eliminación de datos debido a la limitación inherente al manejo de datos reales.

El alcance del servicio web es recibir las respuestas de ambos entornos, realizar comparaciones y generar un informe con las diferencias encontradas. Este informe resaltarán los cambios específicos en el comportamiento de la aplicación, desde las diferencias en los atributos de las respuestas hasta los tipos de datos utilizados.

Es importante destacar que el sistema estará diseñado para procesar únicamente respuestas en formato JSON y siguiendo el estilo de arquitectura REST. Inicialmente, no se tendrán en cuenta estructuras muy complejas, se incluirán comparaciones con tipos de datos como int, float, string y bool. Además, no incluirá la funcionalidad de modificar o corregir automáticamente los errores detectados en el código. El servicio web se basará en tecnologías modernas y se enfocará en brindar una experiencia de uso sencilla. Además, se diseñará con la capacidad de escalar y manejar grandes volúmenes de respuestas para adaptarse a entornos de producción con alto tráfico. Asimismo, se contarán con configuraciones que permitirán ajustar el tiempo de procesamiento de las comparaciones, de acuerdo con las necesidades del implementador, para evitar afectar el entorno productivo. Es importante resaltar que este proyecto se centrará en la detección de errores causados por cambios en el comportamiento de las respuestas, y no en la evaluación de la lógica de negocio ni en la detección de vulnerabilidades de seguridad.

5. Justificación del trabajo de grado

Al proporcionar una herramienta que permite realizar comparaciones con interacciones reales en lugar de simuladas, se logrará identificar posibles fallos y, así, reducir significativamente el margen de errores en los despliegues productivos. Esto permitirá abordar de manera proactiva los errores inyectados en el código.

La incorporación de esta herramienta como un nuevo paso en el proceso de pruebas generará un mayor grado de confianza tanto en las etapas previas al despliegue como en la experiencia del usuario final. Al detectar y solucionar los errores antes de la fase de producción, se fortalecerá la confianza y satisfacción de los usuarios al ofrecer un producto más confiable y de mayor calidad. Esto contribuirá a generar una percepción positiva hacia la solución y a garantizar una experiencia satisfactoria para los usuarios.

En términos económicos, la detección temprana de errores a través de esta herramienta permitirá asegurar de manera más efectiva la operación del negocio. Al evitar fallos y problemas en el ambiente productivo, se reducirán los costos asociados con el soporte y la resolución de incidencias, además de evitar posibles pérdidas económicas por interrupciones en el servicio.

↳ Se podría resaltar también el tiempo que se ahorra el grupo de desarrollo al no tener que corregir estos errores detectados.

6. Marco teórico de referencia y antecedentes

6.1. Bases Teóricas

6.1.1. Pruebas de software

6.1.1.1. ¿Qué son las pruebas de software?

En la investigación titulada “Software Testing Techniques: A Literature Review” (Jamil et al.) ofrece una valiosa perspectiva sobre el concepto y la importancia de las pruebas de software en el desarrollo de sistemas. Las pruebas de software se definen como un proceso esencial de evaluación que tiene como objetivo determinar si un sistema cumple con los requisitos previamente especificados. Estas pruebas permiten comparar el resultado real obtenido durante la ejecución del software con el resultado esperado, lo que revela la presencia de posibles fallas o errores en el software desarrollado y proporciona información clave sobre el estado de la calidad del producto.

Las pruebas de software se llevan a cabo mediante la ejecución de diferentes técnicas y enfoques. En su investigación sobre técnicas de pruebas, (Rani and Gupta) mencionan que existen diversas técnicas, tales como pruebas unitarias, pruebas de integración, pruebas funcionales y pruebas de regresión, que se utilizan para evaluar diferentes aspectos del software durante su desarrollo y mantenimiento.

El proceso de pruebas de software se basa en la creación de casos de prueba que cubran diferentes escenarios y condiciones del sistema, incluyendo datos de entrada específicos y situaciones de uso realistas. Estos casos de prueba se ejecutan sistemáticamente y se comparan los resultados obtenidos con los resultados esperados, lo que permite detectar cualquier desviación y asegurar la corrección y la confiabilidad del software.

6.1.1.2. Importancia de las pruebas en el desarrollo de software.

puntos no
deben ir
en títulos

Las pruebas desempeñan un papel importante en la mejora de la confiabilidad y estabilidad del software. En el libro “The Art of Software Testing” (Myers et al.), destacan que las pruebas rigurosas permiten descubrir y solucionar errores antes de que el software sea utilizado por los usuarios finales, lo que reduce la posibilidad de fallos inesperados y aumenta la satisfacción del cliente. La detección temprana y la corrección de errores también disminuyen los riesgos asociados con el uso de aplicaciones defectuosas, como pérdida de datos o interrupción de servicios críticos.

Además de la detección de errores, las pruebas proporcionan información valiosa sobre la calidad del producto. (Jamil et al.) señalan que las pruebas permiten comparar los resultados reales con los esperados, identificando cualquier discrepancia y proporcionando una evaluación objetiva del estado del software. Este conocimiento de la calidad del producto ayuda a tomar decisiones informadas sobre su lanzamiento y proporciona confianza tanto a los desarrolladores como a los usuarios finales.

Falta mayúscula inicial Los autores (y) señalan ...

6.1.1.3. Tipos de pruebas de software

Existen diversos tipos de pruebas que se aplican con el objetivo de evaluar diferentes aspectos del sistema y garantizar su calidad. A continuación, se describen algunos de los principales tipos de pruebas utilizados en la industria:

- **Pruebas funcionales:** se centran en verificar si el software cumple con los requisitos funcionales especificados. Según (Sommerville), estas pruebas se basan en diseñar casos de prueba que evalúen el comportamiento del sistema ante diferentes entradas y condiciones. Su objetivo principal es determinar si el software realiza correctamente las funciones esperadas, sin errores o resultados inesperados, como se evidencia en la figura 2.

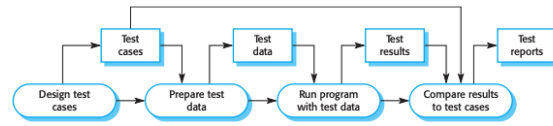


Figura 2: A model of the software testing process
(Sommerville)

- **Pruebas de rendimiento:** las pruebas de rendimiento se utilizan para evaluar el desempeño y la capacidad de respuesta del software bajo condiciones específicas de carga y estrés. De acuerdo con (Almeida et al.), estas pruebas se enfocan en medir aspectos como el tiempo de respuesta, la velocidad de procesamiento y el consumo de recursos del sistema. Su objetivo es identificar posibles cuellos de botella y optimizar el rendimiento del software.
- **Pruebas de usabilidad:** las pruebas de usabilidad se centran en evaluar la facilidad de uso y la experiencia del usuario al interactuar con el software. Según (Nielsen), estas pruebas se realizan con usuarios reales, quienes proporcionan retroalimentación sobre la interfaz, la navegación y la comprensión de las funcionalidades. El objetivo es identificar problemas de usabilidad y realizar mejoras para aumentar la satisfacción del usuario.
- **Pruebas de seguridad:** las pruebas de seguridad se realizan para evaluar la resistencia del software ante ataques maliciosos y garantizar la protección de los datos y la integridad del sistema. Según (Howard and Lipner), estas pruebas implican la identificación de vulnerabilidades y la aplicación de técnicas de penetración para verificar la robustez de las medidas de seguridad implementadas.

6.1.2. Estrategias de pruebas

Las estrategias de pruebas son enfoques sistemáticos utilizados para diseñar y ejecutar pruebas de software con el objetivo de lograr una cobertura adecuada y garantizar la calidad del producto final. A continuación, se describen algunas de las

estrategias de pruebas más comunes y su importancia en el proceso de desarrollo de software.

6.1.3. Concepto de estrategias de pruebas

El concepto de estrategias de pruebas hace referencia a enfoques sistemáticos y planificados para realizar pruebas de software con el objetivo de garantizar la calidad y fiabilidad del producto. Estas estrategias implican la selección y combinación adecuada de técnicas, herramientas y recursos disponibles para lograr una cobertura exhaustiva de las funcionalidades del software y detectar posibles fallas o errores. A continuación, se presenta una descripción del concepto de estrategias de pruebas.

Según (Bertolino), las estrategias de pruebas son “planes y enfoques organizados que guían el proceso de pruebas y proporcionan una estructura para garantizar la efectividad y eficiencia de las actividades de prueba”. Estas estrategias ayudan a identificar los tipos de pruebas adecuados, las técnicas y herramientas a utilizar, así como la asignación de recursos y la definición de objetivos de prueba.

6.1.3.1. Enfoques comunes en estrategias de pruebas

- **Pruebas de caja blanca (white-box):** se basa en el conocimiento interno de la estructura y el diseño del software. Se analiza el código fuente y se crean casos de prueba para cubrir diferentes caminos de ejecución, condiciones lógicas y ramas del código. Las pruebas de caja blanca son útiles para verificar la lógica interna del software y garantizar la cobertura de todas las sentencias y condiciones. Según (Myers et al.), las pruebas de caja blanca son especialmente efectivas para encontrar errores de lógica y de flujo de control.
- **Pruebas de caja negra (black-box):** se centra en probar el software sin conocer su estructura interna. Las pruebas se basan únicamente en los requisitos y la especificación del software. Se diseñan casos de prueba para cubrir

diferentes escenarios y se evalúa la respuesta del software frente a diferentes entradas. Las pruebas de caja negra son útiles para validar la funcionalidad y la usabilidad del software desde la perspectiva del usuario. Según (Pressman), las pruebas de caja negra se centran en verificar si el software cumple con los requisitos funcionales y no funcionales.

- **Pruebas unitarias:** este enfoque se centra en probar las unidades individuales de código fuente, como funciones, métodos o clases. Se busca verificar la corrección de cada unidad y su interacción con otras unidades. Según (Binder), las pruebas unitarias se realizan durante el desarrollo para detectar y corregir errores tempranos.
- **Pruebas de marcha blanca controlada:** este enfoque se centra en probar el software en un entorno similar al de producción, utilizando datos reales. Se busca validar el comportamiento del sistema en condiciones reales antes de su implementación. Según (15), las pruebas de marcha blanca controlada permiten identificar problemas relacionados con la configuración, la interoperabilidad y el desempeño del software.

Ref. numérica o con nombres de autors?
IEEE

6.1.3.2. Definición de casos de prueba

Los casos de prueba son un componente esencial en el proceso de pruebas de software. Los casos de prueba son un conjunto de condiciones o acciones diseñadas para evaluar el comportamiento de un sistema o componente específico. Estos casos representan situaciones reales o escenarios que se utilizan para verificar si el software cumple con los requisitos establecidos.

La definición de casos de prueba varía dependiendo de la metodología de pruebas utilizada. Según (Pressman), los casos de prueba pueden estar basados en requisitos, en los que cada caso se enfoca en verificar un requisito específico, o pueden ser escenarios realistas que simulan la interacción del usuario con el sistema. En ambos casos, los casos de prueba deben ser claros, comprensibles y medibles, de manera que

se pueda determinar si el software pasa o falla cada caso de prueba.

6.1.3.3. Componentes de un caso de prueba

Identificador único: cada caso de prueba debe tener un identificador único que lo distinga de otros casos. Esto facilita la referencia y seguimiento durante el proceso de pruebas. Según L. Beizer, “el uso de identificadores únicos es fundamental para la gestión efectiva de los casos de prueba” (Beizer).

Descripción: se debe proporcionar una descripción clara y concisa del objetivo del caso de prueba. Esto incluye los requisitos o funcionalidades específicas que se están probando. Según J. Fewster y D. Graham, “la descripción debe ser comprensible y brindar una visión clara de lo que se espera probar” (Fewster and Graham).

Entradas: especifica los datos o condiciones de entrada que se utilizan para ejecutar el caso de prueba. Estas entradas pueden incluir valores específicos, archivos de prueba, configuraciones o cualquier otra información relevante. S. Kaner et al. afirman que “las entradas deben ser lo suficientemente variadas como para cubrir diferentes escenarios y casos límite” (15).

Pasos de prueba: son las instrucciones detalladas que describen cómo ejecutar el caso de prueba. Estos pasos deben ser claros, precisos y secuenciales, de manera que cualquier persona pueda seguirlos fácilmente. Según R. Patton, “los pasos de prueba deben ser lo suficientemente detallados como para proporcionar una guía precisa de lo que se debe hacer” (Patton).

Resultados esperados: indica los resultados o comportamientos esperados del sistema bajo prueba después de ejecutar el caso. Esto permite comparar los resultados reales con los esperados y determinar si el software se comporta según lo previsto. K. Wiegers señala que “los resultados esperados deben ser específicos y medibles” (Wiegers).

Criterios de aceptación: establece los criterios que deben cumplirse para considerar que el caso de prueba es exitoso. Estos criterios pueden incluir valores específicos, mensajes de error esperados o cualquier otra condición relevante. Según D. Graham y M. Perry, “los criterios de aceptación deben ser claros y objetivos” (Fewster and Graham).

6.1.3.4. Técnicas de diseño de casos de prueba

Técnica del caso de prueba basado en especificaciones: la técnica del caso de prueba basado en especificaciones se centra en la revisión de los requisitos y especificaciones del software para identificar condiciones de prueba. Esta técnica se basa en comprender y analizar los documentos de requisitos y utilizar esa información para diseñar casos de prueba relevantes.

Según (Beizer), esta técnica ayuda a garantizar que los casos de prueba estén alineados con los requisitos del sistema y cubran adecuadamente los escenarios especificados.

Técnica del caso de prueba basado en estructura: la técnica del caso de prueba basado en estructura se enfoca en la estructura interna del software, como el código fuente y la arquitectura del sistema. Esta técnica utiliza conocimientos detallados sobre el diseño interno del software para identificar rutas de ejecución y puntos críticos que deben ser probados.

Según (Fewster and Graham), esta técnica permite diseñar casos de prueba que se centren en la lógica interna del software y ayuden a descubrir errores relacionados con la implementación.

Técnica del caso de prueba basado en experiencia: la técnica del caso de prueba basado en experiencia se basa en el conocimiento y la experiencia acumulada de los profesionales en pruebas de software. Estos profesionales utilizan su experiencia previa en proyectos similares para identificar áreas críticas, situaciones inusuales y escenarios relevantes que deben ser probados.

Técnica del caso de prueba basado en errores conocidos: la técnica del caso de prueba basado en errores conocidos se basa en el análisis de errores y fallas anteriores en el software para diseñar casos de prueba que aborden específicamente esas áreas problemáticas. Esta técnica se centra en aprender de los errores del pasado y diseñar pruebas que ayuden a prevenir la aparición de errores similares en el futuro.

6.1.4. Automatización de pruebas

6.1.4.1. Concepto de automatización de pruebas

La automatización de pruebas es un proceso que consiste en utilizar herramientas y software especializado para ejecutar pruebas de manera automatizada, en lugar de realizarlas manualmente. Esta técnica permite mejorar la eficiencia y la precisión de las pruebas, al reducir la intervención humana y proporcionar resultados rápidos y confiables. A continuación, se presentan algunas definiciones y perspectivas sobre el concepto de automatización de pruebas.

Según (Beizer), la automatización de pruebas se refiere al uso de herramientas y técnicas para ejecutar pruebas repetitivas, secuenciales y predecibles de forma automatizada. Estas pruebas automatizadas pueden abarcar desde pruebas unitarias hasta pruebas de sistema, permitiendo una mayor cobertura y una detección más temprana de errores.

(Binder) define la automatización de pruebas como el proceso de escribir y ejecutar scripts o programas que realizan pruebas automáticamente, sin la intervención manual del tester. Esta automatización puede incluir la interacción con la interfaz de usuario, la manipulación de datos y la verificación de resultados esperados.

6.1.4.2. Ventajas y desafíos de la automatización de pruebas

Ventajas

La automatización permite ejecutar pruebas de manera rápida y eficiente, reduciendo el tiempo y los recursos necesarios en comparación con las pruebas manuales (Sommerville).

Mayor cobertura de pruebas: La automatización facilita la ejecución de un gran número de pruebas en un período de tiempo limitado, lo que aumenta la cobertura de pruebas y permite detectar más errores potenciales (Kaczmarek et al.).

Reutilización de casos de prueba: Los casos de prueba automatizados pueden ser reutilizados en diferentes versiones del software, lo que ahorra tiempo en la creación de nuevos casos de prueba y garantiza una cobertura consistente (Salam and Mohd).

Mejora de la precisión y confiabilidad: Al eliminar la intervención manual, se reducen los errores humanos y se obtienen resultados más consistentes y confiables en las pruebas (Papadakis et al.).

Desafíos

La automatización de pruebas requiere una inversión inicial en herramientas y recursos de desarrollo, así como tiempo para desarrollar los scripts y mantener las pruebas actualizadas (Ammann and Offutt)

Selección adecuada de pruebas para automatizar: No todas las pruebas son adecuadas para la automatización. Es importante identificar qué pruebas se beneficiarán más de la automatización y cuáles deben seguir siendo ejecutadas manualmente (Kaner et al.).

Mantenimiento y actualización de las pruebas: A medida que el software evoluciona, las pruebas automatizadas deben ser actualizadas y mantenidas para garantizar su relevancia y efectividad continua (Jorgensen and Shepperd).

Necesidad de habilidades y conocimientos técnicos: La automatización de pruebas requiere personal con habilidades técnicas y conocimientos en el uso de herramientas y lenguajes de programación específicos (Sommerville).

6.1.4.3. Herramientas y tecnologías para la automatización de pruebas

La automatización de pruebas en el desarrollo de software se ha vuelto cada vez más importante debido a su capacidad para mejorar la eficiencia y la calidad de las pruebas. Existen diversas herramientas y tecnologías disponibles para facilitar este proceso. A continuación, se presentan algunas de las herramientas más utilizadas.

Selenium: selenium es una popular suite de herramientas de automatización de pruebas para aplicaciones web. Permite la automatización de acciones en navegadores web, como hacer clic en botones, llenar formularios y navegar por páginas. Esta herramienta es ampliamente adoptada y cuenta con una gran comunidad de usuarios y desarrolladores.

Appium: Appium es una herramienta de automatización de pruebas específicamente diseñada para aplicaciones móviles. Permite realizar pruebas en dispositivos iOS, Android y Windows utilizando un único conjunto de API. Appium es conocido por su compatibilidad con múltiples plataformas y por ofrecer un enfoque basado en estándares para la automatización de pruebas móviles.

JUnit: JUnit es un popular framework de pruebas unitarias para aplicaciones Java. Proporciona una estructura para escribir y ejecutar pruebas unitarias de manera eficiente. JUnit ofrece funciones para realizar aserciones, configurar y limpiar el entorno de prueba, y organizar las pruebas en suites. Es ampliamente utilizado en el desarrollo de software basado en Java.

TestNG: TestNG es otro framework de pruebas unitarias para aplicaciones Java. Ofrece características avanzadas en comparación con JUnit, como la capacidad de realizar pruebas en paralelo, la definición de dependencias entre pruebas y la generación de informes detallados. TestNG se ha vuelto popular en la comunidad de desarrollo de Java debido a su flexibilidad y funcionalidad mejorada.

Cucumber: Cucumber es una herramienta de automatización de pruebas basada en el lenguaje de dominio específico Gherkin. Permite escribir pruebas en un formato legible por humanos utilizando el enfoque de Desarrollo Dirigido por Comportamiento

acortar:
siglas en inglés
↗

(BDD). Cucumber se utiliza ampliamente para la colaboración entre los equipos de desarrollo y de pruebas, ya que facilita la comprensión y la comunicación de los requisitos y escenarios de prueba.

6.1.5. Comparación de respuestas en ambientes productivos y de pruebas

6.1.5.1. Importancia de comparar las respuestas en diferentes ambientes

En el desarrollo de software, es crucial comparar las respuestas y el comportamiento del sistema en diferentes ambientes, como el ambiente productivo y el ambiente de pruebas. Esta comparación proporciona información valiosa sobre la estabilidad, la eficiencia y la calidad del software.

En el estudio titulado “A Comparative Analysis of Production and Test Failures” realizado por (Dallmeier et al.), se demostró que la comparación de las respuestas en diferentes ambientes revela patrones y tendencias en los errores y fallas del software. Esto ayuda a los equipos de desarrollo a comprender mejor los puntos débiles del sistema y tomar medidas correctivas de manera más efectiva.

En el artículo “Comparing Production and Test Coverage” escrito por (Elbaum et al.), se destaca que la comparación de las respuestas en diferentes ambientes permite evaluar la efectividad de las pruebas realizadas durante el ciclo de desarrollo. Ayuda a identificar las áreas del sistema que no están adecuadamente cubiertas por las pruebas y que pueden ser fuentes potenciales de errores.

La comparación de las respuestas en diferentes ambientes proporciona una visión integral del software y ayuda a identificar posibles problemas y deficiencias. Permite realizar ajustes y mejoras antes de que el software se despliegue en el ambiente productivo, lo que a su vez contribuye a la entrega de un producto de alta calidad.

6.1.5.2. Diferencias comunes entre el ambiente productivo y el ambiente de pruebas

En el desarrollo de software, existen diferencias significativas entre el ambiente productivo y el ambiente de pruebas. Estas diferencias pueden afectar el comportamiento y rendimiento del sistema, así como la detección de errores y la validación de su correcto funcionamiento.

Según la investigación realizada por (Elbaum et al.) sobre la comparación de la cobertura de producción y pruebas, se observó que los errores y fallas en el ambiente productivo suelen ser diferentes a los encontrados durante las pruebas. Esto se debe a que el ambiente de pruebas generalmente no puede reproducir exactamente las condiciones del ambiente productivo, lo que resulta en la aparición de fallas que no fueron detectadas durante las pruebas.

Además, (Dallmeier et al.) realizaron un análisis comparativo de las fallas en producción y pruebas, y encontraron que las fallas en el ambiente productivo tienden a ser más críticas y tener un impacto más significativo en los usuarios finales. Esto se debe a que el ambiente productivo está expuesto a una mayor variedad de configuraciones, datos reales y cargas de trabajo, lo que puede revelar problemas que no fueron detectados en el ambiente de pruebas controlado.

Otra diferencia importante es la presencia de datos reales en el ambiente productivo, que pueden tener características y patrones que no se encuentran en los datos de prueba. (Jones and Bonsignour) enfatizan en su libro “The Economics of Software Quality” que la variabilidad en los datos de producción puede exponer vulnerabilidades y errores que no fueron considerados en las pruebas.

Las diferencias comunes entre el ambiente productivo y el ambiente de pruebas incluyen la presencia de errores no detectados durante las pruebas, la aparición de fallas más críticas en producción y la variabilidad de los datos reales. Estas diferencias resaltan la importancia de comparar las respuestas entre ambos ambientes para

identificar y solucionar posibles problemas que pueden surgir en la implementación del software.

6.2. Estado del Arte

6.2.1. Antecedentes de aplicaciones que permita comparar respuestas de diferentes ambientes

A continuación se presenta un análisis de las herramientas disponibles para realizar pruebas de comparación de respuestas rest api en diferentes ambientes. El análisis se realiza a partir de la documentación disponible en cada herramienta.

6.2.1.1. Postman

Postman se utiliza principalmente para realizar pruebas de API. Permite enviar solicitudes HTTP a un servidor o servicio web específico y recibir respuestas en diferentes formatos, como JSON o XML. Algunas de las funcionalidades y características clave son:

Creación y envío de solicitudes: proporciona una interfaz intuitiva para construir y enviar solicitudes HTTP. Permite especificar la URL, los parámetros, los encabezados y el cuerpo de la solicitud, y realizar solicitudes GET, POST, PUT, DELETE, entre otras.

Gestión de entornos y variables: permite definir entornos y variables que facilitan la configuración y personalización de las pruebas. Esto es especialmente útil cuando se trabaja con diferentes ambientes, como desarrollo, prueba y producción.

Automatización de pruebas: permite la automatización de pruebas mediante la creación de scripts utilizando JavaScript. Estos scripts pueden realizar pruebas de regresión, pruebas de integración y otras pruebas repetibles, lo que agiliza el proceso de pruebas.

Colecciones de solicitudes: permite organizar las solicitudes en colecciones, lo que facilita la reutilización y el mantenimiento de las pruebas. Las colecciones pueden contener diferentes solicitudes, así como scripts y variables asociadas.

Generación de documentación: permite generar documentación de API automáticamente a partir de las solicitudes y respuestas configuradas. Esto es útil para documentar y compartir la información sobre la API con otros miembros del equipo.

6.2.1.2. API Fortress

API Fortress es una plataforma de pruebas y monitorización de API (Application Programming Interface) que se utiliza para garantizar la calidad, rendimiento y confiabilidad de las API. Proporciona una serie de funcionalidades y características que facilitan el proceso de pruebas y monitoreo de las API.

API Fortress se utiliza para realizar pruebas funcionales y de rendimiento en API. Algunas de sus características principales son:

Creación de pruebas: crear pruebas funcionales para verificar que las API se comporten según lo esperado. Se pueden configurar diferentes casos de prueba para validar el funcionamiento de las rutas, los parámetros, las respuestas y las validaciones de datos.

Automatización de pruebas: permite la automatización de pruebas mediante la creación de flujos de trabajo. Estos flujos de trabajo pueden incluir diferentes pruebas, así como acciones adicionales como la generación de datos de prueba, el manejo de variables y la ejecución programada de las pruebas.

Monitorización de API: proporciona capacidades de monitorización en tiempo real de las API. Permite definir umbrales y alertas para detectar cualquier anomalía o error en el rendimiento de las API. Esto ayuda a identificar problemas y realizar un seguimiento proactivo de la salud de las API.

Generación de informes: ofrece la generación de informes detallados sobre los resultados de las pruebas y el rendimiento de las API. Estos informes pueden incluir métricas como el tiempo de respuesta, la tasa de éxito, los errores detectados y otras estadísticas relevantes.

Integraciones y colaboración: se integra con otras herramientas y sistemas, como sistemas de gestión de incidencias, sistemas de control de versiones y herramientas de CI/CD (Continuous Integration/Continuous Deployment). También permite la colaboración en equipo, lo que facilita la comunicación y el intercambio de información entre los miembros del equipo de desarrollo y pruebas.

6.2.1.3. Assertible

Assertible es una herramienta de pruebas de API y monitoreo de servicios web. Se utiliza para automatizar pruebas funcionales, realizar pruebas de integración y asegurar la calidad de las API.

La funcionalidad principal de Assertible se centra en la ejecución de pruebas automatizadas para verificar que las API se comporten correctamente. Permite definir casos de prueba con diferentes escenarios y realizar solicitudes HTTP para validar el comportamiento de las API, algunas de las características y usos clave de Assertible son:

Pruebas de respuesta: verificar y validar las respuestas de las API, asegurándose de que los datos devueltos sean los esperados. Puede realizar comparaciones de cuerpos de respuesta, encabezados, códigos de estado y otros atributos relevantes.

Pruebas de integración: permite realizar pruebas de integración entre diferentes servicios web y API. Esto es útil para garantizar que las integraciones entre sistemas funcionen correctamente y cumplan con los requisitos establecidos.

Monitoreo de servicios: ofrece monitoreo continuo de servicios web y API. Puede configurar reglas y alertas para detectar cualquier anomalía en el rendimiento de las API, como tiempos de respuesta lentos o errores frecuentes.

Integraciones y automatización: Assertible se integra con herramientas de CI/CD y sistemas de integración continua, lo que permite ejecutar pruebas automáticamente como parte del proceso de desarrollo. También se puede utilizar en entornos de integración y entrega continua para validar la calidad de las API en cada despliegue.

Generación de informes: genera informes detallados sobre los resultados de las pruebas, incluyendo métricas de rendimiento, éxitos y fallas, y otros datos relevantes. Estos informes son útiles para el análisis y seguimiento de la calidad de las API a lo largo del tiempo.

7. Metodología de la investigación

La metodología para el desarrollo del proyecto se basará en la metodología Scrum, que es un enfoque ágil ampliamente utilizado en el desarrollo de software.

Scrum es una metodología ágil de desarrollo de software que se centra en la entrega iterativa e incremental de un producto. Se basa en principios de transparencia, inspección y adaptación, lo que permite una respuesta rápida y flexible a los cambios en los requisitos del proyecto.

La metodología Scrum se divide en ciclos llamados "sprints", que son períodos de tiempo fijos y cortos (generalmente de 1 a 4 semanas) los que se desarrollan y entregan funcionalidades del producto. Para este proyecto se llevará a cabo sprints de 2 semanas.

Para utilizar Scrum en el proyecto, se seguirán los siguientes pasos:

1. Identificar requisitos y crear el backlog del producto.
2. Planificar los sprints.
3. Realizar los sprints.
4. Iterar y adaptar.
5. Pruebas y aseguramiento de la calidad.
6. Entrega y validación incremental.
7. Iteraciones adicionales.
8. Cierre del proyecto.

o historias de usuario? (HU)

Para el cierre del proyecto, se presentarán los resultados obtenidos tras la implementación del servicio web de comparaciones.

Faltan roles de Scrum!
Cuál es el equipo (personas) del proyecto?

8. Recursos a emplear

Los recursos utilizados en el desarrollo del prototipo incluyen personal involucrado y herramientas necesarias para garantizar el cumplimiento del proceso de desarrollo.

1. Recursos Humanos

- a) Director
- b) Co-Director
- c) Arquitecto de software

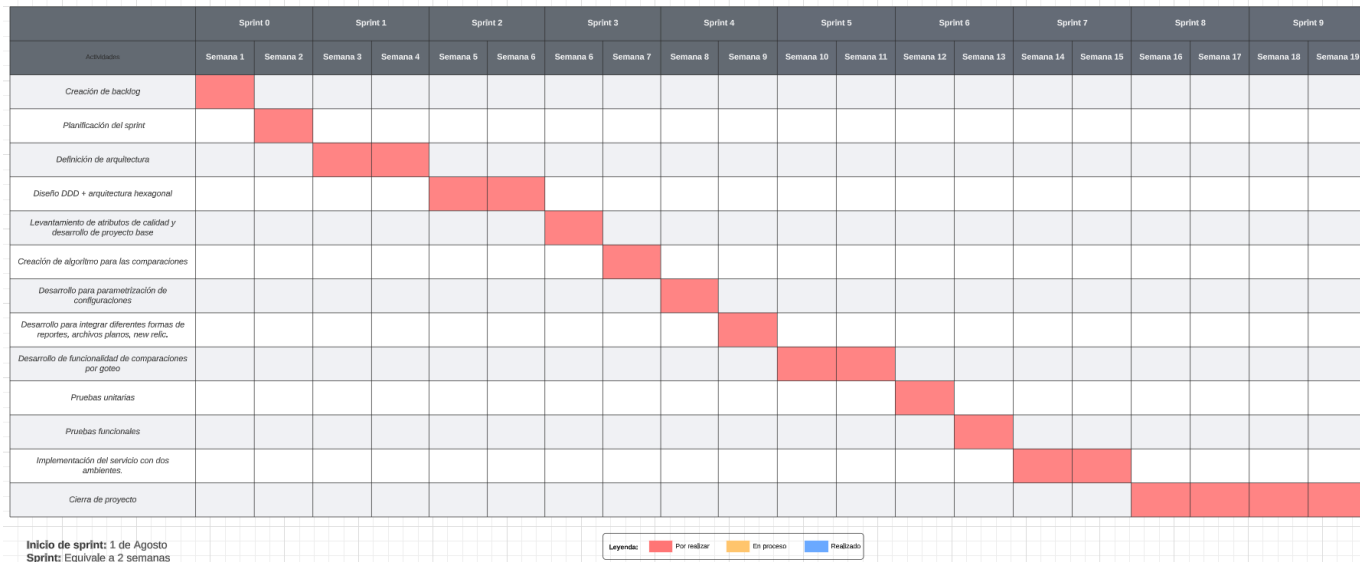
2. Recursos de herramientas y software

- a) Visual Studio Code
- b) Lucidchart
- c) Macbook Pro 2021

Aquí podría ir la descripción de roles de Scrum para el recurso humano.
Min:

- Product Manager
- Scrum Master
- Miembro del equipo

9. Cronograma de actividades



Estas actividades pueden detallarse en la metodología.

Importante: Relacionar cada actividad con un objetivo específico.

Deben haber la suficiente cantidad de actividades que permitan cumplir el objetivo.

10. Referencias Bibliográficas

Referencias

- [Almeida et al.] Almeida, V., Dowdy, L., Dowdy, L., Almeida, V. A. F., Menas, for Higher Education (Firm), O., and an O'Reilly Media Company. Safari. *Performance by Design: Computer Capacity Planning by Example*.
- [Ammann and Offutt] Ammann, P. and Offutt, J. An introduction to software testing. In *International Conference on Software Testing, Verification and Validation (ICST 2008)*, pages 498–499. IEEE.
- [Beizer] Beizer, B. *Software Testing Techniques*. International Thomson Computer Press.
- [Bertolino] Bertolino, A. Software testing research: Achievements, challenges, dreams.
- [Binder] Binder, R. *Testing Object-oriented Systems: Models, Patterns, and Tools*. Addison-Wesley object technology series. Addison-Wesley.
- [Dallmeier et al.] Dallmeier, V., Zimmermann, T., Zeller, A., and Bird, C. A comparative analysis of production and test failures. *Empirical Software Engineering*, 20(2):478–511.
- [Elbaum et al.] Elbaum, S., Rothermel, G., and Penix, J. Comparing production and test coverage. *IEEE Transactions on Software Engineering*, 28(7):666–678.
- [Fewster and Graham] Fewster, D. and Graham, M. *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley Professional.
- [Howard and Lipner] Howard, M. and Lipner, S. *The security development lifecycle : SDL, a process for developing demonstrably more secure software*. Microsoft Press.

Las referencias o son todas numéricas (IEEE)
o manejan su propio formato con nombre de autor,
entre otros...

- [Jamil et al.] Jamil, M. A., Arif, M., Abubakar, N. S. A., and Ahmad, A. Software testing techniques: A literature review. pages 177–182. Institute of Electrical and Electronics Engineers Inc.
- [Jones and Bonsignour] Jones, C. and Bonsignour, O. *The Economics of Software Quality*. Addison-Wesley Professional.
- [Jorgensen and Shepperd] Jorgensen, M. and Shepperd, M. Experience-based guidelines for test case generation. In *Proceedings of the 2007 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 225–234. IEEE.
- [Kaczmarek et al.] Kaczmarek, J., Persson, P., and Engström, E. Automated black-box test case generation for path coverage. In *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 2003)*, pages 269–280. IEEE.
- [Kaner et al.] Kaner, C., Falk, J., and Nguyen, H. Q. Automated software testing. *Software Testing & Quality Engineering Magazine*, 6(3):20–27.
- [15] Kaner, C., Falk, J., and Nguyen, H. Q. (1999). *Testing Computer Software*. Wiley, New York, 2nd edition edition.
- [Laranjeiro et al.] Laranjeiro, N., Agnelo, J., and Bernardino, J. A black box tool for robustness testing of rest services. *IEEE Access*, 9:24738–24754.
- [Mendes et al.] Mendes, N., Duraes, J., and Madeira, H. Evaluating and comparing the impact of software faults on web servers. pages 33–42.
- [Myers et al.] Myers, G., Sandler, C., and Badgett, T. *The Art of Software Testing*. ITPro collection. Wiley.
- [Nielsen] Nielsen, J. *Usability Engineering*. Academic Press.
- [Papadakis et al.] Papadakis, M., Daka, E., and Malevris, N. Automated software testing: a systematic literature review. *Software Quality Journal*, 26(3):621–669.

- [Patton] Patton, R. *Software Testing*. Sams Publishing.
- [Pressman] Pressman, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill higher education. McGraw-Hill Education.
- [Rani and Gupta] Rani, S. and Gupta, D. A comparative study of different software testing techniques: a review. *J. Adv. Shell Program*, 5(1):1–8.
- [Salam and Mohd] Salam, A. and Mohd, M. R. Automated test case generation from uml activity diagrams using backtracking search algorithm. *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on*, pages 717–722.
- [Sommerville] Sommerville, I. *Software engineering*.
- [Stecklein et al.] Stecklein, J. M., Dabney, J., Dick, B., Haskins, B., Lovell, R., and Moroney, G. Error cost escalation through the project life cycle.
- [Wiegers] Wiegers, K. E. *Software Requirements*. Microsoft Press.

11. Glosario de Términos

- **Pruebas de software:** Proceso de evaluación utilizado para verificar si un sistema o aplicación cumple con los requisitos especificados y detectar posibles errores o fallas.
- **Ambiente productivo:** Entorno en el cual se ejecuta y opera el software en producción, es decir, en uso por los usuarios finales.
- **Ambiente de pruebas:** Entorno controlado utilizado para probar el software antes de su implementación en el ambiente productivo, con el objetivo de identificar y corregir posibles problemas.
- **Comparación de respuestas:** Proceso de contrastar y analizar las respuestas generadas por el software en el ambiente productivo y el ambiente de pruebas, con el fin de identificar diferencias y posibles errores introducidos durante el desarrollo.
- **Errores de software:** Defectos o fallas en el software que pueden afectar su funcionamiento y rendimiento, y que necesitan ser identificados y corregidos.
- **Calidad del producto:** Medida de la excelencia o satisfacción del software en términos de su capacidad para cumplir con los requisitos y expectativas establecidos.
- **Casos de prueba:** Conjunto de condiciones y datos de entrada utilizados para probar una función o característica específica del software y verificar su comportamiento esperado.
- **Automatización de pruebas:** Utilización de herramientas y tecnologías para ejecutar pruebas de manera automatizada, ahorrando tiempo y esfuerzo en comparación con las pruebas manuales.
- **Estrategias de pruebas:** Planificación y enfoques específicos utilizados para llevar a cabo las pruebas de software de manera efectiva y eficiente.

- **Resultado real:** Salida o respuesta obtenida del software en el ambiente productivo durante su funcionamiento en condiciones reales.
- **Resultado esperado:** Salida o respuesta que se espera obtener del software según los requisitos y criterios establecidos previamente.