

Universidad Tecnológica de La Habana

“José Antonio Echeverría”



**Facultad de Ingeniería Informática**

**Diseño e implementación de pruebas automatizadas al módulo Carga y Descarga del sistema de planificación de recursos de Habana Club International.**

*Informe prácticas profesionales 1*

**Autores:** Pedro Pablo Chevalier Pérez ([pchevalier@ceis.cujae.edu.cu](mailto:pchevalier@ceis.cujae.edu.cu))

Christian Peña González ([cpena@ceis.cujae.edu.cu](mailto:cpena@ceis.cujae.edu.cu))

**Tutor:** Dr. C. Raisa Socorro Llanes (raisa@ceis.cujae.edu.cu)

La Habana, Cuba

Abril, 2025

Resumen

Abstract

Índice General

[Introducción 1](#_Toc194592909)

[Capítulo 1: Marco Teórico 5](#_Toc194592910)

[1.1 Procesos de las pruebas de software 5](#_Toc194592911)

[1.2 Tipos de pruebas de software 6](#_Toc194592912)

[1.3 Pruebas de software en sistemas de planificación de recursos empresariales 8](#_Toc194592913)

[1.4 Diferencias entre las pruebas manuales y las pruebas automatizadas 9](#_Toc194592914)

[1.5 Metodologías de pruebas de software 10](#_Toc194592915)

[1.6 Herramientas para las pruebas automatizadas 11](#_Toc194592916)

[Referencias Bibliográficas 13](#_Toc194592917)

# Introducción

En las últimas décadas, el software se ha consolidado como un pilar fundamental de la sociedad moderna, transformando radicalmente sectores como la salud, la educación, las finanzas y la comunicación[1]. Su influencia se extiende desde los dispositivos móviles que se utilizan diariamente hasta los complejos sistemas que controlan infraestructuras críticas, redefiniendo cómo las personas interactúan con la tecnología y entre sí mismas. Esta transformación digital ha generado un ecosistema donde el software ya no es una simple herramienta auxiliar, sino un componente esencial para el funcionamiento de la civilización actual.

La dependencia del software ha puesto de manifiesto la vital importancia de garantizar su calidad. Un error en un sistema bancario puede paralizar transacciones financieras en todo el mundo; una falla en equipos médicos podría comprometer vidas humanas; un defecto en sistemas de control industrial podría desencadenar catástrofes ambientales[2]. Las implicaciones económicas de estas fallas son igualmente alarmantes, con estimaciones que sitúan las pérdidas anuales en decenas de miles de millones de dólares[3].

Más allá del impacto financiero inmediato, el software defectuoso genera una cascada de consecuencias negativas: ciclos de desarrollo que se alargan indefinidamente por la necesidad constante de corregir errores; la erosión gradual pero implacable de la confianza de los usuarios; la frustración de los clientes traducida en abandono; y en casos extremos, disputas legales que pueden amenazar la supervivencia misma de las organizaciones[4, 5].

El ciclo de vida del desarrollo de software(SDLC por sus siglas en inglés) ha evolucionado para adaptarse a las necesidades actuales de la industria, priorizando la agilidad, la integración continua y la entrega rápida de software de calidad. En la actualidad, metodologías como DevOps, Agile y CI/CD(continuous integration and continuous delivery) han reemplazado en gran medida los enfoques tradicionales en cascada, promoviendo iteraciones frecuentes, retroalimentación constante y pruebas automatizadas en cada fase del desarrollo[6, 7]. Desde la planificación y el diseño hasta la implementación y el mantenimiento, el aseguramiento de la calidad (QA) juega un papel clave al integrarse de manera continua en el ciclo de vida[6].

El aseguramiento de la calidad del software representa un enfoque integral y proactivo que atraviesa todo el ciclo de vida del desarrollo. Lejos de ser una fase aislada que se ejecuta al final del proceso, constituye una filosofía orientada a prevenir defectos desde las etapas iniciales de conceptualización hasta el despliegue y mantenimiento. Este paradigma establece que la calidad no se puede "probar" posteriormente en un producto, sino que debe "construirse" desde su concepción[1, 2, 6].

En este contexto, las pruebas de software emergen como una disciplina fundamental cuyo objetivo principal no es demostrar la ausencia de errores, sino descubrir su presencia[2]. Esta distinción, aunque sutil, representa un cambio fundamental en el enfoque: las pruebas eficaces requieren una mentalidad orientada a la detección de problemas más que a la validación de funcionalidades ya implementadas.

Además de la estrategia de pruebas, es esencial contar con un equipo de pruebas capacitado y con experiencia. Los probadores deben tener un profundo conocimiento del dominio de la aplicación, así como habilidades técnicas para diseñar y ejecutar pruebas efectivas[2, 7]. La colaboración entre desarrolladores y probadores es crucial para asegurar que las pruebas sean integrales y que los defectos se identifiquen y solucionen rápidamente[8].

La automatización de las pruebas ha revolucionado esta disciplina, permitiendo la ejecución sistemática, consistente y repetible de casos de prueba sin intervención humana directa. No obstante, sería erróneo considerar la automatización como una panacea universal. Las pruebas manuales conservan un valor irreemplazable en aspectos como la evaluación de usabilidad, las pruebas exploratorias y aquellas situaciones donde la intuición y experiencia humanas resultan insustituibles. La estrategia óptima consiste en una combinación equilibrada de ambos enfoques, aprovechando las fortalezas de cada uno para crear un ecosistema de calidad robusto y eficiente[9, 10].

Esta transición hacia metodologías ágiles y DevOps en la industria del desarrollo de software ha evidenciado la necesidad crítica de integrar pruebas automatizadas como parte fundamental del ciclo de desarrollo. Sin embargo, múltiples organizaciones siguen operando sin incorporar estas prácticas de manera sistemática en sus proyectos, lo que genera ineficiencias operativas y riesgos potenciales para la calidad del producto final.

Este es precisamente el caso de **AVANGENIO**, una empresa de desarrollo de software que, hasta hace poco, llevaba a cabo sus proyectos sin contar con equipos dedicados a la automatización de pruebas. Los procesos de verificación y validación se realizaban predominantemente de forma manual, lo que implicaba limitaciones significativas en términos de cobertura, consistencia y capacidad de regresión. Esta situación, común en muchas empresas del sector, suele derivar de factores como la percepción de alto costo inicial de implementación, la escasez de personal especializado o la subestimación del retorno de inversión de las pruebas automatizadas.

Estas limitaciones en la eficiencia y la cobertura de las pruebas provocaron el retraso en entregas anteriores de los módulos del sistema de gestión para Habana Club International, tomándose la decisión de añadir un equipo para realizar pruebas automatizadas al nuevo módulo en desarrollo, llamado “Carga y Descarga”.

Partiendo de la **situación problemática** anteriormente descrita, es posible plantear el **problema a resolver**:

¿Cómo se puede diseñar e implementar un sistema de pruebas automatizadas para el nuevo módulo en desarrollo que sea compatible con las tecnologías utilizadas?

Para resolver la problemática planteada, se propone el siguiente **objetivo general**:

Desarrollar un sistema de pruebas automatizadas para el nuevo módulo garantizando la compatibilidad con las tecnologías utilizadas para su desarrollo.

Para cumplir con el objetivo general planteado, se definen los **objetivos específicos** siguientes:

1. Identificar los elementos fundamentales para desarrollar un sistema de pruebas automatizadas compatible con las tecnologías utilizadas en la creación del sistema de gestión de recursos.
2. Diseñar los casos de prueba para el módulo “Carga y Descarga” que garanticen la explotación de las funcionalidades
3. Implementar los casos de prueba siguiendo patrones de diseño que favorezcan la reutilización y el mantenimiento de los conjuntos de prueba.

# Capítulo 1: Marco Teórico

En el presente capítulo se abordan los conceptos fundamentales de las pruebas de software y las metodologías utilizadas para su integración en el desarrollo de software. Además, se presenta la automatización de las pruebas de software como una necesidad dentro del proceso de desarrollo de software. Por último, se hace un análisis de la aplicabilidad de los temas tratados a la solución de la problemática planteada.

## Procesos de las pruebas de software

La calidad de un producto de software es determinada por el grado de cumplimiento de las necesidades del cliente y los usuarios. Según [1] es *“proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan”.*

​Las **pruebas de software** comprenden un conjunto de procesos y técnicas diseñados para evaluar y verificar la calidad de un producto informático. Su propósito esencial es identificar y corregir defectos en el software antes de su implementación en producción, minimizando riesgos y asegurando su fiabilidad. Integrar estas pruebas en el ciclo de vida del desarrollo de software (SDLC) es crucial, ya que la detección temprana de anomalías puede reducir significativamente las iteraciones en el proceso de desarrollo y optimizar el uso de recursos[11].

El proceso de pruebas tiene cuatro etapas fundamentales: planificación, diseño, ejecución y análisis, cada una de ellas desempeña un papel fundamental en el propósito de garantizar la calidad del producto[1].

En la fase de planificación se definen los objetivos, el alcance y los criterios de aceptación del software. Además, se identifican los recursos humanos y tecnológicos necesarios, se asignan roles y responsabilidades, y se elabora un cronograma que contemple hitos y plazos. Una planificación meticulosa permite prever riesgos potenciales y establecer estrategias para mitigarlos, asegurando que las pruebas se realicen de manera organizada y eficiente[2, 12].

Una vez definidos los objetivos, se procede a la fase de diseño, donde se elaboran los casos de prueba y se determinan las condiciones y escenarios que deben evaluarse. Esto implica identificar los requisitos funcionales y no funcionales, así como definir los datos de entrada y los resultados esperados. Un buen diseño de pruebas debe ser exhaustivo y adaptable, permitiendo cubrir tanto situaciones comunes como escenarios excepcionales que podrían afectar el desempeño del software[2, 8].

A partir de los casos de prueba diseñados los equipos de testing ejecutan las pruebas de forma manual o automatizada, utilizando herramientas específicas según las necesidades del proyecto. Durante la ejecución se registra el comportamiento del sistema, se capturan evidencias y se documentan los resultados. Esta fase es crucial, ya que permite identificar defectos y fallos en el software, proporcionando la información necesaria para corregirlos antes de su implementación definitiva[2].

La última etapa consiste en analizar y evaluar los resultados obtenidos durante la ejecución de las pruebas. Se revisan los informes y se clasifican los defectos según su severidad e impacto en el sistema. Este análisis permite determinar si el software cumple con los estándares de calidad requeridos y si está listo para pasar a la siguiente fase del ciclo de vida del desarrollo. Además, se generan recomendaciones y se planifican acciones correctivas, lo que contribuye a mejorar tanto el producto final como los procesos de desarrollo y pruebas futuros[2].

Cada una de estas etapas es interdependiente y contribuye de forma decisiva a asegurar que el software no solo cumpla con los requisitos funcionales y de rendimiento, sino que también ofrezca una experiencia de usuario segura y confiable. Implementar un proceso de pruebas robusto y bien estructurado es, por tanto, una inversión clave en la calidad y éxito de cualquier proyecto de desarrollo de software.

## Tipos de pruebas de software

Existen diversos tipos de pruebas, cada uno diseñado para evaluar aspectos específicos del software y aplicable en distintas etapas del ciclo de vida del desarrollo. A continuación, se describen los principales tipos:

### Pruebas Unitarias:

Estas pruebas se centran en evaluar de forma individual cada componente o unidad del software, como funciones o métodos. El objetivo es asegurar que cada unidad funcione correctamente de forma aislada, permitiendo detectar errores en etapas muy tempranas del desarrollo[13, 14].

### Pruebas de Integración:

Una vez que se han verificado las unidades de forma individual, es necesario comprobar que interactúan correctamente entre sí. Las pruebas de integración se enfocan en la comunicación y el intercambio de datos entre módulos, detectando problemas en la interfaz y en la coordinación de componentes[13, 14].

### Pruebas Funcionales:

Estas pruebas evalúan si el software cumple con los requisitos especificados y realiza las funciones esperadas. Se realizan casos de prueba basados en escenarios de uso real, verificando la correcta ejecución de las operaciones, la manipulación de datos y la generación de resultados adecuados[13, 14].

### Pruebas de Sistema:

En esta etapa se examina el software en su totalidad, considerando la interacción de todos los módulos en un entorno que simula el ambiente de producción. Se evalúan tanto aspectos funcionales como no funcionales, garantizando que el sistema completo opere de manera coherente y estable[13, 14].

### Pruebas de Regresión:

Con cada nueva actualización o modificación del software, es fundamental asegurarse de que las funcionalidades existentes no se vean afectadas. Las pruebas de regresión se ejecutan de forma repetida para detectar la reaparición de errores o nuevos defectos introducidos por cambios recientes[13].

### Pruebas de Rendimiento:

Este tipo de pruebas se concentra en evaluar la capacidad del software para funcionar eficientemente bajo diversas cargas de trabajo. Se analizan aspectos como tiempos de respuesta, consumo de recursos y estabilidad cuando el sistema es sometido a altos volúmenes de transacciones o usuarios simultáneos[13].

### Pruebas de Seguridad:

Dado que el software puede estar expuesto a vulnerabilidades y amenazas, las pruebas de seguridad son cruciales. Estas evaluaciones identifican posibles brechas en la protección de datos, problemas en los mecanismos de autenticación y autorización, y otros riesgos que puedan comprometer la integridad y confidencialidad del sistema[13].

### Pruebas de Usabilidad:

Estas pruebas se enfocan en la experiencia del usuario, evaluando la interfaz, la navegabilidad y la facilidad de uso del software. Un sistema intuitivo y accesible es fundamental para lograr la satisfacción de los usuarios finales[13].

### Pruebas de Aceptación:

Finalmente, las pruebas de aceptación, ya sean internas o por parte del cliente (UAT, User Acceptance Testing), verifican que el software cumpla con las expectativas y requisitos del negocio. Es el último paso antes de la puesta en producción, donde se valida la funcionalidad desde la perspectiva del usuario final[13].

## Pruebas de software en sistemas de planificación de recursos empresariales

Las pruebas de software en los ERPs son fundamentales para garantizar la robustez y confiabilidad de sistemas que gestionan procesos críticos en las organizaciones. Debido a la complejidad de estos sistemas(que integran módulos de contabilidad, recursos humanos, inventarios, logística y más) la realización de pruebas exhaustivas se vuelve indispensable para asegurar que cada componente funcione de manera óptima, tanto de forma aislada como en conjunto

En este tipo de sistemas es esencial abordar las pruebas funcionales: se verifica que cada módulo cumpla con los requerimientos específicos y que los flujos de trabajo empresariales se ejecuten sin contratiempos. Estas pruebas se diseñan a partir de los procesos de negocio que el ERP debe soportar, asegurando que cada transacción y operación se procese correctamente.

Las pruebas de integración cobran especial relevancia en los ERPs, dado que los diferentes módulos deben comunicarse de forma fluida. Este tipo de pruebas ayuda a identificar posibles fallos en la interconexión de los componentes, garantizando la coherencia y sincronización de la información entre áreas tan diversas como ventas y finanzas o inventario y compras.

Las pruebas de rendimiento y escalabilidad son críticas, ya que los ERPs suelen manejar grandes volúmenes de datos y múltiples usuarios simultáneos. Evaluar la respuesta del sistema bajo carga permite identificar cuellos de botella y optimizar recursos para mantener la operatividad en entornos de alta demanda.

Las pruebas de seguridad son indispensables para proteger la integridad de los datos y asegurar que el sistema cumpla con normativas y estándares de protección de información. La detección temprana de vulnerabilidades evita posibles brechas que podrían comprometer la confidencialidad y disponibilidad de datos sensibles.

## Diferencias entre las pruebas manuales y las pruebas automatizadas

Las pruebas manuales y las pruebas automatizadas son dos enfoques fundamentales en el aseguramiento de la calidad del software, y cada uno ofrece ventajas específicas que, combinadas, permiten cubrir una amplia gama de escenarios y necesidades en el desarrollo.

Las pruebas manuales son realizadas por probadores sin ayuda de scripts o herramientas automatizadas. Este enfoque es especialmente valioso cuando se trata de evaluar la experiencia del usuario y la usabilidad, ya que permite a los evaluadores interactuar directamente con la aplicación, detectar problemas de navegación, inconsistencias en la interfaz o aspectos subjetivos que podrían pasar desapercibidos para una máquina. Además, en etapas tempranas o en procesos de exploración, las pruebas manuales facilitan la identificación de errores imprevistos, pues el evaluador puede adaptar su enfoque en función de lo que observa en tiempo real[15].

Las pruebas automatizadas consisten en la ejecución de scripts que verifican la funcionalidad del software de manera repetitiva y sistemática. Este método resulta indispensable en el SDLC, ya que permite:

Eficiencia y rapidez: Los tests automatizados se pueden ejecutar en cada integración o despliegue, reduciendo significativamente el tiempo de validación y permitiendo identificar errores de forma inmediata[16].

Consistencia: Al eliminar la variabilidad humana, se garantiza que cada prueba se realice de forma uniforme, lo que mejora la confiabilidad de los resultados[16].

Cobertura amplia: Permiten ejecutar un gran número de casos de prueba, incluyendo pruebas de regresión, de rendimiento y de seguridad, asegurando que las nuevas implementaciones no afecten funcionalidades ya existentes[16].

Integración en entornos CI/CD: La automatización se integra perfectamente en pipelines de integración y despliegue continuo, asegurando que cada cambio en el código sea validado de manera oportuna, lo que es esencial para mantener la calidad en proyectos de desarrollo ágil[16].

Si bien las pruebas automatizadas se han vuelto una necesidad dentro del ciclo de desarrollo moderno, especialmente en entornos de integración continua y despliegue rápido, las pruebas manuales siguen siendo cruciales para evaluar aspectos que requieren el juicio humano, como la experiencia del usuario y la interfaz gráfica. La combinación de ambos enfoques permite aprovechar lo mejor de cada uno: la precisión, rapidez y cobertura que ofrecen las pruebas automatizadas, junto con la capacidad de análisis detallado y contextual que brindan las pruebas manuales[17].

## Metodologías de pruebas de software

Históricamente, el modelo en cascada ha sido utilizado en proyectos donde se requiere una secuencia lineal de etapas. En este enfoque, las pruebas se realizan al final del proceso de desarrollo, lo que puede derivar en una detección tardía de errores y en altos costos de corrección[17]. Debido a esto, las metodologías de pruebas actuales se han diversificado para responder a las demandas de ciclos de desarrollo más cortos, la necesidad de integración continua y la creciente complejidad de las aplicaciones[2]. Aunque este método ha sido reemplazado en gran medida por enfoques más ágiles, sigue siendo útil en entornos con requisitos bien definidos y poca variabilidad.

La revolución de los métodos ágiles ha transformado la forma de abordar las pruebas de software. En este paradigma, las pruebas se integran desde las etapas iniciales del desarrollo y se ejecutan de manera iterativa junto con la codificación. Técnicas como el **Test Driven Development (TDD),** donde se escriben primero los casos de prueba para luego desarrollar el código que los satisfaga, y el **Behavior Driven Development (BDD),** que utiliza un lenguaje común entre desarrolladores y clientes para definir escenarios de comportamiento, permiten una mayor colaboración y una respuesta rápida ante cambios. Estas metodologías facilitan la integración continua, reducen el ciclo de retroalimentación y aseguran que el software cumpla con los requerimientos del usuario[18].

Dentro de los enfoques modernos la necesidad de integrar las pruebas automatizadas en ciclos de desarrollo acelerados han promovido la integración de estas pruebas en pipelines de CI/CD para detectar defectos en cada nueva integración al producto final[19].

## Herramientas para las pruebas automatizadas

Las herramientas para pruebas automatizadas son fundamentales para mejorar la eficiencia y la cobertura en el proceso de aseguramiento de calidad. Estas herramientas permiten ejecutar pruebas de forma repetitiva y sistemática, integrándose en entornos CI/CD. Las herramientas más conocidas y utilizadas en la industria son:

### Selenium WebDriver

Sin duda, una de las herramientas más reconocidas para la automatización de pruebas en aplicaciones web. Con soporte para múltiples lenguajes de programación y navegadores, Selenium permite simular la interacción del usuario y validar el comportamiento de la aplicación en diferentes entornos. Su flexibilidad y comunidad activa lo convierten en una opción robusta para proyectos de diversa envergadura[20].

### Cypress

Cypress está especialmente dedicada al ámbito de las aplicaciones modernas basadas en JavaScript. Se destaca por su fácil configuración, tiempos de ejecución rápidos y su capacidad para realizar pruebas de extremo a extremo, integrándose de manera fluida en pipelines de CI/CD. Su enfoque en el testing de front-end y su interfaz intuitiva hacen que sea una excelente opción para equipos que buscan resultados inmediatos y una curva de aprendizaje amigable[20].

### Playwright

Desarrollada por Microsoft, es otra herramienta emergente en el mundo de las pruebas automatizadas. Similar a Cypress en cuanto a su capacidad para realizar pruebas de aplicaciones web, Playwright ofrece la ventaja de soportar múltiples navegadores y dispositivos, permitiendo una mayor cobertura en entornos variados. Su diseño moderno y su integración con lenguajes como Python y JavaScript facilitan la creación de pruebas robustas y escalables[20].

### Appium

**Appium** se posiciona como una herramienta esencial para las pruebas de aplicaciones móviles. Permitiendo la automatización tanto de aplicaciones nativas como híbridas, Appium se integra con múltiples plataformas móviles y soporta diversos lenguajes de programación. Esto facilita la ejecución de pruebas de interfaz y funcionalidad en dispositivos reales o emulados, garantizando que las aplicaciones móviles cumplan con los estándares de calidad requeridos[20].

Además de estas herramientas especializadas en pruebas de interfaz, existen frameworks orientados a pruebas unitarias y de integración, como **JUnit** para aplicaciones en Java y **pytest** para Python. Estas herramientas son clave para validar el comportamiento de componentes individuales y asegurar que las integraciones entre ellos sean correctas, permitiendo una detección temprana de errores a nivel de código.

# Referencias Bibliográficas

1. Pressman, R.S., *Software engineering: a practitioner's approach*. 2005: Palgrave macmillan.

2. Myers, G.J., C. Sandler, and T. Badgett, *The art of software testing*. 2011: John Wiley & Sons.

3. Krasner, H., *The cost of poor software quality in the US: A 2020 report.* Proc. Consortium Inf. Softw. QualityTM (CISQTM), 2021. **2**: p. 3.

4. Pargaonkar, S., *A Study on the Benefits and Limitations of Software Testing Principles and Techniques: Software Quality Engineering.* International Journal of Scientific and Research Publications (IJSRP) 13(08) (ISSN: 2250-3153), 2023.

5. Carter, D.O., *Advancing Software Quality: A Comprehensive Exploration of Code Quality Metrics, Static Analysis Tools, and Best Practices.* Journal of Science & Technology, 2024. **5**(1): p. 69-81.

6. Pargaonkar, S., *A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering.* International Journal of Scientific and Research Publications (IJSRP), 2023. **13**(08): p. 345-358.

7. Crispin, L. and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*. 2009: Addison-Wesley.

8. Jorgensen, P.C., *Software Testing: A Craftsman’s Approach, Fourth Edition*. 2013: Taylor & Francis.

9. Wang, J., et al., *Software testing with large language models: Survey, landscape, and vision.* IEEE Transactions on Software Engineering, 2024.

10. Pargaonkar, S., *Synergizing Requirements Engineering and Quality Assurance: A Comprehensive Exploration in Software Quality Engineering* International Journal of Science and Research (IJSR), 2022. **12**(8).

11. Black, R. and J.L. Mitchell, *Advanced Software Testing - Vol. 3: Guide to the ISTQB Advanced Certification as an Advanced Technical Test Analyst*. 2011: Rocky Nook.

12. Stapp, L., A. Roman, and M. Pilaeten, *ISTQB® Certified Tester Foundation Level: A Self-Study Guide Syllabus v4.0*. 2023: Springer Nature Switzerland.

13. Spillner, A., T. Linz, and H. Schaefer, *Software Testing Foundations: A Study Guide for the Certified Tester Exam*. 2014: Rocky Nook.

14. Jamil, M.A., et al. *Software testing techniques: A literature review*. in *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*. 2016. IEEE.

15. Thant, K. and H.H.K. Tin, *THE IMPACT OF MANUAL AND AUTOMATIC TESTING ON SOFTWARE TESTING EFFICIENCY AND EFFECTIVENESS.* 2023. **3**: p. 88-93.

16. Kumar, D. and K.K. Mishra, *The Impacts of Test Automation on Software's Cost, Quality and Time to Market.* Procedia Computer Science, 2016. **79**: p. 8-15.

17. Homès, B., *Fundamentals of software testing*. 2024: John Wiley & Sons.

18. Beck, K., *Test driven development: By example*. 2022: Addison-Wesley Professional.

19. Spinellis, D., *State-of-the-Art Software Testing.* IEEE Software, 2017. **34**(5): p. 4-6.

20. Donvir, A. *A Comparative Analysis of JavaScript Unit and End-to-End Testing Frameworks: Enhancing Quality Assurance in Modern Web Applications*. in *2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN)*. 2024.