



Sistema para la gestión y automatización de pruebas de software mediante integración continua en una universidad pública

System for managing and automating software testing using continuous integration in a public university

Marco Antonio Morales Colores ✉

Universidad Autónoma de Baja California, México
marco.morales@uabc.edu.mx

Christian Xavier Navarro Cota

Universidad Autónoma de Baja California, México
cnavarro@uabc.edu.mx

Everardo Gutiérrez López

Universidad Autónoma de Baja California, México
everardo.gutierrez@uabc.edu.mx

Omar Christian Aparicio Ortiz

Universidad Autónoma de Baja California, México
oc.aparicio@uabc.edu.mx

José Ángel González Fraga

Universidad Autónoma de Baja California, México
angel_fraga@uabc.edu.mx



<https://doi.org/10.36825/RITI.14.35.003>

Recibido: Mayo 22, 2026
Aceptado: Octubre 02, 2026

Resumen: En la actualidad, la calidad del software es un factor crítico en el desarrollo de sistemas, debido a su impacto en la confiabilidad, el rendimiento y la satisfacción del usuario. Las pruebas de software desempeñan un papel fundamental en el aseguramiento de la calidad; sin embargo, en muchos entornos estos procesos continúan realizándose de manera manual, lo que puede ocasionar ineficiencias, limitada trazabilidad y riesgos en la calidad del producto final. El objetivo de este trabajo es desarrollar e implementar una solución tecnológica basada en integración continua, que mejore el seguimiento y la automatización del proceso de pruebas de software en el departamento de desarrollo de sistemas de una universidad pública. La propuesta integra una aplicación web para la gestión de solicitudes de pruebas, seguimiento de proyectos y consulta de resultados, junto con una plataforma de integración continua que automatiza la ejecución de pruebas unitarias, funcionales y de rendimiento, además de realizar análisis estático de código. Para el desarrollo de la solución se adoptó la metodología *Extreme Programming* (XP). La evaluación del sistema se realizó mediante el Modelo de Aceptación Tecnológica (TAM) y la Escala de Usabilidad del Sistema (SUS). Los resultados obtenidos indican mejoras en la eficiencia del proceso, optimización de recursos y fortalecimiento de la calidad del software.

Palabras clave: *Pruebas de Software, Integración Continua, Pruebas Automatizadas.*

Abstract: Currently, software quality is a critical factor in system development, as it directly impacts reliability, performance, and user satisfaction. Software testing plays a fundamental role in software quality assurance by validating system behavior. However, in many environments, testing processes are still performed manually, which may lead to inefficiencies, limited traceability, and potential risks affecting product quality. The objective of this work is to develop a technological solution to improve the monitoring and automation of the software testing process within the systems development department of a public university. The proposed solution consists of two main components. The first is a web-based application designed to manage testing requests, monitor project progress, and facilitate test result consultation. The second component is a continuous integration platform that automates the execution of unit, functional, and performance tests, as well as static code analysis. Extreme Programming (XP) was adopted as the development methodology. The system was evaluated using the Technology Acceptance Model (TAM) and the System Usability Scale (SUS), focusing on perceived usefulness, ease of use, and usability. The results indicate that the proposed solution improves process efficiency, optimizes resource utilization, and enhances software quality.

Keywords: *Software Testing, Continuous Integration, Automated Testing.*

1. Introducción

El desarrollo de software se define como un conjunto de actividades orientadas a la creación de productos que satisfacen necesidades tecnológicas específicas [1]. En la actualidad, este proceso ha experimentado un crecimiento significativo debido a su impacto en diversos sectores, como la salud, educación, finanzas y la industria [2]. En este contexto, podemos afirmar que la calidad del software se considera en un elemento crítico, ya que afecta directamente en la confiabilidad, rendimiento y experiencia del usuario.

Tradicionalmente, las metodologías de desarrollo tradicionales han enfocado el aseguramiento de la calidad en las etapas finales del ciclo de vida [3]. En contraste, las metodologías ágiles promueven un enfoque iterativo e incremental, con una participación constante del cliente y una mayor flexibilidad en el proceso de desarrollo [4]. No obstante, independientemente del enfoque metodológico, la creciente demanda del mercado por productos de alta calidad en tiempos reducidos representa un desafío significativo para los equipos de desarrollo, impactando directamente en la calidad de los sistemas [5]. Ya que la calidad es considerada uno de los atributos más importantes del software, una baja calidad tiende a generar distintos defectos desde una mala experiencia hasta daños [6].

En este escenario, las pruebas de software constituyen un mecanismo fundamental para evaluar la calidad de los sistemas y verificar el cumplimiento de los requisitos [7]. No obstante, en muchos entornos organizacionales, estas actividades se realizan de manera manual, lo que limita la eficiencia del proceso, dificulta la trazabilidad y aumenta la probabilidad de errores humanos. Algunas metodologías integran pruebas automatizadas, lo que proporciona ventajas como la detección temprana de defectos, mayor estabilidad del software y reducción en los tiempos y costos de desarrollo [8].

Por otro lado, DevOps es un paradigma cuyo objetivo es la automatización de los procesos de construcción, pruebas y despliegue de software [9]. Este enfoque permite ciclos más rápidos de desarrollo sin comprometer la calidad del producto [10]. Dentro de DevOps, la integración continua facilita la integración y validación temprana del código mediante la ejecución automatizada de pruebas, permitiendo identificar defectos de manera oportuna y mejorar la calidad del software [11].

El presente trabajo se desarrolla en una universidad pública del noroeste de México, específicamente en un departamento encargado del desarrollo de sistemas y aseguramiento de la calidad de software. Actualmente, el proceso de pruebas se realiza mediante medios manuales, lo que dificulta el seguimiento, la trazabilidad y la eficiencia operativa. Como solución, se propone el desarrollo de una herramienta tecnológica compuesta por una aplicación web para la gestión de solicitudes de pruebas, seguimiento de proyectos y consulta de resultados, junto con una plataforma de integración continua que automatiza pruebas unitarias, funcionales, de rendimiento y análisis estático de código.

La integración de ambos componentes permite establecer un proceso de pruebas más eficiente, automatizado y medible, proporcionando información en tiempo real sobre el estado de los proyectos, optimizando recursos y mejorando la calidad de los sistemas desarrollados.

El objetivo de este trabajo es desarrollar e implementar una solución tecnológica basada en integración continua para mejorar el seguimiento y la automatización del proceso de pruebas de software en una universidad pública. En este contexto, el presente trabajo busca responder la siguiente pregunta de investigación: ¿Cómo contribuye una solución tecnológica basada en integración continua a optimizar el seguimiento y la automatización del proceso de pruebas de software en una universidad pública?, el resto del artículo está organizado de la siguiente manera: la Sección 2 presenta el estado del arte y los trabajos relacionados; la Sección 3 describe la metodología empleada para el desarrollo de la solución propuesta; la Sección 4 presenta el proceso de evaluación y los resultados obtenidos; la Sección 5 presenta la discusión y finalmente, la Sección 6 presenta las conclusiones del trabajo.

2. Estado del arte

Para conocer el estado actual del arte en el tema de pruebas automatizadas con integración continua, se realizó un mapeo sistemático de la literatura siguiendo las directrices de Kitchenham que establece tres fases; planeación, ejecución y reporte [12] y la metodología PRISMA para la fase de identificación, selección y filtrado [13]. Inicialmente se definieron las siguientes preguntas de investigación:

- ¿En qué tipo de proyectos o áreas se han aplicado pruebas automatizadas?
- ¿Qué herramientas o software existen para la administración y seguimiento de pruebas automatizadas?
- ¿Qué tipos de pruebas automatizadas se han ejecutado en proyectos de integración continua?
- ¿Cuáles son las principales ventajas de implementar pruebas automatizadas?

A partir de estas preguntas se construyó una cadena de búsqueda que combinó términos relacionados con automatización de pruebas, integración continua y desarrollo de software. A continuación, se muestra la cadena de búsqueda generada con las palabras claves:

("Test automation" OR "Automated testing" OR "Continuous testing" OR "Software testing") AND ("Continuous integration" OR "CI/CD" OR "CI" OR "CD" OR "DevOps" OR "Build automation") AND ("Software development" OR "Web development" OR "Software engineering")

Esta cadena se empleó en tres principales motores de búsqueda: Scopus, IEEE y EBSCO. Se aplicaron criterios de inclusión orientados a seleccionar artículos científicos en idioma inglés que abordan las pruebas automatizadas con enfoque en integración continua, considerando aquellos que reportan beneficios de la automatización y que implementan mecanismos de seguimiento del proceso de pruebas. Respecto a los criterios de exclusión se excluyeron los estudios que se limitarán al uso de pruebas manuales sin incorporar herramientas de automatización e integración continua. Como resultado, se identificaron inicialmente 1,090 artículos. Tras la eliminación de duplicados y registros irrelevantes, se obtuvieron 892 estudios. Posteriormente, mediante filtros por título, resumen y lectura completa, se redujo el conjunto a 30 artículos seleccionados para el análisis final. En la Figura 1 se muestra el proceso de ejecución de la metodología PRISMA con los resultados mencionados:

Los resultados indican que las pruebas automatizadas se aplican principalmente en proyectos de desarrollo de software y tecnologías de la información (75%), con énfasis en aplicaciones web y móviles (40%), mientras que su uso en el ámbito académico representa un 25%. En [14] se destaca la importancia de la automatización en aplicaciones web y su integración con CI/CD para mejorar la detección temprana de errores. Asimismo, el 45% de los proyectos emplea integración y entrega continua, y el 35% adopta prácticas DevOps, como se evidencia en proyectos como el de *Cisco Systems* [15].

En cuanto a herramientas de (CI/CD), Jenkins es la más utilizada (50%), seguida de GitLab CI/CD y GitHub (35%), mientras que las herramientas propietarias representan el 15%. En [16] se presenta un caso de implementación de integración continua con Jenkins, SonarQube y GitHub en el cual se diseñó, implementó y evaluó una arquitectura de integración continua.

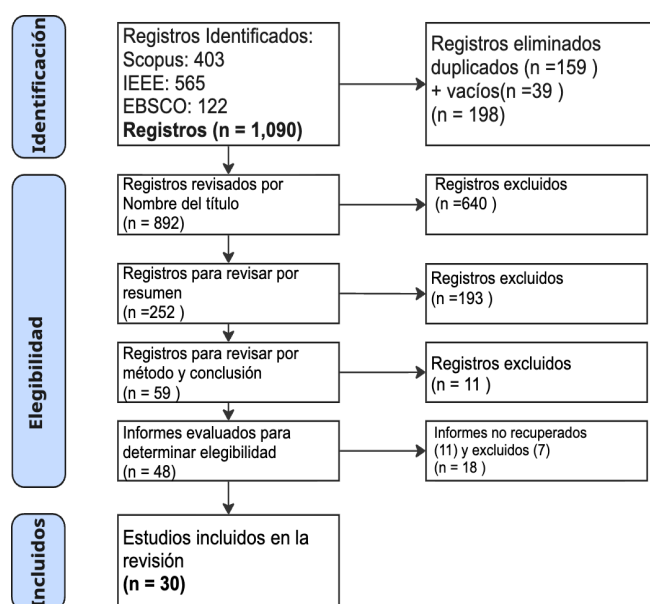


Figura 1. Proceso de metodología PRISMA.

Respecto a los tipos de pruebas en la Tabla 1, se puede apreciar que predominan las pruebas unitarias (45%), seguidas de las funcionales y *end-to-end* (35%), mientras que las pruebas no funcionales tienen menor presencia (20%). Según [17], la automatización de pruebas permite aumentar la cobertura, reducir el esfuerzo y mejorar la eficiencia del proceso.

Tabla 1. Tipos de pruebas ejecutadas en proyectos.

Áreas	Porcentaje%
Unitarias	45%
Funcionales	35%
Seguridad	5%
Rendimiento	10%
Compatibilidad	5%

Fuente: Elaboración propia.

La Tabla 2 presenta los principales beneficios identificados en los estudios analizados sobre la implementación de pruebas automatizadas. Los resultados muestran que la mejora de la calidad del software constituye el beneficio más reportado (30%), seguida de la reducción de los tiempos de desarrollo (25%). Asimismo, la detección temprana de errores y la aceleración de los ciclos de desarrollo representan cada una el 15% de los casos analizados. En menor proporción, se identificaron beneficios relacionados con la satisfacción del usuario, la reducción de costos y la trazabilidad (5% cada uno). En este sentido, el estudio de [18] destaca que la automatización integrada con la integración continua contribuye a disminuir los costos operativos y a incrementar la confiabilidad del software.

Tabla 2. Principales beneficios de las pruebas automatizadas.

Áreas	Porcentaje%
Mejora de la calidad	30%
Reducción en tiempos	25%
Detección temprana de errores	15%
Aceleración de ciclos	15%
Satisfacción del usuario	5%
Reducción de costos	5%
Trazabilidad	5%

Fuente: Elaboración propia.

En la Tabla 3 se presenta una síntesis de los estudios primarios que implementan integración continua, indicando el contexto de aplicación, la incorporación de mecanismos de gestión y seguimiento de pruebas, los tipos de pruebas automatizadas y la principal contribución de cada estudio. Los resultados muestran que la integración continua constituye una práctica ampliamente adoptada para automatizar pruebas y mejorar la calidad del software. Sin embargo, la mayoría de los estudios se centra en la automatización de uno o dos tipos de pruebas, principalmente unitarias y funcionales, sin integrar mecanismos de gestión y seguimiento del proceso de pruebas. Asimismo, se identificó una escasa incorporación del análisis estático del código dentro de soluciones integrales y una limitada aplicación de estas propuestas en el ámbito académico. Estos hallazgos evidencian una oportunidad para el desarrollo de herramientas que integren la gestión, el seguimiento y la automatización del proceso de pruebas dentro de una misma plataforma, como la propuesta presentada en este trabajo.

Tabla 3. Tabla de síntesis de estudios primarios.

ID	Contexto de aplicación	Integración Continua	Gestión y seguimiento de pruebas	Tipos de pruebas automatizadas	Principal contribución
S1 [19]	Investigación científica	✓	✗	Unitarias	Detección temprana de errores
S2 [15]	Desarrollo de software	✓	✗	Funcional / compatibilidad	Reducción en tiempos
S3 [20]	Desarrollo de software	✓	✗	Unitaria / Integración	Mejora de la calidad
S4 [21]	Desarrollo de software	✓	✗	Unitarias	Detección temprana de errores
S5 [22]	E-commerce	✓	✗	Funcional / Rendimiento	Reducción en tiempos
S6 [23]	Desarrollo de software	✓	✗	Rendimiento	Satisfacción del usuario
S7 [18]	Académica	✓	✗	Funcional / compatibilidad	Reducción en tiempos
S8 [10]	Telecomunicaciones	✓	✗	Unitarias	Reducción en tiempos
S9 [24]	Desarrollo de software	✓	✗	Unitarias	Detección temprana de errores
S10 [25]	Desarrollo de software	✓	✗	Funcional	Reducción en tiempos

Fuente: Elaboración propia. **Nota:** S1–S10 corresponden a los estudios primarios identificados en el mapeo sistemático. La referencia bibliográfica de cada estudio se encuentra en la sección de referencias.

3. Materiales y métodos

La metodología empleada para el desarrollo de la solución tecnológica propuesta es Programación Extrema (*Extreme Programming, XP*). Esta metodología se caracteriza por su fuerte énfasis en la calidad del software, lo que la hace adecuada para el presente proyecto. XP forma parte del enfoque ágil y está orientada al desarrollo de software mediante la definición de metas comunes alcanzables para el equipo de trabajo, así como la participación activa del cliente a lo largo del proceso. Asimismo, se fundamenta en principios como la comunicación, la simplicidad, la retroalimentación y la flexibilidad [26]. Dado su carácter iterativo e incremental, esta metodología permite realizar ajustes y mejoras continuas en cada una de las etapas del desarrollo. A continuación, se presenta la Figura 2 en donde se detallan las etapas del desarrollo con programación extrema del proyecto.



Figura 2. Ciclo de desarrollo de la metodología XP.

3.1. Planeación

En esta etapa se organizan las actividades clave para el desarrollo del proyecto, incluyendo el acercamiento con el cliente, el análisis del contexto y de la unidad receptora mediante técnicas de recolección de información, así como la elaboración de diagramas técnicos. Además, se realiza el análisis de necesidades y problemáticas, junto con el levantamiento de requerimientos funcionales y no funcionales de la solución.

Posteriormente se llevaron a cabo entrevistas a los principales usuarios como el jefe del departamento, líder de desarrollo e ingeniero de pruebas, con el objetivo de obtener información sobre los procesos de desarrollo y aseguramiento de la calidad. Para finalizar, la información recopilada fue analizada mediante codificación axial para identificar relaciones y categorías relevantes dentro del proceso. Algunas categorías fueron; tecnología de desarrollo, roles, usuarios, metodología de desarrollo y artefactos de pruebas.

En esta etapa se emplearon técnicas de ingeniería de procesos para modelar y analizar el proceso actual del Departamento de Desarrollo, con el propósito de identificar los actores involucrados, las actividades realizadas, el flujo de información y las herramientas utilizadas durante el proceso de pruebas. Para ello se utilizaron una gráfica rica [27], un diagrama IDEF0 (Definición de Integración para el Modelado de Funciones) [28] y un Diagrama de Actividades por Rol (RAD) [29]. La Figura 3 presenta el modelo RAD del proceso actual, el cual permitió representar la interacción entre los diferentes actores y las actividades que conforman el proceso de gestión y ejecución de pruebas.

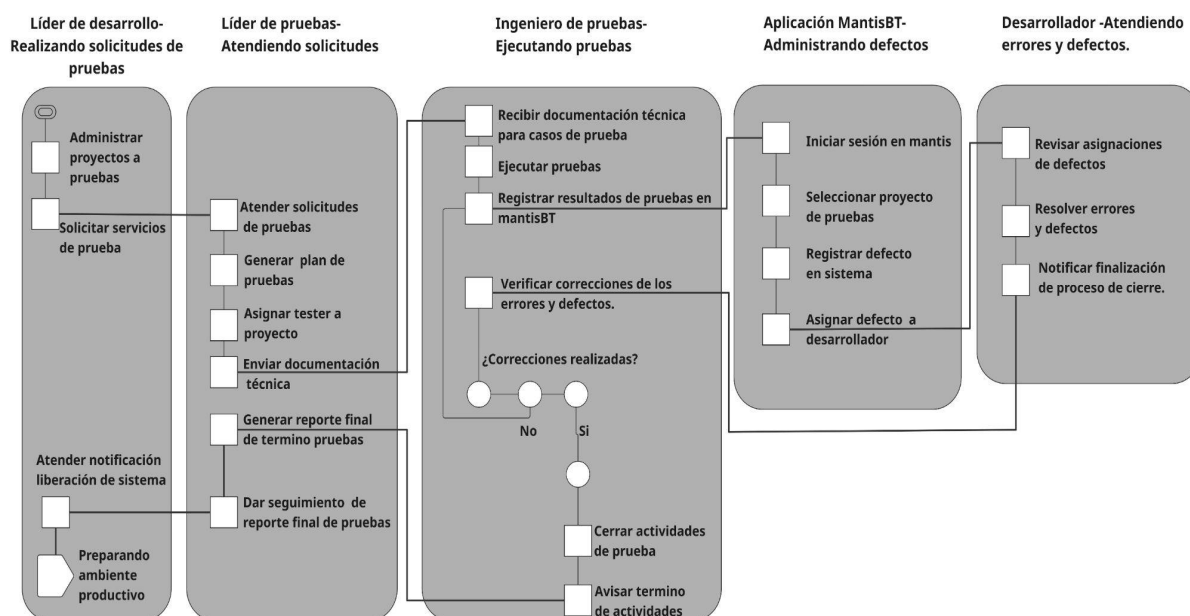


Figura 3. Diagrama de Actividades por Rol (RAD) del proceso actual de gestión y ejecución de pruebas en el departamento.

A partir del análisis del proceso se identificaron diversas limitaciones, entre las que destacan la ejecución manual de las pruebas, la ausencia de mecanismos para la gestión y el seguimiento de las actividades, la limitada trazabilidad de los resultados, la sobrecarga operativa del área de aseguramiento de la calidad y la ejecución tardía de las pruebas dentro del ciclo de desarrollo. Estas limitaciones evidenciaron la necesidad de desarrollar una solución tecnológica que integrara la gestión, el seguimiento y la automatización del proceso de pruebas.

3.2. Diseño

En esta etapa del proceso se definieron los requerimientos funcionales de la solución tecnológica propuesta. La solución fue diseñada para atender las necesidades relacionadas con el seguimiento de solicitudes de pruebas y la ejecución de pruebas automatizadas mediante integración continua. En este apartado se presentan los requerimientos funcionales y los prototipos desarrollados para la solución tecnológica. Los requerimientos funcionales se enfocan en describir las acciones y funcionalidades que un sistema debe realizar para satisfacer las necesidades del usuario o para cumplir un determinado objetivo [30]. Durante esta etapa se definieron nueve requerimientos funcionales, orientados a la aplicación web, abarcando funcionalidades como inicio de sesión, administración de usuarios, proyectos, gestión de solicitudes de pruebas y consulta de resultados.

Asimismo, para la infraestructura de integración continua se diseñaron siete requerimientos funcionales relacionados con el registro y ejecución de pruebas unitarias, funcionales y de rendimiento, así como análisis estático, consulta del *dashboard* de pruebas y envío de resultados generados durante el proceso de validación. La Tabla 4 presenta un resumen de dichos requerimientos, organizados de acuerdo con la funcionalidad principal de la aplicación web.

Tabla 4. Requerimientos funcionales de la aplicación web.

ID	Requerimiento funcional	Módulo
RF-01	Iniciar sesión en sistema	Login
RF-02	Mostrar pantalla de inicio de rol	Login
RF-03	Administrar usuarios	Catálogo
RF-04	Administrar proyectos	Proyectos
RF-05	Registrar solicitudes de pruebas	Gestión solicitudes de pruebas
RF-06	Modificar solicitudes de pruebas	Gestión solicitudes de pruebas
RF-07	Consultar solicitudes de pruebas	Gestión solicitudes de pruebas
RF-08	Agregar actualizaciones de prueba	Gestión solicitudes de pruebas
RF-09	Consultar Dashboard de pruebas	Dashboard

Fuente: Elaboración propia.

Respecto al diseño de los prototipos en la aplicación web se llevó a cabo en dos etapas. Inicialmente, se desarrollaron prototipos de baja fidelidad en papel con base en los requerimientos funcionales identificados, permitiendo representar de forma rápida la estructura del sistema y facilitar modificaciones. Posteriormente, los prototipos evolucionaron a un nivel de fidelidad media, mediante el uso de la herramienta Figma, incorporando los requerimientos definidos y las observaciones obtenidas durante las evaluaciones con los usuarios. En la Figura 4 se puede apreciar los elementos en diseño de la aplicación web y la infraestructura de integración continua.



Figura 4. Diseño de solución tecnológica.

3.3. Codificación

En la etapa de codificación se consideraron los lineamientos establecidos por el departamento de desarrollo de sistemas, los cuales definen el uso de herramientas, tecnologías e infraestructura, promoviendo principalmente el empleo de tecnologías de código abierto. Como se mencionó en la etapa de diseño, la solución desarrollada se compone de dos elementos principales. El primero corresponde a una aplicación web orientada al seguimiento de los proyectos sometidos a pruebas, en la cual se registran las solicitudes y los tipos de pruebas que serán ejecutadas. El segundo elemento consiste en una infraestructura de integración continua encargada de ejecutar de manera automatizada las pruebas sobre los proyectos seleccionados. Una vez finalizado el proceso de validación, los resultados generados pueden ser consultados directamente desde la aplicación web de seguimiento de pruebas en el módulo de dashboard en donde se presentan graficas e información detallada de las pruebas. De esta manera, ambos componentes se integran de forma coordinada para administrar, ejecutar y dar trazabilidad al proceso de pruebas de software. Además de estos dos componentes, se incorporó un sistema piloto utilizado como caso práctico para validar la solución propuesta. Sobre este sistema, la infraestructura de integración continua ejecuta de manera automatizada pruebas unitarias, análisis estático de código, pruebas funcionales y pruebas de rendimiento.

3.3.1. Aplicación Web

Para el desarrollo del sistema web de seguimiento de pruebas se utilizaron tecnologías orientadas al desarrollo *full stack*. En el *frontend* se emplearon React, TypeScript y Vite, mientras que en el *backend* se implementó una arquitectura basada en NestJS sobre Node.js. Para la gestión de datos se utilizó Prisma como ORM y PostgreSQL como sistema gestor de base de datos. En materia de seguridad, se integró Keycloak para la autenticación y autorización de usuarios mediante roles y tokens JWT. La solución sigue una arquitectura cliente-servidor, donde el *frontend* interactúa con el *backend* a través de solicitudes HTTP, permitiendo una estructura modular, escalable y con separación clara de responsabilidades. En la Figura 5, se detalla la arquitectura de la aplicación web para el seguimiento de las pruebas.

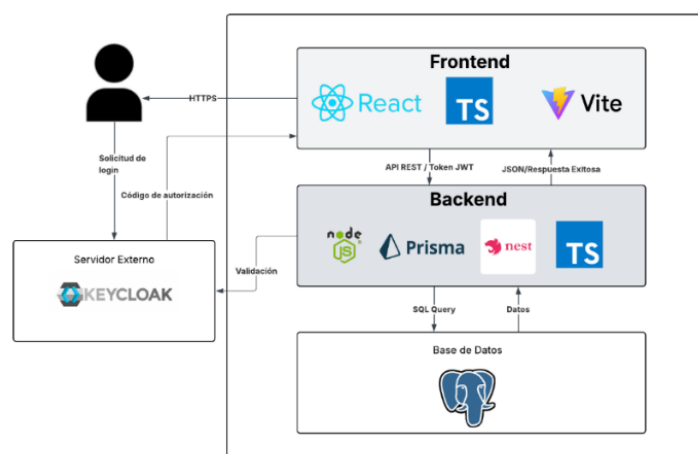


Figura 5. Arquitectura de la aplicación web.

Como resultado de la etapa de codificación, se desarrolló la aplicación web propuesta, la cual integra los módulos de autenticación, inicio, proyectos, solicitudes de pruebas, dashboard de resultados y administración de usuarios.

Antes de realizar el registro de una solicitud de prueba es necesario registrar un proyecto, el registro comprende el nombre del proyecto, área y descripción. Una vez registrado un proyecto, es posible generar una solicitud de prueba en la que se registran datos como el tipo de prueba, equipos involucrados, fechas programadas, archivos adjuntos y nivel de prioridad. Además, cada solicitud cuenta con un estado que se actualiza conforme avanza el proceso de pruebas, permitiendo dar seguimiento al progreso de la evaluación. En la Figura 6, se presenta el módulo de consulta y registro de solicitudes de pruebas.

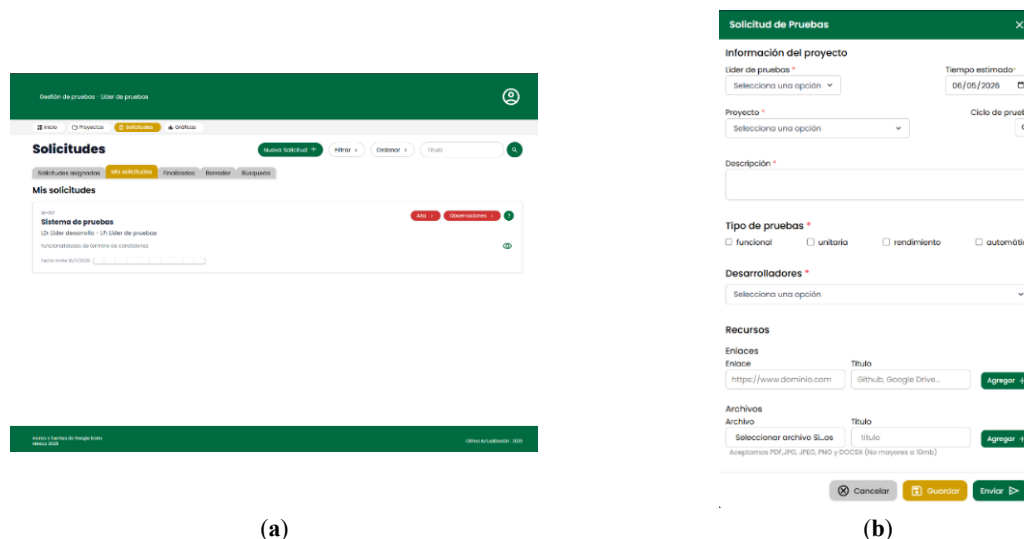


Figura 6. Módulo de solicitud de prueba: (a) consulta de solicitudes; (b) registro de solicitudes.

Por último, el sistema integra un módulo denominado Dashboard de pruebas, el cual permite consultar las pruebas automatizadas asociadas a cada solicitud de pruebas. Este Dashboard funciona como un puente de integración entre la infraestructura de pruebas automatizadas y la aplicación web, centralizando la visualización de los resultados obtenidos durante la ejecución de las pruebas. En este módulo se presentan los resultados finales del proceso de validación, indicando los casos de prueba ejecutados, aquellos que fueron aprobados y los que presentaron fallos. En la Figura 7 se muestra el módulo con los reportes para cada tipo de prueba.

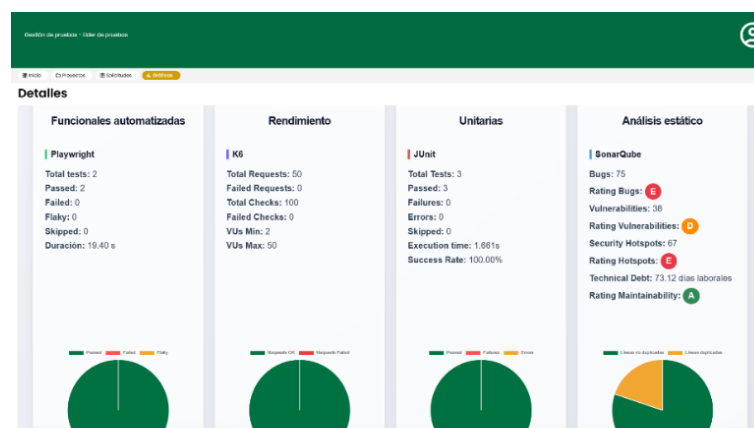


Figura 7. Reporte de pruebas automatizadas.

3.3.2. Infraestructura de integración continua con pruebas automatizadas

Para la implementación de la infraestructura de integración continua con pruebas automatizadas se utilizaron diversas herramientas y tecnologías especializadas, integradas dentro de un entorno centralizado de desarrollo, despliegue y validación de software.

GitLab CI/CD se empleó como plataforma principal de orquestación, encargándose de administrar la ejecución de todas las etapas (*stages*) del pipeline mediante runners configurados en un entorno Shell (*pruebasshellrunner*). Este agente de ejecución se encuentra alojado en una máquina virtual con sistema operativo Linux Ubuntu, configurada con 16 GB de memoria RAM y 6 procesadores (CPU), proporcionando la capacidad de procesamiento necesaria para soportar compilaciones concurrentes y pruebas automatizadas de alta demanda. Para el desarrollo y compilación del *backend* del sistema piloto se utilizaron Java (Temurin JDK 21/OpenJDK 11) y Maven, los cuales permitieron gestionar dependencias, compilar el código fuente, empaquetar el proyecto y ejecutar las pruebas unitarias.

La calidad del código se evaluó mediante SonarQube, plataforma utilizada para realizar análisis estático del código fuente, identificar vulnerabilidades, detectar errores (bugs) y calcular métricas de cobertura de pruebas. El despliegue de la aplicación se realizó sobre Apache Tomcat, utilizado como servidor web y contenedor de servlets Java, donde se ejecuta el artefacto final generado en formato .war.

En la etapa de pruebas funcionales se implementó Playwright, *framework* basado en Node.js utilizado para automatizar navegadores web Chromium y validar el comportamiento de la interfaz de usuario mediante pruebas de extremo a extremo (*End-to-End*, E2E). Para las pruebas de rendimiento y carga se utilizó k6 (xk6), herramienta de código abierto que permite simular múltiples usuarios concurrentes y evaluar la estabilidad y desempeño del sistema en distintos servicios y *endpoints*, como autenticación y consultas de información.

Asimismo, se emplearon Node.js y Python como entornos de ejecución complementarios. Node.js fue utilizado para ejecutar dependencias asociadas a Playwright y *scripts* personalizados para la generación de reportes, mientras que Python se utilizó para procesos de conversión de formatos, particularmente de archivos XML a HTML.

Finalmente, se utilizaron protocolos y herramientas del sistema como SSH, SCP y Bash, los cuales permitieron automatizar la transferencia segura de archivos, la administración de artefactos y el reinicio de servicios dentro del servidor de despliegue. En la Figura 8 se muestra la arquitectura de la infraestructura de integración continua con pruebas automatizadas.

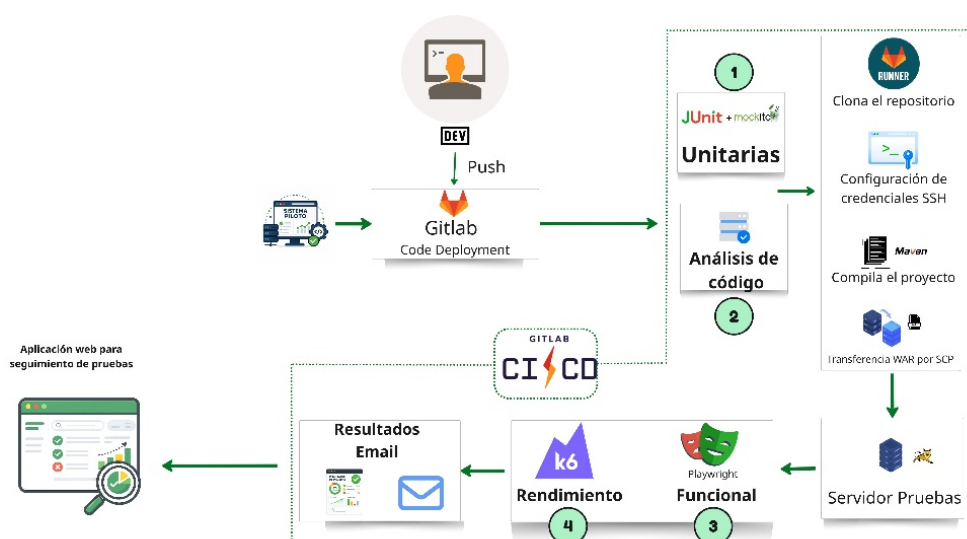


Figura 8. Arquitectura de infraestructura de integración continua.

3.3.3. Implementación y ejecución

En este apartado se especifica el flujo de trabajo automatizado que sostiene la solución de integración continua, compuesto por distintas fases que van desde la preparación del entorno hasta la notificación de resultados.

- Preparación y compilación. El flujo inicia con la etapa de preparación (*artifact-dir*), donde el sistema crea una estructura de directorios dinámica basada en la fecha y el ID del pipeline para almacenar los artefactos generados. Posteriormente, en la etapa de pruebas unitarias (*unit-test*), el sistema configura las versiones correctas de Java y Maven, compila el código, ejecuta los *tests* del *backend* y almacena los reportes en formato XML.
- Calidad de código y despliegue. En la etapa de revisión (*code-review*), el código es enviado a SonarQube para su análisis. Mediante su API REST, el pipeline extrae “*badges*” (insignias visuales) que reflejan el estado de calidad del proyecto. Una vez validado, se procede a la etapa de empaquetado y despliegue (*build-and-deploy*). Aquí, Maven empaqueta la aplicación excluyendo las pruebas, y el artefacto .war resultante se envía por SCP al servidor remoto. Mediante comandos SSH, se detiene el servicio Tomcat, se limpia el despliegue anterior, se posiciona el nuevo ejecutable y se reinicia el servidor.

- Pruebas funcionales y de rendimiento. Con la aplicación desplegada y utilizando los recursos de la máquina virtual Ubuntu del runner, el pipeline ejecuta pruebas automatizadas en paralelo: pruebas funcionales (*functional-test*): utilizando Playwright, se levantan instancias de navegadores y se navega por el flujo de la aplicación para confirmar que las interfaces interactúan correctamente con el *backend*. Pruebas de Rendimiento (*performance-test*): A través de scripts de k6, se simulan usuarios concurrentes consultando módulos críticos (Login, Consulta) para generar métricas de rendimiento y tiempos de respuesta.
- Consolidación y distribución de reportes. Toda la información recolectada entra a una fase de transformación. Los reportes XML de pruebas unitarias se convierten a formatos legibles en HTML (*xml-to-html*). Luego, un script en Node.js (*report*) unifica los resultados funcionales, de rendimiento y de calidad en un único reporte maestro consolidado. Finalmente, la etapa de notificación (*send-report*, *publish-report*) comprime estos archivos en formato ZIP y los distribuye a los líderes técnicos mediante correo electrónico, además de publicarlos en un repositorio Git exclusivo para el control de versiones de reportes. Estos reportes son preparados para ser consultados por la aplicación web para el seguimiento de pruebas en tiempo real.
- Finalización del proceso. Al finalizar el proceso de todas las pruebas, es posible liberar el sistema a un ambiente de producción de manera automática si así se requiere.
- La Figura 9 presenta el *pipeline* de integración continua implementado para el sistema piloto, el cual automatiza etapas como pruebas unitarias, análisis estático de código, despliegue, pruebas funcionales, pruebas de rendimiento y generación de reportes. Esta arquitectura permite realizar validaciones continuas y mejorar la trazabilidad del proceso de aseguramiento de la calidad.

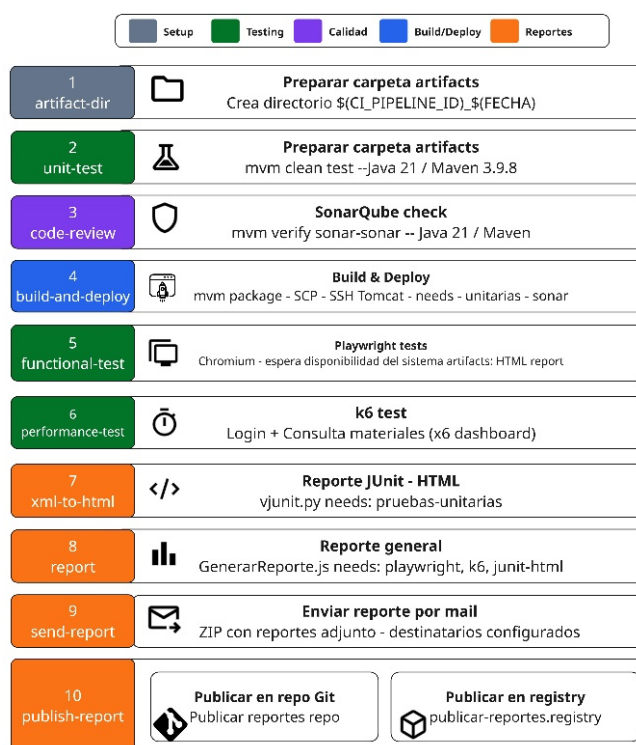


Figura 9. Pipeline de pruebas automatizadas.

4. Proceso de evaluación y resultados

El presente apartado describe la evaluación realizada a la solución tecnológica, cuyo propósito es optimizar el proceso de gestión y ejecución de pruebas de software. La evaluación tuvo como objetivo medir variables relacionadas con la utilidad percibida, facilidad de uso, usabilidad, experiencia de usuario y agilización del proceso, mediante una comparación entre el proceso manual actual y el uso de la solución tecnológica propuesta.

La metodología de evaluación se dividió en dos etapas. En la primera, 7 participantes analizaron y describieron el proceso manual actual relacionado con el registro de proyectos, seguimiento de pruebas y consulta de resultados. En la segunda etapa, los mismos usuarios ejecutaron tareas específicas dentro del sistema, tales como registro de proyectos, generación de solicitudes de pruebas, seguimiento de avances y consulta de *dashboards* de resultados.

Para la recolección de datos se emplearon instrumentos cuantitativos y cualitativos, incluyendo los cuestionarios SUS (*System Usability Scale*) [31] y TAM (*Technology acceptance model*) [32], entrevistas, cuestionarios de apreciación general y la técnica Think Aloud. En la evaluación participaron siete usuarios pertenecientes a distintos roles del departamento: jefe de departamento, líderes de desarrollo y líderes de pruebas. La muestra fue seleccionada de manera intencional debido a que los participantes representan los roles clave involucrados directamente en el proceso de pruebas.

Con el propósito de fortalecer el análisis de los resultados, se calcularon estadísticos descriptivos y la consistencia interna de los instrumentos mediante el coeficiente Alfa de Cronbach. El cuestionario TAM presentó una excelente confiabilidad ($\alpha=0.934$), obteniendo valores de $\alpha=0.840$ para la dimensión de utilidad percibida y $\alpha=0.958$ para facilidad de uso percibida. Por su parte, el cuestionario SUS alcanzó un Alfa de Cronbach de $\alpha=0.627$, valor aceptable considerando el tamaño reducido de la muestra y la naturaleza del instrumento. Asimismo, la aplicación obtuvo una puntuación promedio SUS de 87.14 (DE=7.96), lo que indica un nivel alto de usabilidad. Las dimensiones de utilidad percibida y facilidad de uso alcanzaron medias de 6.02 y 6.24, respectivamente, en una escala de siete puntos, evidenciando una percepción favorable por parte de los participantes. En la Tabla 5, se presentan los resultados mencionados.

Tabla 5. Estadísticos descriptivos y confiabilidad de los instrumentos.

Instrumento	Media	Mediana	DE	Alfa de Cronbach
SUS	87.14	87.50	7.96	0.627
TAM - Utilidad percibida	6.02	5.83	0.68	0.840
TAM - facilidad de uso	6.24	6.50	0.77	0.958

Fuente: Elaboración propia.

En términos de eficiencia, los resultados evidenciaron una reducción significativa en los tiempos de ejecución de las actividades. Mientras que el proceso manual podría requerir entre 5 y 14 días para registrar proyectos, dar seguimiento y consultar resultados, con el sistema estas actividades pudieron realizarse en un rango de 4 a 13 minutos.

De igual manera, las pruebas automatizadas ejecutadas mediante integración continua redujeron considerablemente los tiempos de ejecución respecto al proceso manual. En proyectos de integración continua y pruebas automatizadas, diversos estudios reportan reducciones significativas en los tiempos de ejecución, comúnmente entre un 80% y 95% respecto a procesos manuales tradicionales. Los resultados obtenidos en la infraestructura muestran porcentajes similares, evidenciando un impacto considerable en la eficiencia operativa del proceso de pruebas. En la Tabla 6, se pueden apreciar los resultados de eficiencia en las pruebas automatizadas.

Tabla 6. Resultados de eficiencia en pruebas automatizadas.

Tipo prueba	Scripts	Tiempo manual	Tiempo automatizado	Ahorro tiempo
Unitaria	51 scripts	3 horas	1 minuto	99.44%
Funcional	177 scripts	25 horas	14 minutos	99.07%
Rendimiento	2 scripts	5 horas	15 minutos	95.00%
Análisis estático	5 scripts	10 horas	20 minutos	96.67%

Fuente: Elaboración propia

La Tabla 7 presenta la comparación de indicadores objetivos antes y después de implementar la automatización de pruebas funcionales. Los resultados muestran un incremento en la cobertura funcional del sistema, una reducción significativa en el tiempo de ejecución de las pruebas y un aumento en la capacidad para detectar defectos durante la integración continua.

Tabla 7. Comparación de métricas objetivas antes y después de implementar la solución en pruebas automatizadas.

Métrica	Antes	Después	Mejora observada
Cobertura funcional de pruebas de casos de prueba	0%	45%	Incremento en la cobertura mediante pruebas automatizadas en módulos del sistema.
Tiempo promedio de ejecución	1.79 días	50 minutos	Reducción significativa del tiempo de ejecución.
Pruebas automatizadas	0	177	Automatización del proceso de pruebas funcionales.
Defectos funcionales detectados automáticamente	20	65	Incremento en la capacidad de detección temprana de defectos mediante pruebas automatizadas.

Fuente: Elaboración propia.

Como parte de la implementación de la plataforma se integró SonarQube al pipeline de integración continua para incorporar análisis estático del código fuente. La primera ejecución permitió establecer una línea base de calidad del proyecto mediante indicadores objetivos relacionados con la confiabilidad, seguridad y mantenibilidad del software.

Los resultados iniciales, presentados en la Tabla 8, muestran que el proyecto no cumplía con los criterios definidos por el Quality Gate, lo que evidencia la utilidad del análisis estático para identificar de manera temprana oportunidades de mejora antes del despliegue del software.

Tabla 8. Línea base de indicadores de calidad obtenidos mediante análisis estático con SonarQube.

Indicador de calidad	Valor inicial	Descripción
Defectos encontrados	72	Defectos potenciales detectados mediante análisis estático que pueden ocasionar comportamientos incorrectos del software.
Vulnerabilidades	16	Posibles debilidades de seguridad identificadas en el código fuente que podrían comprometer la aplicación si no son corregidas.
Code smell	1,936	Problemas relacionados con la mantenibilidad del código que incrementan su complejidad y dificultan su evolución.
Duplicación de código	12.3%	Porcentaje de código duplicado identificado dentro del proyecto.
Puntos críticos de seguridad	25	Fragmentos de código que requieren revisión manual debido a que podrían representar riesgos de seguridad dependiendo de su implementación.
Quality Gate	Fallado	El proyecto no cumple inicialmente con los criterios mínimos de calidad definidos en SonarQube.

Fuente: Elaboración propia.

5. Discusión

Los resultados obtenidos confirman que la integración continua y la automatización de pruebas contribuyen a mejorar la eficiencia del proceso de aseguramiento de la calidad del software, coincidiendo con los estudios identificados en el mapeo sistemático de la literatura. A diferencia de la mayoría de los trabajos analizados y de las herramientas comerciales de gestión de pruebas, la solución propuesta integra en una misma plataforma la gestión y seguimiento de solicitudes de pruebas, la automatización de pruebas funcionales y el análisis estático del código mediante SonarQube, adaptándose al flujo de trabajo específico de una universidad pública. Los resultados evidencian una reducción en los tiempos de ejecución, un incremento en la cobertura funcional y una mayor capacidad para detectar defectos durante el proceso de integración continua.

No obstante, el estudio presenta algunas limitaciones, ya que la evaluación se realizó en un único departamento de una universidad pública y con una muestra de siete participantes, lo que limita la generalización de los resultados. Asimismo, el análisis inicial mediante SonarQube permitió establecer una línea base objetiva de la calidad del código, identificando bugs potenciales, vulnerabilidades, problemas de mantenibilidad y duplicación de código, los cuales constituyen una base para el seguimiento de la mejora continua. Durante la implementación

de la solución se identificó que la adopción de prácticas DevOps en instituciones públicas requiere aprovechar herramientas de código abierto y adaptar gradualmente los procesos existentes para minimizar el impacto sobre la infraestructura y los recursos disponibles. En este sentido, la integración de GitLab CI/CD, SonarQube, Playwright y k6 demostró que es posible implementar automatización de pruebas y análisis continuo sin realizar inversiones significativas en software propietario. Asimismo, uno de los principales retos técnicos consistió en integrar y normalizar los resultados generados por las distintas herramientas para presentarlos de forma unificada dentro de la plataforma, facilitando el seguimiento del proceso de pruebas y la toma de decisiones.

Como trabajo futuro, se plantea ampliar la evaluación en otros proyectos e instituciones, así como dar seguimiento a la evolución de las métricas de calidad obtenidas mediante el análisis estático del código. Estas experiencias evidencian que la adopción gradual de prácticas DevOps apoyadas en herramientas de código abierto constituye una alternativa viable para fortalecer el aseguramiento de la calidad del software en instituciones públicas con recursos limitados.

6. Conclusiones

Como conclusión, el análisis del proceso actual permitió identificar diversas problemáticas relacionadas con el seguimiento y la ejecución de pruebas de software, lo que condujo al diseño e implementación de una solución tecnológica basada en integración continua para mejorar la gestión, trazabilidad y automatización del proceso de pruebas en una universidad pública. La propuesta integra una aplicación web para la administración de solicitudes de pruebas con una infraestructura de integración continua que automatiza la ejecución de pruebas y el análisis estático del código.

Los resultados obtenidos demuestran que la solución mejora la eficiencia del proceso de aseguramiento de la calidad, al reducir significativamente los tiempos de ejecución, fortalecer el seguimiento de los proyectos e incorporar mecanismos automáticos para la validación continua del software. Asimismo, la evaluación realizada evidenció una aceptación favorable por parte de los usuarios, confirmando la utilidad y facilidad de uso de la plataforma en el contexto institucional donde fue implementada.

Finalmente, este trabajo demuestra la viabilidad de integrar herramientas de código abierto dentro de una infraestructura institucional para fortalecer los procesos de desarrollo y aseguramiento de calidad del software. Como trabajo futuro, se plantea ampliar la cobertura de pruebas automatizadas, dar seguimiento a la evolución de las métricas de calidad obtenidas mediante análisis estático y extender la solución a otros proyectos e instituciones con características similares.

En este sentido, la investigación permitió responder la pregunta planteada, demostrando que una solución tecnológica basada en integración continua contribuye a mejorar el seguimiento, la trazabilidad y la automatización del proceso de pruebas de software, además de proporcionar indicadores objetivos que fortalecen el aseguramiento de la calidad dentro de un entorno universitario.

7. Agradecimientos

Se agradece a todas las personas que colaboraron en el desarrollo del presente proyecto, especialmente a los tutores, asesores, sinodales y participantes del departamento involucrados en las distintas actividades de evaluación, pruebas y validación del sistema. Asimismo, se reconoce el apoyo brindado por la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) al primer autor para la realización de este trabajo.

8. Referencias

- [1] Sommerville, I. (2011). *Ingeniería de software* (9.ª Ed.). Pearson Educación.
- [2] Ruk, S. A., Khan, M. F., Khan, S. G., Zia, S. M. (2019). *A survey on adopting agile software development: Issues and its impact on software quality*. IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS), Kuala Lumpur, Malaysia. <https://doi.org/10.1109/ICETAS48360.2019.9117324>
- [3] Yakovyna, V., Seniv, M., Symets, I. (2020). *The relation between software development methodologies and factors affecting software reliability*. IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), Zbarazh, Ukraine. <https://doi.org/10.1109/CSIT49958.2020.9321937>

- [4] Hazzan, O., Dubinsky, Y. (2006). *Can diversity in global software development be enhanced by agile software development?* Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner (GSD), Shanghai China. <https://doi.org/10.1145/1138506.1138520>
- [5] Smada, D., Rotuna, C., Boncea, R., Petre, I. (2018). Automated Code Testing System for Bug Prevention in Web-based User Interfaces. *Informatica Economica*, 22 (3), 23–32. <https://doi.org/10.12948/issn14531305/22.3.2018.03>
- [6] Zhao, Y., Hu, Y., Gong, J. (2021). *Research on international standardization of software quality and software testing*. IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall), Xi'an, China. <https://doi.org/10.1109/ICISFall51598.2021.9627426>
- [7] International Software Testing Qualifications Board. (s.f.). *ISTQB glossary of testing terms*. <https://glossary.istqb.org/>
- [8] Donvir, A. (2024). *A comparative analysis of JavaScript unit and end-to-end testing frameworks: Enhancing quality assurance in modern web applications*. IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN), Indore, India. <https://doi.org/10.1109/CICN63059.2024.10847365>
- [9] Wettinger, J., Breitenbücher, U., Leymann, F. (2014). DevOpsSlang – Bridging the gap between development and operations. En M. Villari, W. Zimmermann, K.-K. Lau (Eds.), *Service-oriented and cloud computing* (pp. 79–91). Springer. https://doi.org/10.1007/978-3-662-44879-3_8
- [10] Marijan, D., Liaaen, M., Sen, S. (2018). *DevOps improvements for reduced cycle times with integrated test optimizations for continuous integration*. IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan. <https://doi.org/10.1109/COMPSAC.2018.00012>
- [11] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., Filkov, V. (2015). *Quality and productivity outcomes relating to continuous integration in GitHub*. Proceedings of the 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2015), Bergamo, Italy. <https://doi.org/10.1145/2786805.2786850>
- [12] Kitchenham, B., Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. EBSE Technical Report EBSE-2007-01. Keele University/University of Durham. https://www.elsevier.com/_data/promis_misc/525444systematicreviewsguide.pdf
- [13] PRISMA. (2020). *PRISMA flow diagram*. <https://www.prisma-statement.org/prisma-2020-flow-diagram>
- [14] Naveen, B., Grandhi, J. K., Lasya, K., Reddy, E. M., Srinivasu, N., Bulla, S. (2023). *Efficient automation of web application development and deployment using Jenkins: A comprehensive CI/CD pipeline for enhanced productivity and quality*. IEEE International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), Erode, India. <https://doi.org/10.1109/ICSSAS57918.2023.10331631>
- [15] Seth, N., Khare, R. (2015). *ACI (automated continuous integration) using Jenkins: Key for successful embedded software development*. IEEE 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS), Chandigarh, India. <https://doi.org/10.1109/RAECS.2015.7453279>
- [16] Baitha, S., Soorya, V., Kothari, O., Rajagopal, S. M., Panda, N. (2024). *Streamlining software development: A comprehensive study on CI/CD automation*. IEEE 4th International Conference on Sustainable Expert Systems (ICSES), Kaski, Nepal. <https://doi.org/10.1109/ICSES63445.2024.10763207>
- [17] Deshpande, A., Veenadevi, S. V., Aleti, S. (2021). *Test automation and continuous integration using Jenkins for smart card OS*. IEEE 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India. <https://doi.org/10.1109/ICCCNT51525.2021.9580021>
- [18] Gota, L., Gota, D., Miclea, L. (2020). *Continuous integration in automation testing*. IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania. <https://doi.org/10.1109/AQTR49680.2020.9129990>
- [19] Elmsheuser, J., Krasznahorkay, A., Obreshkov, E., Undrus, A. (2017). A roadmap to continuous integration for ATLAS software development. *Journal of Physics: Conference Series*, 898 (7), 1-5. <https://doi.org/10.1088/1742-6596/898/7/072009>
- [20] Siziú Nery, G., da Costa, D. A., Kulesza, U. (2019). *An empirical study of the relationship between continuous integration and test code evolution*. IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA. <https://doi.org/10.1109/ICSME.2019.00075>

- [21] Abdulla, H. H. H. A., Albalooshi, F. A. (2023). *Automated testing for DevOps in GitHub environment: A comprehensive analysis*. International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA), Surabaya, Indonesia.
<https://doi.org/10.1109/ICAMIMIA60881.2023.10427702>
- [22] Debroy, V., Brimble, L., Yost, M., Erry, A. (2018). *Automating web application testing from the ground up: Experiences and lessons learned in an industrial setting*. IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), Västerås, Sweden. <https://doi.org/10.1109/ICST.2018.00042>
- [23] Arachchi, S. A. I. B. S., Perera, I. (2018). *Continuous integration and continuous delivery pipeline automation for agile software project management*. IEEE Moratuwa Engineering Research Conference (MERCon), Moratuwa, Sri Lanka. <https://doi.org/10.1109/MERCon.2018.8421965>
- [24] Beller, M., Gousios, G., Zaidman, A. (2017). *Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub*. IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina. <https://doi.org/10.1109/MSR.2017.62>
- [25] Winkler, D., Meixner, K., & Biffl, S. (2018). *Towards flexible and automated testing in production systems engineering projects*. IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy. <https://doi.org/10.1109/ETFA.2018.8502650>
- [26] Beck, K., Andres, C. (2005). *Extreme programming explained: Embrace change* (2nd Ed.). Addison-Wesley Professional.
- [27] Checkland, P., Scholes, J. (1990). *Soft systems methodology in action*. John Wiley & Sons.
- [28] Hurenko, O., Bohdanov, I., Tsybuliak, N., Lopatina, H., Suchikova, Y., Popova, A. (2023). *Development of an inclusive educational environment in higher education institutions: A project approach using IDEF0*. IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), Lviv, Ukraine. <https://doi.org/10.1109/CSIT61576.2023.10324022>
- [29] Ould, M. A. (2005). *Business process management: A rigorous approach*. British Computer Society.
- [30] Wiegers, K. E., Beatty, J. (2013). *Software requirements* (3rd Ed.). Microsoft Press.
- [31] Brooke, J. (1996). SUS: A “quick and dirty” usability scale. En P. W. Jordan, B. Thomas, B. A. Weerdmeester, A. L. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). Taylor & Francis
- [32] Davis, F. D. (1993). User acceptance of information technology: System characteristics, user perceptions and behavioral impacts. *International Journal of Man-Machine Studies*, 38 (3), 475–487.
<https://doi.org/10.1006/imms.1993.1022>