ASEGURAMIENTO DE CALIDAD EN EL DESARROLLO DE SOFTWARE: UN MODELO DE IMPLEMENTACIÓN BASADO EN PRUEBAS AUTOMATIZADAS Y MEJORES PRÁCTICAS

Trabajo presentado para optar al título de Tecnólogo Superior en Desarrollo de Software

Proyecto de grado presentado por:

Isaac Sebastián Guerra Chamorro

Kevin Mauricio Quito Ayavaca.

Carrera: Desarrollo de Software

Tutor académico: Ing. Nancy Eras Eras.

Cuenca, 25 de febrero de 2025

INSTITUTO DE TECNOLOGIAS SUDAMERICANO
GENUINA EDUCACION SUPERIOR
ESTABLECIDO EN 1995

DERECHOS DE AUTOR

INSTITUTO DE TECNOLOGIAS SUDAMERICANO
GENUINA EDUCACION SUPERIOR
ESTABLECIDO EN 1995

CARRERA DE DESARROLLO DE SOFTWARE

CERTIFICACIÓN DEL TUTOR

Aprobación del Trabajo de Titulación

Doy fe que el trabajo desarrollado por el/la/los estudiantes: GUERRA CHAMORRO ISAAC SEBASTIÁN, QUITO AYAVACA KEVIN MAURICIO, con el título "ASEGURAMIENTO DE CALIDAD EN EL DESARROLLO DE SOFTWARE: UN MODELO DE IMPLEMENTACIÓN BASADO EN PRUEBAS AUTOMATIZADAS Y MEJORES PRÁCTICAS", cumple con los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del jurado examinador que se designe.

Atentamente,

NANCY MARIBEL ERAS ERAS

C.I 0107424657

INSTITUTO DE TECNOLOGIAS SUDAMERICANO
GENUINA EDUCACION SUPERIOR
ESTABLECIDO EN 1995

DECLARACIÓN DE AUTORÍA DEL TRABAJO

Yo, **QUITO AYAVACA KEVIN MAURICIO**, estudiante del **Instituto Tecnológico Superior Particular Sudamericano** de la ciudad de Cuenca - Ecuador, que cursó la Tecnología en **DESARROLLO DE SOFTWARE,** declaro en forma libre y voluntaria que la presente investigación que versa sobre **"ASEGURAMIENTO DE CALIDAD EN EL DESARROLLO DE SOFTWARE: UN MODELO DE IMPLEMENTACIÓN BASADO EN PRUEBAS AUTOMATIZADAS Y MEJORES PRÁCTICAS"** así como las expresiones vertidas en la misma, son autoría de la compareciente, quien ha realizado en base a recopilación bibliográfica, consultas de internet y consultas de campo.

En consecuencia, asumo la responsabilidad de la originalidad de la misma y el cuidado al remitirme a las fuentes bibliográficas respectivas para fundamentar el contenido expuesto.

Atentamente,

Kevin Quito

QUITO AYAVACA KEVIN MAURICIO

Cédula: 0107099640

INSTITUTO DE TECNOLOGIAS SUDAMERICANO
GENUINA EDUCACIÓN SUPERIOR
ESTABLECIDO EN 1995

DECLARACIÓN DE AUTORÍA DEL TRABAJO

Yo, **GUERRA CHAMOROO ISAAC SEBASTIÁN**, estudiante del **Instituto Tecnológico Superior Particular Sudamericano** de la ciudad de Cuenca - Ecuador, que cursó la Tecnología en **DESARROLLO DE SOFTWARE,** declaro en forma libre y voluntaria que la presente investigación que versa sobre **"ASEGURAMIENTO DE CALIDAD EN EL DESARROLLO DE SOFTWARE: UN MODELO DE IMPLEMENTACIÓN BASADO EN PRUEBAS AUTOMATIZADAS Y MEJORES PRÁCTICAS"** así como las expresiones vertidas en la misma, son autoría de la compareciente, quien ha realizado en base a recopilación bibliográfica, consultas de internet y consultas de campo.

En consecuencia, asumo la responsabilidad de la originalidad de la misma y el cuidado al remitirme a las fuentes bibliográficas respectivas para fundamentar el contenido expuesto.

Atentamente,

GUERRA CHAMORRO ISAAC SEBASTIÁN

Cédula: 0106607534

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

ACRÓNIMOS

QA *Quality Assurance*

SDLC *Software Development Life Cycle*

LCD *Liquid Crystal Display*

OOP *Object Oriented Programming*

VCS *Version-control systems*

TFS *Team Foundation Server*

PMI *Project Management Institute*

DevOps *Development and Operations*

GQM *Goal Question Metric*

TDD *Test Driven Development*

SQL *Structured Query Language*

CI *Continuous Integration*

CD *Continuous Delivery*

ISC *Internet Systems Consortium*

WIP *Work In Progress*

HTML *Hypertext Markup Language*

RESUMEN

El trabajo de investigación desarrollado abarca la integración de prácticas avanzadas de control de calidad en el entorno de desarrollo del Instituto Sudamericano. El planteamiento presentado contempla la elaboración de un marco metodológico con validaciones sistematizadas y directrices para potenciar los flujos de trabajo técnico con los desarrolladores. La metodología que se implemento fue *Quality Assurance* (QA), destacado por incorporar principios de gestión ágil junto con protocolos estructurados de verificación. El proceso de validación para las pruebas unitarias y automatizadas, se incorporo dos herramientas; Cypress un herramienta especializada en la automatización de pruebas y GitLab una herramienta para la gestión de código e integración continua. Estas dos herramientas facilitó la verificación sistemática de componentes, el seguimiento de versiones nuevas y la ejecución de pruebas automatizadas. Al implementar QA, esto contempló los espacios de prueba, el diseño de validaciones unitarias y de integración, así como el establecimiento de flujos automatizados de integración y despliegue continuo. La aplicación de está metodología y herramientas demostrando una optimización al código generado, proporcionando al equipo de investigación herramientas efectivas para la identificación temprana de errores, bugs o fallos y el mantenimiento de la aplicación realizada por el equipo de desarrollo.

Palabras clave: Pruebas Unitarias,Pruebas Automatizadas, Cypress, GitLab, Integración Continua, SonarQube, Control de Calidad, Kanban.

ABSTRACT

The research work encompasses the integration of advanced quality control practices in the development environment of Instituto Sudamericano. The presented approach contemplates the elaboration of a methodological framework with systematic validations and guidelines to enhance technical workflows with developers. The implemented methodology was QA, distinguished by incorporating agile management principles along with structured verification protocols. The validation process for unit and automated testing incorporated two tools; Cypress, a specialized tool in test automation, and GitLab, a tool for code management and continuous integration. These two tools facilitated systematic component verification, new version tracking, and automated test execution. When implementing QA, this included test environments, the design of unit and integration validations, as well as the establishment of automated integration and continuous deployment flows. The application of this methodology and tools demonstrated optimization of the generated code, providing the research team with effective tools for early identification of errors, bugs, or failures, and maintenance of the application developed by the development team. **Keywords:** Unit Testing, Automated Testing, Cypress, GitLab, Continuous Integration, SonarQube, Quality Control, Kanban.

Dedicatoria

Primero, agradecemos a Dios, por ser nuestra fuerza y apoyo en cada momento.

A cada uno de los miembros de nuestras amadas familias, por su amor y confianza, que han acompañado todo este proceso. Gracias por creer en nosotros y por ser nuestra motivación en todo este tiempo.

A cada uno de nuestros apreciados docentes, por compartir sus valiosas enseñanzas y mostrarnos siempre como ir más allá de lo esperado. Gracias a eso entendemos lo que somos capaces de lograr.

A nuestra tutora de tesis, la Ing. Nancy Eras, por guiarnos en este proyecto. Sus cualidades y apoyo nos han permitido mejorar paso a paso cada día.

Para terminar, dedicamos este trabajo al Club de Programación, un espacio que nos ha permitido destacar, aprender y conocer nuevas habilidades que nos impulsan a dejar nuestra huella en el futuro de la tecnología y a seguir mejorando.

Los Autores

INTRODUCCIÓN

En el desarrollo de software existen herramientas de análisis de código de forma automática, la mantenibilidad del software es uno de los atributos de calidad externos fundamentales y está reconocida como un área de investigación preocupante en la ingeniería de software. Lo que logra proporcionar una mejora significativa del código para permitir a los desarrolladores detectar vulnerabilidades y fallas de manera más eficiente para cumplir con las expectativas de los usuarios y asegurar el éxito de un producto. Se analiza los conjuntos de métricas para minimizar el entorno de errores, bugs o cualquier condición fallida en el software. Es esencial predecir las fallas del software antes del proceso de lanzamiento, ya que al momento de probar y querer reparar el software sería costoso y la falta de tiempo que tendría.

En cada fase del SDLC, la calidad no es solo una característica adicional, es una prioridad que garantiza que el producto final sea funcional y seguro. Varias herramientas detectan posibles problemas, lo que contribuye a un desarrollo más eficiente, profesional y enfocado a garantizar software sostenible. La solución de QA automatizado es detectar las deficiencias o problemas que pueda tener un software o producto antes de que el usuario lo pruebe Ozkaya (2021).

El QA juega un papel central en la entrega final del software y en el cumplimiento de la satisfacción del cliente. Esta disciplina permite un ahorro en costes a largo plazo, ya que los errores se detectan y corrigen antes de que se vuelvan críticos y costosos de solucionar. Para una compañía, contar con un equipo o departamento de QA puede traducirse en mayores beneficios y en un crecimiento sostenido, ya que asegura que el producto lanzado al mercado es de alta calidad. La excelencia en calidad, además, fomenta la lealtad del cliente, creando una relación de confianza y asegurando que el usuario final tenga una experiencia positiva Mishra y Otaiwi (2020).

Objetivos Completos de la Investigación

Objetivo General

Desarrollar un modelo de implementado para el aseguramiento de calidad en el desarrollo de software basado en pruebas de QA Testing y mejores prácticas para la aplicación del Instituto Sudamericano.

Objetivos específicos

- Examinar estudios y artículos relevantes sobre QA en el SDLC para identificar estrategias efectivas que permitan implementar QA en cada fase del desarrollo, consolidando estas estrategias en una guía práctica para el equipo de desarrollo.
- Desarrollar un plan metodológico que integre prácticas de calidad, incluyendo la implementación de pruebas automatizadas y la capacitación del equipo en las mejores prácticas de QA.
- Implementar el uso de herramientas de automatización QA para mejorar la calidad del código, facilitando la detección temprana de errores y optimizando el proceso de desarrollo mediante la creación de una suite de pruebas manuales y automatizadas.
- Realizar una auditoría de calidad del software del equipo, identificando debilidades en el proceso de desarrollo y elaborando un informe que detalle las áreas problemáticas, acompañado de recomendaciones basadas en la validación por expertos en el campo.

Preguntas de investigación

1. ¿Cómo influye la implementación de QA automatizado en la calidad del producto por parte del cliente?
2. ¿Qué herramientas y técnicas son efectivas para garantizar la calidad del software en diferentes tipos de proyectos?
3. ¿Qué tan importante es la implementación del QA automatizado para los resultados generales de un proyecto?

4. ¿Cómo ayuda el QA automatizado a las empresas a optimizar sus procesos y reducir costos?

   El QA Testing ayuda a las empresas de con varias formas prácticas. Al automatizar las pruebas del software, los equipos pueden trabajar más rápido y con menos errores. Esto se traduce en ahorros importantes, ya que detectar problemas temprano es mucho más barato que corregirlos cuando el producto ya está en uso, mientras mas tiempo existe un fallo mas va a costar solucionarlo. Los desarrolladores se benefician al recibir retroalimentación rápida sobre sus cambios. Pueden ver de inmediato si algo se rompió, en lugar de esperar días por pruebas manuales. Esto les permite entregar mejores resultados y sentirse más seguros de su trabajo, priorizando buen código y buenas prácticas. Dentro de las empresas, la ventaja principal es poder lanzar productos de mejor calidad en menos tiempo, lo cual da buena reputación y clientes satisfechos. Aunque implementar QA automatizado requiere una inversión inicial, los beneficios a largo plazo son claros: primero menos costos operativos, segundo equipos más productivos y tercero productos más confiables. Es una herramienta importante que ayuda tanto a los desarrolladores como a la empresa a alcanzar sus objetivos de calidad y eficiencia.

Justificación

La automatización del aseguramiento de la calidad en software tiene como factor determinante mejorar entornos de desarrollo donde el tiempo es un factor de prioridad. Las empresas se ven afectadas a lanzar proyectos rápidamente sin poner en cuidado la calidad del software. El QA se implementó para ofrecer sus herramientas en ayudar a detectar y corregir errores antes de que el cliente logre interactuar con el sistema, reduciendo los costos de corrección, reduciendo el tiempo de perdida con la corrección de errores y aumentando la satisfacción del usuario final.

Al realizarse la automatización de las pruebas unitarias,tanto los equipos de desarrollo, como las empresas pueden lograr reducir el tiempo que se destinan a tareas repetitivas. También, al automatizar pruebas en el código del sistema, obtienen un ahorro notable y tangible en el tiempo, en costos y facilita un sistema de lanzamiento mas rápido con nuevas versiones y actualizaciones.

Al implementar herramientas de QA fortalece el sector tecnológico, esto permite que las em-

presas locales obtengan una buena estabilidad de las compañías frente a las demandas del mercado de software. El estudio de QA automatizado ofrece soluciones a los retos que enfrenta el grupo de desarrollo de investigación del Instituto Tecnológico Sudamericano, ya que se lograría mejorar la calidad del software, optimizando sus procesos de desarrollo y asegurando que el proyecto sea más confiable, obteniendo así una buena experiencia para el usuario con un producto final de alta calidad.

CAPÍTULO I PROBLEMÁTICA

2.1. Problemática

El desarrollo de software en la actualidad enfrenta una problemática debido a la falta de estrategias en las pruebas unitarias del código, lo cual generó impactos negativos en la confiabilidad de los sistemas y el rendimiento de la aplicación. Las pruebas unitarias y automatizadas son necesarias para la verificación individual de cada componente del código, resultando esencial para detectar bugs en el proceso de desarrollo. Los errores en el código provocan inesperados fallos en el sistema del proyecto, elevando significativamente los recursos necesarios para la identificación y resolución de defectos.

Las pruebas unitarias validan el comportamiento de los componentes, así también verificando si cumplen con las condiciones que tiene desarrollado el proyecto. Los problemas dentro del desarrollo de software son imprevisibles cuando los métodos no son validados y se ejecutan en diferentes escenarios. La ausencia de protocolos estandarizados para estas pruebas se descubren únicamente después de afectar a los usuarios finales, comprometiendo el sistema del software.

Las consecuencias de estos problemas se ven relacionado significativamente en las fases de desarrollo y el aumento de los costos operativos de la aplicación. La identificación tardía de estos errores hace que las herramientas resuelvan problemas que podrían haber logrado evitar a una etapa temprana. Este problema no solo afecta a los horarios de trabajo de los desarrolladores, sino que también aumenta los retrasos prolongados a las tareas en realizar y afectar al equipo para lograr sus objetivos.

CAPÍTULO II: MARCO REFERENCIAL

3.1. Marco Teórico

Las organizaciones y empresas tienen un desafío de cambiar radicalmente su administración en busca de calidad en la satisfacción del cliente. Como señala Colina et al. (2018) el énfasis en la calidad ha sido una tendencia creciente en el sector empresarial desde hace décadas. Partiendo de esta base, podemos observar cómo las herramientas de automatización han revolucionado los procesos de control de calidad.

El software es una de las herramientas más útiles para optimizar los procesos organizacionales con el fin de asegurar la optimización, la capacidad y la satisfacción de la demanda, por ende el software debe contar con estándares que garanticen su calidad. Goericke (2019) indica que los niveles de producción de software son los que aceleran las mejoras en la calidad.

Ciertamente, la calidad funcional, así como la no funcional, solo pueden alcanzarse siempre y cuando se cumplan dos aspectos clave: dado que las personas necesitan comprender claramente el significado real del término 'calidad' (entendida como el grado de excelencia en el cumplimiento de requisitos y expectativas); y a su vez, deben conocer concretamente las acciones necesarias para conseguirla. Por lo tanto, la comprensión y el conocimiento práctico son fundamentales para materializar estos objetivos de calidad. Como respuesta a esta necesidad, los investigadores han propuesto estrategias, modelos, métodos, directrices e incluso especificaciones y estándares de calidad que se encargan de que cumpla el software con las expectativas del cliente.

QA es un entorno de alta calidad basado en una gestión adecuada a los modelos que se clasifican como proceso, producto y aplicación. Estos modelos han evolucionado, desde aquellas versiones anteriores que tenía el nombre de pioneros, ahora mantienen similitudes en cuanto a sus características relevantes, estructura y objetivos. Estos modelos demostraron su valor práctico al implementarse en el ámbito empresarial a través de diversos caso de uso.

La relación del software con la seguridad de los sistemas de información es proporcionar a los usuarios una perspectiva positiva para las necesidades de eficiencia y optimización del rendimiento. Estos factores de seguridad se consideran fundamental al implementar un sistema de calidad,

debido a su rápida evolución en el entorno empresarial.

La construcción de soluciones informáticas requiere fundamentos sólidos, donde la *Object Oriented Programming* (OOP) actúa como elemento transformador para afrontar la evolución constante de los sistemas, sin comprometer las fases estructurales del software. Esta metodología facilita la integración de elementos adicionales y el refinamiento de estructuras existentes, preservando la cohesión del conjunto.

El incremento progresivo en las aplicaciones y sus requerimientos convierte la capacidad de crecimiento en factor decisivo. La OOP proporciona una arquitectura adaptable, permitiendo a los equipos técnicos incorporar clasificaciones y entidades adicionales mientras mantienen la estabilidad estructural del sistema.

QA tiene un de control de versiones *Version-control systems* (VCS), como git o *Team Foundation Server* (TFS), son un estándar de factores en el desarrollo de software actual. VCS mantiene un historial adaptable a los cambios de código cometidos por los desarrolladores y ayuda a coordinar y mejorar sus cambios. En el contexto VCS, los cambios se pueden organizar en diferentes ramas, donde una rama es una línea de cambios que se basa en una versión particular del código y se puede fusionar en una versión posterior del código.

En la mayoría de los proyectos de software se utiliza GitHub, sirve para realizar un seguimiento de las solicitudes de cambio que realicen los desarrolladores, como informes de errores, solicitudes de funciones o tareas de mantenimiento. Estos sistemas permiten al equipo de desarrollo gestionar cada solicitud como un ticket, que normalmente tiene un número único, un asignado y un estado de verificación.

En la utilidad de los *Development and Operations* (DevOps) facilita que el ciclo de vida de entrega de software sea más eficiente para permitir que los desarrolladores realicen más cambios comerciales y, en última instancia, tan eficiente que los cambios comerciales puedan lograr realizar de forma continua.

DevOps se implementa mediante una combinación de personas, procesos y herramientas. La integración continua y la entrega continua, también conocidas como CI/CD, son un componente crítico en DevOps. Ayuda a lanzar software de alta calidad y a una velocidad de entrega más rápida.

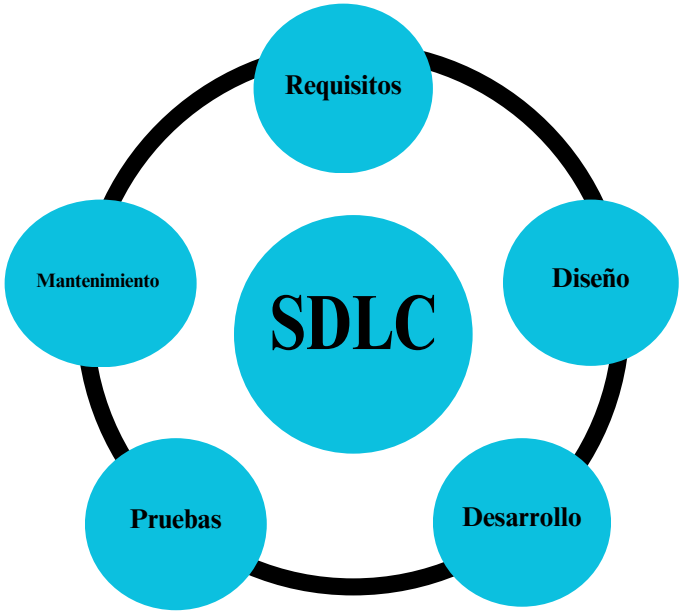
Fomenta una cultura de mentalidad abierta, previsibilidad, formación transversal y realización de tareas compartidas. A menudo, implementar DevOps es más fácil de decir que de hacer. Para resolver el problema de la integración entre diferentes entornos y herramientas, las herramientas DevOps desempeñan un papel fundamental.

El proceso de desarrollo de software se clasifica en diferentes pasos lógicos que permiten a una empresa de desarrollo de software coordinar su trabajo de manera eficiente para crear un software por producto con todas las características necesarias dentro de un plazo y presupuesto determinados.

Los modelos que ayudan a desarrollar el software que buscan los desarrolladores es la visualización detallada y esquemática del SDLC. Incluye todas las actividades necesarias trasladando un producto de software a través de sus fases de SDLC. En la Figura 3.1 se observa la fase múltiple de SDLC

Figura 3.1

Ilustración de fases del SDLCMasso et al. (2020)



- Requisitos: El requisito es importante para reconocer la necesidad del cliente. Para la coherencia de los criterios, se realizarán varias reuniones de revisión. Todos los hallazgos de

la revisión deben registrarse y rastrearse. Se deben realizar entrevistas tanto formales como informales con las partes interesadas adecuadas del solicitante.

- Diseño: Especificaciones se transforman para utilizar los diagramas y la documentación integral de diseños relacionados con el negocio proporcionada.

- Desarrollo: Realiza el grupo desarrollador en la cual son insumos los documentos de estructura junto con la actualización de las revisiones técnicas. Cada código debe colocarse debajo del escáner del equipo, como por ejemplo realizar una inspección a través del código desarrollado y los casos de prueba de la unidad también deben revisarse antes de la ejecución.

- Pruebas: Es la etapa clave de validación del SDLC en la fase de prueba. El enfoque en la prueba completa de las aplicaciones desarrolladas sobre la base de la matriz de requisitos.

- Mantenimiento: Se realiza una reunión de análisis técnico para analizar y finalizar la fase de mantenimiento con el fin de estructurar los resultados y los problemas que se están considerando.

Diversos estudios que realizó Masso et al. (2020) indica que el manejo de SDLC es clave para la entrega de sistemas de software. Este éxito depende de las competencias y la experiencia de la administración de riesgos y los directores de proyectos, así como de elementos como las percepciones, las expectativas de las partes interesadas y la adecuada coordinación de las acciones para mitigar los riesgos. Por otro lado, una gestión inadecuada o la ausencia total de esta se destacan como causas principales del fracaso en los proyectos. Según un reciente informe del *Project Management Institute* (PMI) sobre profesionales en gestión de proyectos, una mala gestión de riesgos y oportunidades sigue siendo uno de los factores más comunes que conducen al fracaso. A esto se suma la falta de preparación tanto de los directores de proyectos como de las organizaciones para afrontar los desafíos que enfrentan. El mismo informe señala que implementar prácticas estándar de gestión de proyectos en toda la organización puede disminuir los riesgos y mejorar los resultados. Estas prácticas permiten minimizar costos, adaptarse a las dinámicas cambiantes del mercado y optimizar la entrega de valor en cada proyecto.

3.2. Marco Conceptual

3.2.1. ¿Qué es el Quality Assurance?

La norma ISO 9000 establece el control en la calidad del sistema, denominado QA como una parte de la gestión de la calidad, cuya responsabilidad es verificar el cumplimiento de los criterios establecidos.

El propósito importante de la validación de calidad es erradicar errores en cada etapa del desarrollo de software y al mismo tiempo garantizar que el producto final siempre mejore. Mientras los protocolos de la calidad, se centra garantizando la creación de alta calidad del software, el control de calidad es una actividad que recopila y examina la calidad de una aplicación una vez que ya ha sido producida. Las pruebas son un componente del sistema de control de calidad, y el control de calidad es un componente del sistema de garantía de calidad.

La relación entre pruebas, aseguramiento de la calidad y control de calidad se muestra en la Figura 3.2 . La creación de directrices y estándares, el control de calidad y la selección de instrumentos apropiados son todas actividades vinculadas al aseguramiento de la calidad.

Figura 3.2

Gráfico de interacción de las pruebasBhanushali (2023)

QA contiene varios requisitos para lograr un buen producto al cliente que son los siguientes:

- Mejora de la Calidad del Producto: QA asegura que el producto final sea de alta calidad, funcional y confiable.
- Satisfacción del Cliente: Un producto libre de errores aumenta la satisfacción del usuario final.
- Estrategia de Pruebas: Definir y seguir una estrategia de pruebas efectiva es esencial para garantizar la calidad.
- Pruebas Automatizadas: Implementar pruebas automatizadas mejora la eficiencia y cobertura del código.

3.2.2. Pruebas de Software

Bhanushali (2023) indicó que las pruebas de software se describen como "un procedimiento estructurado en el que los programas se ejecutan en un sistema informático". Este proceso se enfoca que un conjunto de componentes estén integrados a un sistema de software completo. Cuando los casos de prueba son validados, todas las pruebas correspondientes se llevan a cabo continuando con las metodologías de prueba previamente establecidas y aprobadas.

Son una etapa primordial en el proceso de control de calidad, ya que desempeñan un papel crucial en garantizar una buena funcionalidad y tener un valor elevado del producto final. Al diseñar métodos que son simultáneamente eficientes y efectivos, se ha comprobado que estas estrategias de prueba adecuadas no solo mejora la calidad del software, sino que también optimiza los costos de desarrollo.

El éxito que tiene el mantenimiento de la calidad del software depende de un equipo colaborativo encargado de realizar las pruebas unitarias y automatizadas llamados testers. La estructura y el tamaño de este equipo varían según la complejidad y el alcance del programa, adaptándose a las necesidades específicas de cada proyecto.

Hay dos tipos de pruebas que utilizan los testers generalmente para verificar la calidad en proyectos de software, siendo importante garantizar el correcto funcionamiento de las aplicaciones durante su ciclo de desarrollo:

- Pruebas unitarias: es un método de prueba de software que verifica el correcto funcionamiento de los componentes de un programa de manera aislada. En el desarrollo de software, una ünidad"se refiere típicamente a una función individual, método o clase.

- Pruebas automatizadas: son procesos donde se utiliza software especializado para controlar la ejecución de pruebas, comparar los resultados esperados con los resultados reales, configurar las precondiciones de las pruebas y otros controles y funciones de prueba. También, permite ejecutar casos de prueba predefinidos de forma repetitiva y efectiva, reduciendo significativamente el tiempo y esfuerzo en comparación con las pruebas manuales.

3.2.3. Re-factorización en la Calidad de Software

Aniche et al. (2022) indica que re-factorizar un código es el proceso de modificar un sistema, específicamente el software de tal manera que no afecte el comportamiento del código y el flujo del sistema, pero que mejore su estructura interna. Sin embargo, decidir cuándo y qué re-factorizar ha planteado durante mucho tiempo este desafío para los desarrolladores. Los equipos de desarrollo de software no deberían simplemente re-factorizar sus sistemas de software a voluntad propia, o decidir no re-factorizar un fragmento de código que genere deuda técnica, ya que cualquier actividad de re-factorización conlleva costos altos.

El código fuente mediante refactorización hace que el software sea fácil de entender sin cambiar su comportamiento observable. Según Kaur y Singh (2017), los diferentes métodos de refactorización aplicados en el lugar adecuado pueden ser beneficiosos para la mejora incremental de la calidad del software. Los problemas existentes del software se pueden eliminar mejorando el código con la ayuda de la refactorización, que puede modificar las actividades internar con el propósito de aceptar sus procesos.

Para ello, los desarrolladores de software han confiado cada vez más en diferentes herramientas de análisis estático como forma de recopilar comentarios sobre su código fuente. No sólo utilizan estas herramientas para encontrar problemas relacionados con errores en sus sistemas, sino también para obtener asesoramiento relacionado con la calidad del código.

Existe varias herramientas populares como PMD, ESLint y Sonarqube que ahora se han integrado en diferentes etapas del flujo de trabajo de los desarrolladores, por ejemplo, se realiza un test del código para finalmente tener un informe de calidad general del código que se realizo con Sonarqube.

3.2.4. Pruebas Unitarias

Una prueba unitaria representa un enfoque técnico que se centra en probar componentes individuales de un producto de software.Los profesionales de QA desarrollan pruebas unitarias mientras construyen la aplicación.

El propósito fundamental es garantizar que cada unidad del software funcione como se espera y cumpla con los requisitos, también ayuda a garantizar que el software funcione correctamente antes de su lanzamiento.

Bakharev (2023) indicó que las unidades de una prueba unitaria puede ser funciones, procedimientos, métodos, objetos u otras entidades del código de la aplicación. El equipo de desarrollo es el cual decide que unidad es la más adecuada para entender y lograr el sistema de la aplicación.

Ofrecen beneficios en el ciclo de desarrollo de software, permitiendo la detección temprana de bugs o problemas de código, a su vez tiene lanzamientos mas frecuentes de versiones y asegura que el código pueda ser refactorizado con facilidad. Además tiene una documentación valiosa sobre el comportamiento del sistema, fortaleciendo así la calidad y la confiabilidad de la aplicación.

Las pruebas unitarias generalmente constan de cuatro fases:

1. Planificación y configuración del entorno: Los desarrolladores consideran que las unidades del código necesitan probar y cómo ejecutar toda la funcionalidad de la aplicación.

2. Escritura de casos de prueba y scripts: Los desarrolladores escriben el código de prueba unitaria y preparan los scripts para ejecutar el código.

3. Ejecución de la prueba unitaria: La prueba se ejecuta y revela cómo se comporta el código para cada caso de prueba.

4. Análisis de los resultados: Los desarrolladores pueden identificar errores o problemas en el código y solucionarlos.

Runeson (2006) argumentó que el desarrollo basado en pruebas *Test Driven Development* (TDD) es un enfoque común para las pruebas unitarias. Requiere que el desarrollador cree primero la prueba unitaria, antes de que exista realmente el código de la aplicación, está prueba inicialmente fallará, luego el desarrollador agrega la funcionalidad relevante a la aplicación hasta que las pruebas pasen.

Cuando se han realizado pruebas unitarias de un proyecto de software, los desarrolladores saben que cada unidad individual es eficiente, refactorizado, no tiene errores y estará listo para poner en producción a la aplicación.

3.2.5. TypeScript

TypeScript, desarrollado y mantenido por Microsoft según Gowda (2019), se caracteriza por ser un conjunto muy completo sintáctico estricto de JavaScript que añade escritura estática si es que se requiere al lenguaje, siendo creado específicamente para el desarrollo de aplicaciones complejas y transformándose por defecto a JavaScript. Como señala Nguyen (2022), es el lenguaje más exitoso entre los lenguajes escritos gradualmente y el más popular que se compila en JavaScript, destacándose por tener un sistema de tipos estructural, estático, fuerte, en su mayoría inferido y tipificado gradualmente, siendo notable que se originó principalmente desde Microsoft con contribuciones de Google, todos los programas JavaScript existentes son a su vez códigos TypeScript legítimos.

TypeScript como lenguaje es importante para el desarrollo web al hacer que el código sea más legible y limpio, mejorando el manejo de errores y facilitando su mantenimiento. También, revela errores que se pueden evitar fácilmente, lo que ayuda a reducir el esfuerzo y el tiempo de depuración redundante.

3.2.6. SonarQube

SonarQube, la herramienta de análisis de código estático que garantiza una mejora en la calidad del software. Su principal objetivo es analizar el código de manera exhaustiva para identificar problemas, bugs o fallos y mejorar la calidad del software desde etapas tempranas del desarrollo. Las principales funcionalidades que ofrece SonarQube es el cálculo de métricas clave, como el número de líneas que tiene el código, la complejidad y la verificación del cumplimiento con las condiciones que tiene el código.

Baldassarre et al. (2020) indicó que los elementos del SonarQube se clasifican según tres características de calidad: confiabilidad, mantenibilidad y seguridad. Las reglas de confiabilidad, son conocidas como "bugs", esto representan errores que se pueden reflejarse eventualmente en fallos del sistema. Los problemas relacionados con la mantenibilidad, denominados *Code Smells*, hacen referencia a aspectos del código que dificultan su comprensión y modificación, lo que podría derivar en mayores costos de mantenimiento a largo plazo.

El concepto de "Code Smells" en SonarQube no corresponde exactamente al definido por Baldassarre et al. (2020) quien indica como los indicios superficiales de errores o problemas más profundos del sistema. También abordo uno perspectiva más amplia, SonarQube se enfoca únicamente en problemas asociados con la mantenibilidad. Para facilitar la priorización de los problemas encontrados, SonarQube clasifica las reglas en cinco niveles de severidad: Bloqueante, Crítico, Mayor, Menor e Información. Esta clasificación permite a los equipos de desarrollo priorizar los problemas más críticos, como aquellos que afectan directamente la funcionalidad o la seguridad del sistema. El análisis de SonarQube tiene como propósito determinar su relación con los fallos en el software, ya que se puede presentar errores críticos que pueden comprometer la estabilidad del sistema.

3.2.7. Mocha

Mocha es un framework de pruebas diseñado para entornos de gran escala, que se ejecuta en Node.js. Rodríguez-Martínez et al. (2001)indicó que su principal característica es radicar una gran

capacidad de pruebas ejecutadas para garantizar informes precisos, mientras asigna las excepciones no capturadas a los casos de prueba correspondientes. La eficiencia del sistema se fundamenta por su capacidad para optimizar el procedimientos de pruebas automatizadas, añadiendo una estrategia de ejecución donde los testers generan resultados más extensos de la aplicación. KC (2019) explicó, que el framework Mocha proporciona funcionalidades como la descripción de pruebas automatizadas, genera reportes en un *Hypertext Markup Language* (HTML) sobre las validaciones de la aplicación y la escritura de diferentes módulos de prueba. Mocha ha demostrado mejoras significativas en el rendimiento de las pruebas automatizadas, consolidándose como uno de los frameworks de testing más populares en el desarrollo de software actual.

3.2.8. Cypress

Cypress es una herramienta de pruebas que ofrece a los desarrolladores web sobre el aseguramiento de calidad, ya que es una solución integral para realizar pruebas de extremo a extremo en aplicaciones web. Esta herramienta se destaca por su interfaz intuitiva y sus capacidades de prueba en tiempo real, lo que agiliza el proceso de garantizar una buena funcionalidad en su arquitectura distinta que combina perfectamente las pruebas para las necesidades del desarrollo web moderno.

En el fronted, las respuestas a los eventos de la aplicación ocurren en tiempo real, cumpliendo con las condiciones que se puede operar a través de la red y modificando el código que impide la automatización del navegador.

Explicó Monje Morales (2023) que las diferentes pruebas pueden abarcar varias validaciones, desde simples interacciones con la interfaz del usuario hasta pruebas complejas basadas en cumplir condiciones del código. Cypress permite a los testers crear entornos de pruebas completas y realistas, evaluando la funcionalidad, el rendimiento y la confiabilidad de la aplicación.

De acuerdo con Tasnim Taky (2021), Cypress tiene la opción de modificar cualquier componente de la funcionalidad de la aplicación, generando estados artificialmente, quiere decir que se hace pruebas unitarias y automatizadas, en lugar de realizar pruebas lentas y costosas.

Ventajas:

1. Realiza capturas de pantalla de fallos o bugs durante la ejecución de las pruebas, permitiendo

a los testers obtener información sobre el código de la aplicación web.

2. Maneja automáticamente las solicitudes y condiciones, agilizando el flujo de trabajo de pruebas.

3. La compatibilidad de navegadores se ha expandido más allá de Chorme, ofreciendo ahora soporte para los navegadores Firefox y Edge.

4. Permite crear condiciones específicas de prueba y simular diferentes escenarios sin necesidad de configurar manualmente el estado de la aplicación.

3.2.9. GitLab

GitLab está diseñada para gestionar la totalidad de un proyecto de software, incluyendo la gestión del código fuente, la gestión de aplicaciones, el control de calidad y la coordinación de versiones. Según Rios et al. (2019), GitLab tiene numerosas herramientas para crear, compilar, monitorear y gestionar un proyecto, permitiendo que los equipos de operaciones no estén obligados a utilizar exclusivamente las herramientas integradas.

GitLab contiene herramientas útiles para los ingenieros de software que requieren automatizar muchas partes del SDLC, desde la planificación inicial hasta la implementación final en el monitoreo del nuevo código en uso, ahorrando tiempo y esfuerzo en comparación con los procesos manuales. De acuerdo Choudhury et al. (2020), si todas las pruebas se completan exitosamente, el código puede liberarse con poca o ninguna intervención manual. GitLab es reconocida por su producto de *Continuous Integration* (CI), que permite a los equipos de programación dividir un proyecto complejo en partes, así lograr trabajar en paralelo con tareas especializadas y luego ensamblarlas todo que sea funcional.

CAPÍTULO III: METODOLOGÍA DE INVESTIGACIÓN

4.1. Enfoque investigativo

A medida que el software se integra en todos los aspectos de la vida diaria, su impacto ha alcanzado áreas tan diversas como los sistemas financieros, el transporte, la atención médica y los procesos industriales. Los frecuentes errores del sistema de software, combinados con la incapacidad de los desarrolladores de cumplir los plazos y los requisitos técnicos, plantean preguntas críticas sobre cómo abordar estos problemas. Los métodos de garantía del software, están enfocados en transferir al campo del software estándares y prácticas específicas. Este proceso es importante para definir con mayor precisión qué y por qué se realiza el aseguramiento de la calidad, facilitando un uso como estrategia clave asegurando la confiabilidad y efectividad de los proyectos de software. (Buckley & Poston, 1984)

4.2. Enfoque Cuantitativo

En el ámbito del software requiere establecer parámetros de medición específicos, asegurando la reproducibilidad de resultados entre distintos evaluadores. Esta metodología contempla la determinación precisa de componentes, características y unidades evaluativas, con fases particulares en el proceso de desarrollo. Los marcos de referencia como *Goal Question Metric* (GQM) tiene relaciones entre entidades que proporcionan herramientas fundamentales para la recolección y el análisis sistemático de datos. La aplicación de estas metodologías no solo consolida la fiabilidad en las mediciones, sino que también contribuye significativamente a la optimización de procesos y el fortalecimiento de la calidad del software (Kitchenham et al., 2001).

4.3. Enfoque Cualitativo

Adoptó una perspectiva exploratoria mediante el análisis de casos múltiples, examinando las estrategias implementadas por responsables de calidad en organizaciones de desarrollo de software de escala reducida, con énfasis en la optimización de procesos de QA (Underwood, 2016). La investigación priorizó las metodologías y prácticas organizacionales de QA, analizando enfoques

adoptados por los gestores, en contraposición a los resultados. Esta aproximación metodológica permitió profundizar en las complejidades que enfrentan las empresas emergentes en la gestión de calidad, identificando prácticas efectivas para optimizar la eficiencia operativa y la viabilidad económica en la implementación de protocolos de QA.

4.4. Métodos de investigación

En el ámbito de QA, las encuestas son importantes para obtener datos precisos sobre los procesos y prácticas utilizados por el equipo de investigación. Se realizó un curso de testing para facilitar el manejo de metodología en el ámbito de calidad y re-factorización del código. Este enfoque de métodos mixtos permitió la recopilación de datos cuantitativos sobre la eficacia de las prácticas de garantía de calidad y datos cualitativos sobre las experiencias y perspectivas del equipo de desarrollo.

4.5. Recolección de datos

Se realizó encuestas detalladas para recolectar datos de manera cuantitativa. Estas encuestas se dirigieron al equipo de desarrollo que trabajan con el código de la aplicación. El propósito es obtener información sobre el desarrollo del sistema de software, así poder identificar el conocimiento que tiene manejando el control de calidad del software y si alguna vez han realizado pruebas unitarias en algún proyecto trabajado por ellos.

La encuesta se realizó con los estudiantes del Instituto Tecnológico Sudamericano.

Preguntas de las encuestas realizadas

1. ¿Consideras que el QA es necesario para mejorar la calidad de la aplicación?
2. ¿Qué áreas del código de la aplicación ANI crees que podrían beneficiarse más del QA ?
3. ¿Con qué frecuencia se enfrentan errores y bugs significativos en el desarrollo actual?

4. ¿Existen errores recurrentes que crees que el QA puede ayudar a cambiar o evitar ? ¿Cuáles son?

5. ¿Cómo crees que el QA afectará la productividad del equipo?

6. ¿Qué temores o preocupaciones tienes respecto al impacto del QA en el flujo de trabajo?

7. ¿Tienes experiencia previa con herramientas o procesos de QA? En caso afirmativo, ¿Qué herramientas o métodos has utilizado y cuáles recomendarías?

8. ¿Cuáles son tus expectativas más importantes en relación con el impacto del QA en la aplicación y en el equipo?

Preguntas de de la encuesta con la implementación del curso QA en el equipo de desarrollo de investigación

1. ¿Considera que la implementación de QA ha mejorado la calidad de la aplicación?

2. ¿Cómo calificarías la calidad del código y la estabilidad actual de la aplicación?

3. ¿Qué tan útil considera los reportes generados por el proceso de QA?

4. ¿Cuál ha sido el mayor reto al integrar QA en el desarrollo?

5. ¿Cómo ha cambiado la productividad del equipo con la implementación de QA?

6. ¿Qué tan útil fue el curso de QA para aplicarlo en el día a día?

4.6. Instrumentos y técnicas para el levantamiento de la información

Para la implementación basado en pruebas de calidad basado en QA, se utilizaron diversos instrumentos y técnicas de recolección de información que permitieron instaurar un marco de trabajo robusto y efectivo.

- **Encuestas y Entrevistas**: Se realizaron encuestas medibles al equipo de desarrollo para evaluar:

    - Experiencia previa con herramientas de pruebas de calidad del código
    - Conocimiento de prácticas de QA implementadas
    - Necesidades específicas del proyecto actual
    - Desafíos en el proceso de desarrollo actual
- **Consulta con Expertos**: Se llevaron a cabo sesiones de consulta con:
    - Docentes a cargo del desarrollo
    - Líderes del equipo de programación
- **Análisis de Documentación Técnica**: Se realizó una completa revision de:
    - Documentación oficial de herramientas (GitLab, SonarQube, ESLint, Cypress)
    - Documentación técnica de la aplicación existente
- **Capacitación y Formación**: Se participó en:
    - Cursos especializados en QA

Las técnicas y métodos contribuyeron establecer una base completa para la implementación del sistema de QA, asegurando que se tenga presente tanto las necesidades técnicas como las mejores prácticas.

4.7. Metodología de trabajo

Se planificó en base a la metodología Kanban, un sistema de gestión más visual que enfatiza la entrega continua sin sobrecargar al equipo de desarrollo. Como señalan Anderson (2010), Kanban facilita la optimización del flujo de valor a través del sistema, permitiendo una mejor previsibilidad y calidad en la entrega de resultados. Este enfoque resulta particularmente efectivo en proyectos donde la visualización del progreso y la identificación temprana de cuellos de botella son cruciales para mantener un flujo de trabajo eficiente y adaptable a los cambios.

4.7.1. Aplicación de la metodología para el desarrollo de la aplicación

Para ejecutar el proyecto de manera organizada fue indispensable aplicar procesos ágiles, la metodología apropiada fue Kanban fundamentada en la visualización del flujo de trabajo y la optimización continua de los procesos. Es destacada por su facilidad para mantener un control preciso sobre las diferentes fases del proyecto, que abarcan desde la investigación inicial hasta la evaluación final, organizadas en un periodo de 21 semanas.

4.7.2. Implementación del tablero Kanban

La planificación del proyecto se organizó mediante un tablero Kanban con columnas como:

- Por hacer: Actividades planificadas sin comenzar
- En progreso: Tareas siendo realizadas, dividido por responsables
- Por aprobar: Pendientes para revisar en tutorías
- Finalizados: Tareas completas y revisadas

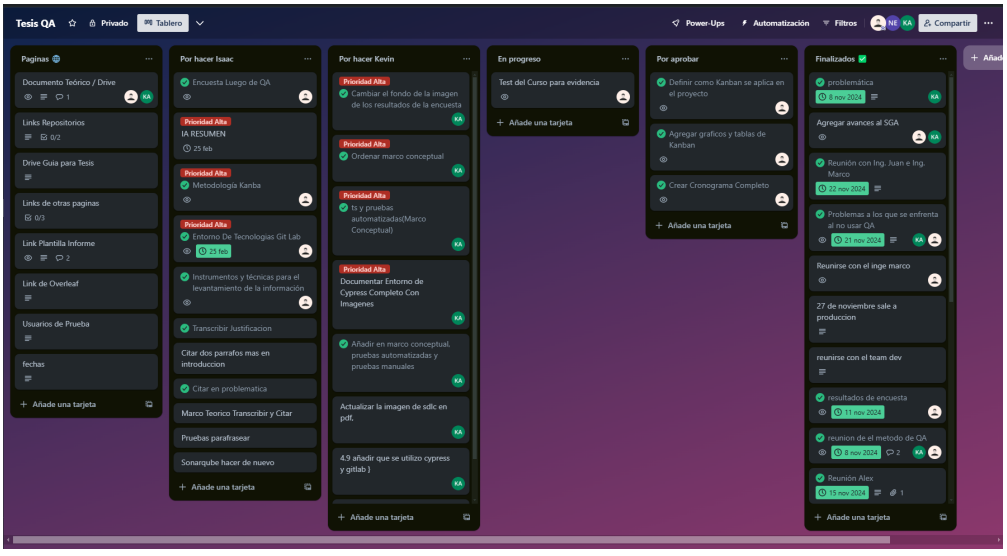


Figura 4.1

Gráfico de Tablero Trello Kanban

4.7.3. Tiempos Limitantes de trabajo en proceso WIP

Se propusieron tiempos específicos para cada fase:

- Fase de Investigación y Planificación: como máximo 2 actividades simultáneas
- Fase de Desarrollo de Guías: máximo 2 actividades simultáneas
- Fase de Implementación Técnica: máximo 1 actividad por responsable
- Fase de Capacitación y Ajustes: máximo 1 actividades
- Fase de Evaluación y Cierre: máximo 2 actividades simultáneas

4.7.4. Mediciones y seguimiento

Para garantizar que el proceso sea eficaz, se ocuparon métricas como:

- Tiempo para tarea: medición del tiempo desde que empieza hasta que termina
- Lead time: tiempo total desde la planificación hasta llegar a completar
- Eficacia del flujo: porcentaje de tiempo en progreso comparado con tiempo total del proyecto

4.7.5. Revisión y reuniones

Se establecieron los siguientes eventos:

- Reuniones por dia para analizar tareas (15 minutos)
- Restructuracion semanal del tablero Kanban en Trello

4.7.6. Administrar de entregables

Los entregables principales se organizaron según las fases definidas en el cronograma:

- Plan de implementación QA (Fase 1)
- Guías y templates de informes QA (Fase 2)

- Ambiente de pruebas configurado para Fronted y Backend (Fase 3)
- Documentación y equipo de desarrollo capacitado (Fase 4)
- Informes de la implementación (Fase 5)

4.7.7. Roles y responsabilidades

Se definieron dos roles principales con responsabilidades específicas:

- Isaac Guerra: Responsable de análisis, desarrollo de guías y evaluación
- Kevin Quito: Responsable de revisión, implementación y capacitación

4.7.8. Políticas de calidad

Se usaron varias reglas para dar como terminado un trabajo

- Definición de Completado en una tarea
- Porcentaje de Finalización requerido
- Revisión coordinada y cruzada

4.7.9. Fases de desarrollo con la metodología Kanban

La Tabla 4.1 muestra la primera fase del proyecto. Las fases consiste en tareas especificas que se llevan a cabo para analizar la funcionalidad del código del proyecto, además, incluye un curso de análisis de QA para mejorar con el aprendizaje de análisis de código con QA.

Tabla 4.1: Fase de Investigación y Planificación

ID	**Actividad**	**Inicio**	**Fin**	**Estado**	**Responsable**
F1-01	Análisis inicial del estado actual	01/10/24	07/10/24	Por iniciar	Isaac Guerra
F1-02	Revisión de literatura y mejores prácticas	08/10/24	14/10/24	Por iniciar	Kevin Quito
F1-03	Selección de herramientas QA	15/10/24	21/10/24	Por iniciar	Isaac Guerra
F1-04	Definición del plan de implementación	22/10/24	28/10/24	Por iniciar	Kevin Quito

La Tabla 4.2 muestra la segunda fase del proyecto, en el cual se realizaron casos de pruebas unitarias, con pruebas de integración y con E2E que analiza todo el proyecto de inicio a fin.

Tabla 4.2: Fase de Desarrollo de Guías y Material

ID	**Actividad**	**Inicio**	**Fin**	**Estado**	**Responsable**
F2-01	Desarrollo inicial de la guía QA	29/10/24	04/11/24	Por iniciar	Isaac Guerra
F2-02	Creación de templates y estándares	05/11/24	11/11/24	Por iniciar	Kevin Quito
F2-03	Preparación material de capacitación	12/11/24	18/11/24	Por iniciar	Isaac Guerra
F2-04	Revisión y ajuste de documentación	19/11/24	25/11/24	Por iniciar	Kevin Quito

La Tabla 4.3 muestra la tercera fase del proyecto, en la que se documenta el análisis de la aplicación con el curso aprendido de QA y se evalúa el rendimiento de la aplicación.

Tabla 4.3: Fase de Implementación Técnica

ID	**Actividad**	**Inicio**	**Fin**	**Estado**	**Responsable**
F3-01	Configuración del ambiente de pruebas	26/11/24	02/12/24	Por iniciar	Isaac Guerra
F3-02	Implementación de pruebas unitarias	03/12/24	09/12/24	Por iniciar	Kevin Quito
F3-03	Desarrollo de pruebas de integración	10/12/24	16/12/24	Por iniciar	Isaac Guerra
F3-04	Automatización inicial de pruebas	17/12/24	23/12/24	Por iniciar	Kevin Quito

La Tabla 4.4 muestra la cuarta fase del proyecto, en la que logre dominar las herramientas avanzadas de QA y también dominar las pruebas unitarias con SonarQube.

Tabla 4.4: Fase de Capacitación y Ajustes

ID	Actividad	Inicio	Fin	Estado	Responsable
F4-01	Preparación final del material	24/12/24	30/12/24	Por iniciar	Isaac Guerra
F4-02	Sesión de capacitación 1	31/12/24	06/01/25	Por iniciar	Kevin Quito
F4-03	Sesión de capacitación 2	07/01/25	13/01/25	Por iniciar	Isaac Guerra
F4-04	Retroalimentación y ajustes	14/01/25	20/01/25	Por iniciar	Kevin Quito
F4-05	Documentación de resultados iniciales	21/01/25	27/01/25	Por iniciar	Isaac Guerra

La Tabla 4.5 muestra la quinta fase del proyecto, en el cual se ejecuta las unidades criticas de la aplicación y se documenta las métricas de calidad obtenidas.

Tabla 4.5: Fase de Evaluación y Cierre

ID	Actividad	Inicio	Fin	Estado	Responsable
F5-01	Evaluación de implementación	28/01/25	03/02/25	Por iniciar	Isaac Guerra
F5-02	Análisis de resultados	04/02/25	10/02/25	Por iniciar	Kevin Quito
F5-03	Preparación de informe final	11/02/25	17/02/25	Por iniciar	Isaac Guerra
F5-04	Cierre y presentación de resultados	18/02/25	24/02/25	Por iniciar	Kevin Quito

La Tabla 4.6 muestra la sexta fase del proyecto, se acabaría redactando el documento de tesis en formato académico y también entregar los informes realizados de QA.

Tabla 4.6: Resumen de Fases

Fase	Duración	Inicio	Fin	Entregables Principales
Investigación y Planificación	4 semanas	01/10/24	28/10/24	Plan de implementación QA
Desarrollo de Guías	4 semanas	29/10/24	25/11/24	Guías y templates QA
Implementación Técnica	4 semanas	26/11/24	23/12/24	Ambiente de pruebas configurado
Capacitación y Ajustes	5 semanas	24/12/24	27/01/25	Personal capacitado y documentación
Evaluación y Cierre	4 semanas	28/01/25	24/02/25	Informe final de implementación

CAPÍTULO IV: INTERPRETACIÓN ANALÍTICA DE ENCUESTAS RESULTANTES

5.1. Creación de encuestas

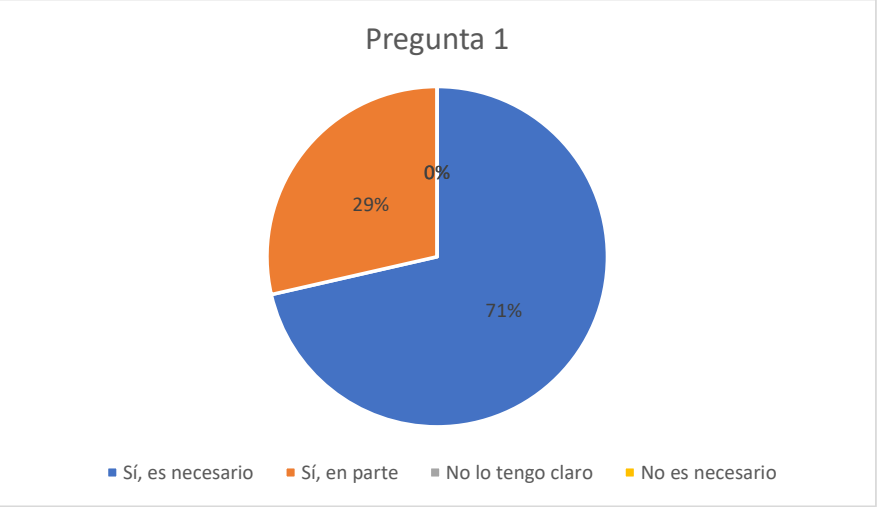
La encuesta como prioridad es para extraer información para un tema específico. Para lo cual, se requiere comprender el flujo de trabajo de la tecnología SonarQube, para el control de calidad de los proyectos realizados por los desarrolladores. La encuesta fue respondida por los desarrolladores de la aplicación, lo que permitió obtener datos importantes para hacer un análisis e identificar los requisitos necesarios para cumplir con los problemas de los encuestados. Los datos obtenidos de las encuestas se muestran en forma de gráficos y porcentajes, lo que permite una interpretación más clara y apoya a la toma de decisiones.

Respuestas de las encuestas realizadas a los desarrolladores del proyecto de investigación:

En la Figura 5.1 se observa que el 71 % de los desarrolladores consideran que sí es necesario QA para mejorar la calidad de la aplicación, mientras que el 29 % considera de manera no importante utilizar QA en la aplicación.

Figura 5.1

Gráfico de pregunta 1.



La Tabla 5.1 presenta la valoración de la pregunta 1.

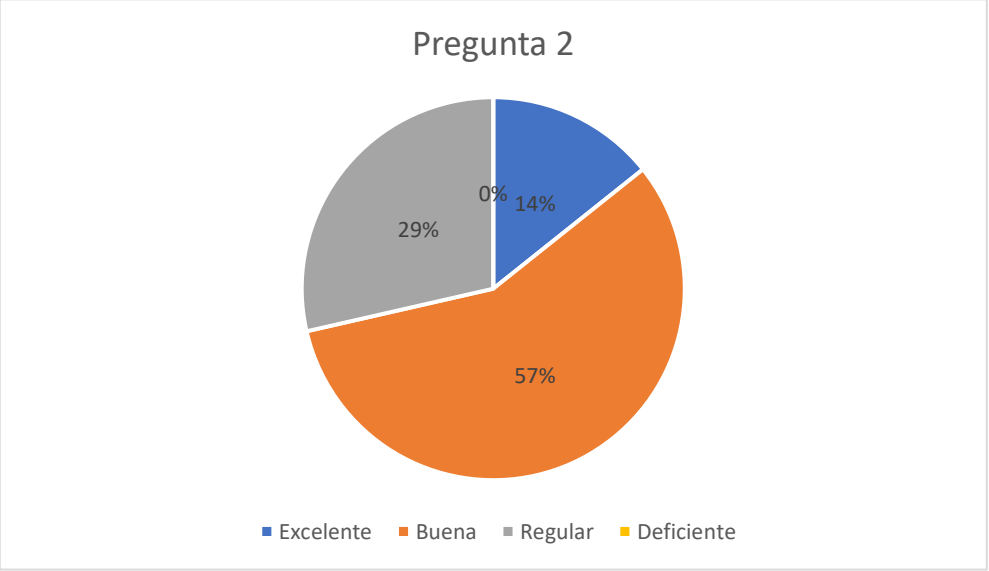
Tabla 5.1: Tabla de respuesta, Pregunta 1

1. ¿Consideras que el QA es necesario para mejorar la calidad de la aplicación?				
Xi	fi	Fi	ni - %	NI
Sí, es necesario	5	5	79 %	0,79
Sí, en parte	2	7	21 %	0,21
No lo tengo claro	0	7	0 %	0
No es necesario	0	7	0 %	0

En la Figura 5.2 se observa que el 57 % de los desarrolladores da una buena calificación a la calidad y estabilidad de la aplicación, mientras que el 29 % y el 14 % dan una calificación regular y excelente a la calidad y estabilidad de la aplicación.

Figura 5.2

Gráfico de pregunta 2.



La Tabla 5.2 presenta la valoración de la pregunta 2.

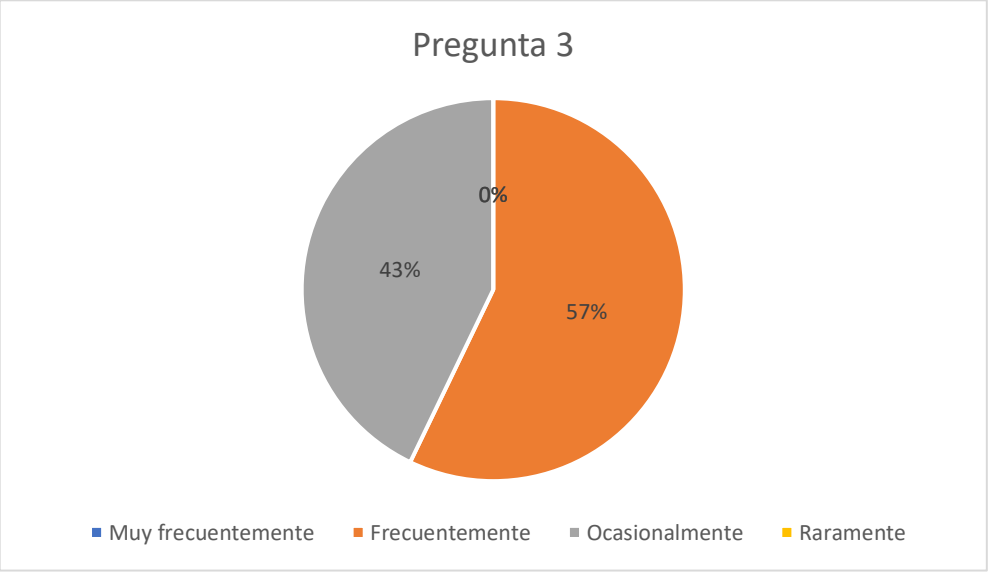
Tabla 5.2: Tabla de respuesta, Pregunta 2

2.¿Cómo calificarías la calidad del código y la estabilidad actual de la aplicación?				
Xi	fi	Fi	ni - %	NI
Buena	4	4	57 %	0,42
Regular	2	6	29 %	0,58
Excelente	1	7	14 %	0,14
Deficiente	0	7	0 %	0

En la Figura 5.3 se observa que el 57 % de los desarrolladores se enfrentan frecuentemente con errores o bugs en el desarrollo de la aplicación, mientras que el 43 % se enfrenta ocasionalmente con errores o bugs en el desarrollo de la aplicación.

Figura 5.3

Gráfico de pregunta 3.



La Tabla 5.3 presenta la valoración de la pregunta 3.

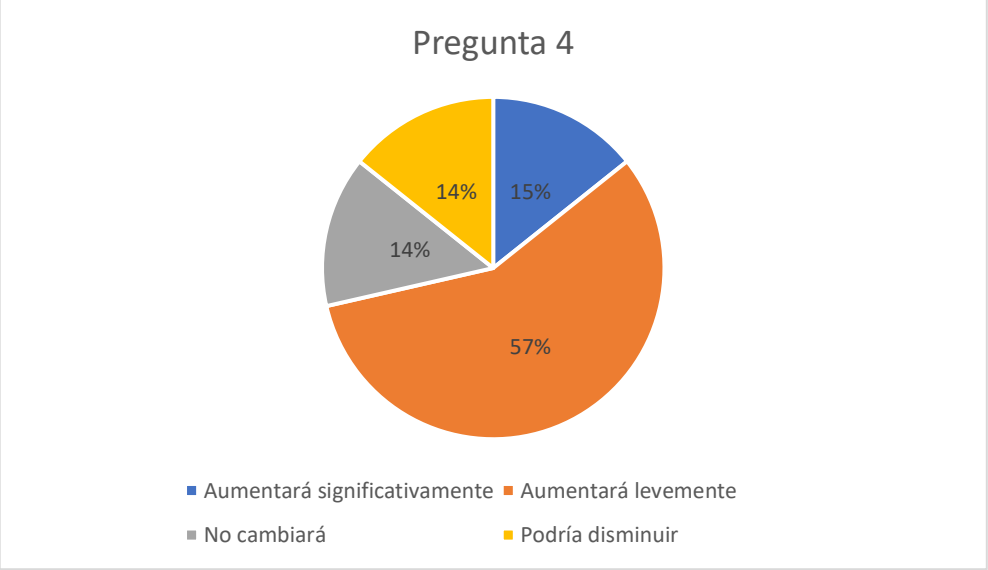
Tabla 5.3: Tabla de respuesta, Pregunta 3

3. ¿Con qué frecuencia se enfrentan errores y bugs significativos en el desarrollo actual?				
Xi	fi	Fi	ni - %	NI
Muy frecuentemente	0	0	0 %	0
Frecuentemente	4	4	57 %	0,57
Ocasionalmente	3	7	43 %	0,43
Raramente	0	7	0 %	0

En la Figura 5.4 se observa que los desarrolladores tienen un porcentaje variado, al creer si el QA afecta a la productividad del equipo, tiene un 15 % que aumentará significativamente la productividad del equipo, el 57 % aumentará levemente la productividad del equipo, el 14 % indica que no cambiará y podriá disminuir la productividad del equipo.

Figura 5.4

Gráfico de pregunta 4.



La Tabla 5.4 presenta la valoración de la pregunta 4.

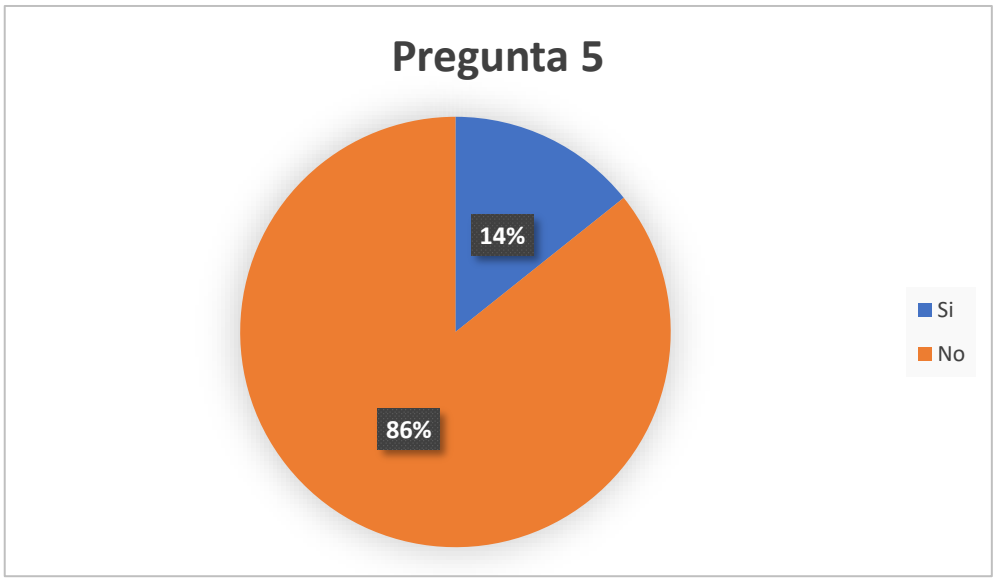
Tabla 5.4: Tabla de respuesta, Pregunta 4

4.¿Cómo crees que el QA afectará la productividad del equipo?				
Xi	fi	Fi	ni - %	NI
Aumentará significativamente	1	1	15 %	0,15
Aumentará levemente	4	5	57 %	0,57
No cambiará	1	6	14 %	0,14
Podría disminuir	1	7	14 %	0,14

En la Figura 5.5 se observa que el porcentaje del 14 % es que si tienen un experiencia previa con herramientas QA, mientras que el 86 % de los desarrolladores no tienen experiencia con herramientas QA.

Figura 5.5

Gráfico de pregunta 5.



La Tabla 5.5 presenta la valoración de la pregunta 5.

Tabla 5.5: Tabla de respuesta, Pregunta 5

5. ¿Tienes experiencia previa con herramientas o procesos de QA?				
Xi	fi	Fi	ni - %	NI
Si	1	1	14 %	0,14
No	6	7	86 %	0,86

- En la pregunta 6, ¿Qué áreas del código de la aplicación ANI crees que podrían beneficiarse más del QA? Los desarrolladores dieron una respuesta a su punto de vista que destaca la autenticación, integración de APIs, validación de datos, UI y rendimiento como áreas clave. Además, recomendaron mejorar el refactor en general y optimizar la interfaz, ya que ha presentado más problemas.

- En la pregunta 7: ¿Existen errores recurrentes que crees que el QA puede ayudar a cambiar o evitar?, los desarrolladores identificaron problemas clave como seguridad, rendimiento, errores de regresión, validación de datos y problemas de UI como áreas donde QA puede intervenir preventivamente.

- En la pregunta 8: ¿Qué temores o preocupaciones tienes respecto al impacto del QA en el flujo de trabajo?, los desarrolladores expresaron principalmente inquietudes sobre posibles retrasos en las entregas y el aumento en las estimaciones de tiempo, aunque algunos no tienen preocupaciones.

- En la pregunta 9: En caso afirmativo, ¿Qué herramientas o métodos has utilizado y cuáles recomendarías?, los desarrolladores mencionaron herramientas específicas como Selenium, Cypress, Jenkins, JUnit, BrowserStack y SonarQube, aunque varios indicaron no tener experiencia con herramientas de QA.

- En la pregunta 10: ¿Cuáles son tus expectativas más importantes en relación con el impacto del QA en la aplicación y en el equipo?, los desarrolladores expresaron expectativas centradas en la obtención de mejor calidad del código, detección temprana de bugs, mejor colaboración en equipo y optimización de la experiencia del usuario, tanto en aplicaciones web como móviles.

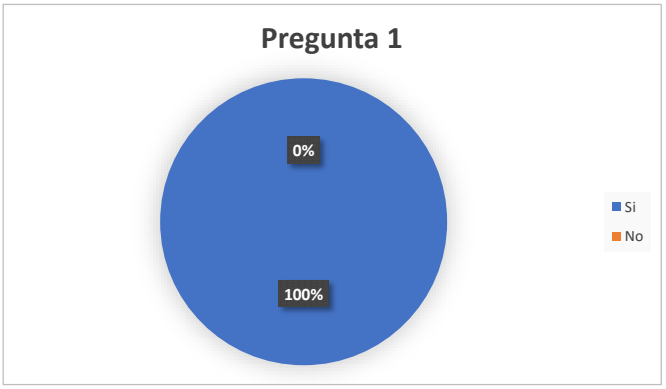
Resultados de la encuesta con la implementación del curso QA en el equipo de desarrollo de investigación

1. ¿Considera que la implementación de QA ha mejorado la calidad de la aplicación?

   En el gráfico se muestra que el 100 % de los desarrolladores consideran que la implementación de QA ha mejorado en la aplicación.

Figura 5.6

Gráfico de la pregunta número 1

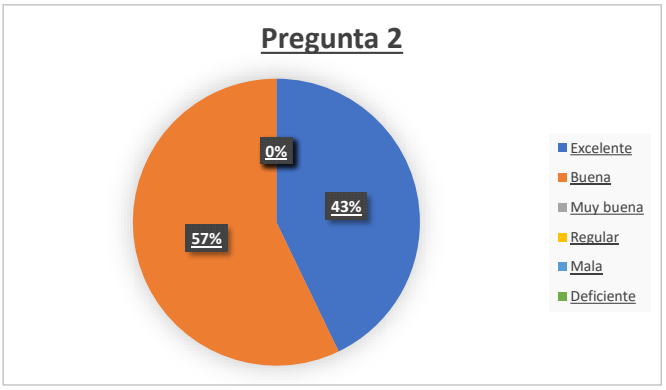


2. ¿Cómo calificarías la calidad del código y la estabilidad actual de la aplicación?

   En el diagrama se puede observar que el 43 % de los desarrolladores dan una excelente calificación a la calidad y estabilidad actual de la aplicación, mientras el 57 % de los desarrolladores dan una buena calidad y estabilidad actual en la aplicación.

Figura 5.7

Gráfico de la pregunta número 2

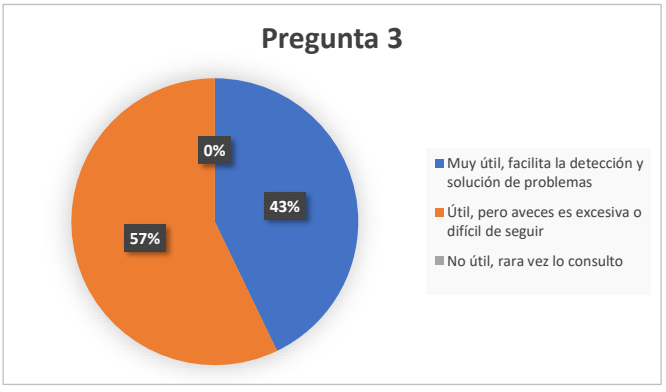


3. ¿Qué tan útil considera los reportes generados por el proceso de QA?

   En el diagrama se puede observar que el 43 % de los desarrolladores consideran muy útil los reportes generados por QA, mientras que el 57 % de los desarrolladores indican que es útil pero presentan ocasionalmente dificultades en el seguimiento del reporte presentado.

Figura 5.8

Gráfico de la pregunta número 3

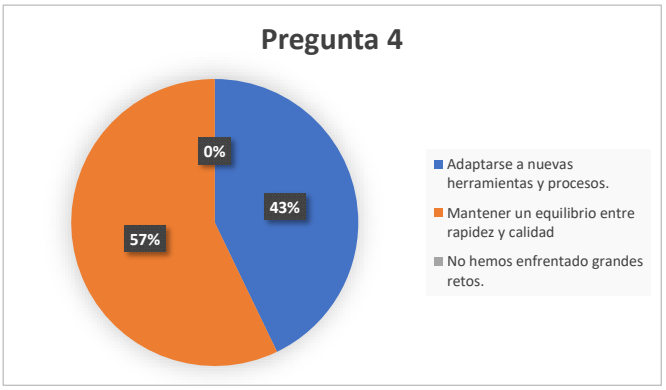


4. ¿Cuál ha sido el mayor reto al integrar QA en el desarrollo?

   En el diagrama se puede observar que el 43 % de los desarrolladores el mayor reto de ellos a sido adaptarse a nuevas herramientas y procesos QA, mientras que el 57 % de los desarrolladores consideran en mantener un equilibrio entre rapidez y calidad para seguir integrando QA en su desarrollo de aplicación.

Figura 5.9

Gráfico de la pregunta número 4

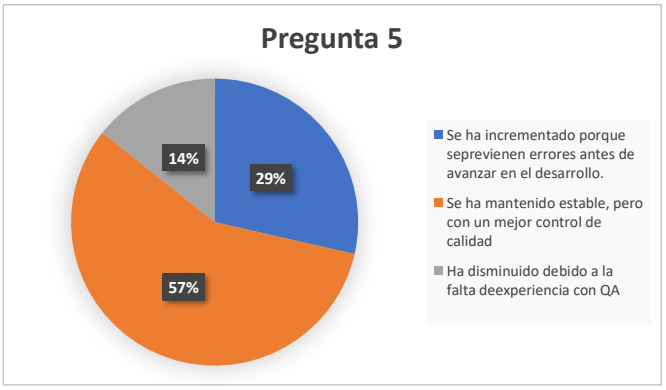


5. ¿Cómo ha cambiado la productividad del equipo con la implementación de QA?

   En el diagrama se puede observar que el 29 % de los desarrolladores ha incrementado prevenir errores con la implementación de QA, mientras que el 57 % se ha mantenido estable con un mejor control de calidad con la implementación de QA y por último el 14 % ha disminuido la productividad del equipo debido a la falta de experiencia con QA.

Figura 5.10

Gráfico de la pregunta número 5

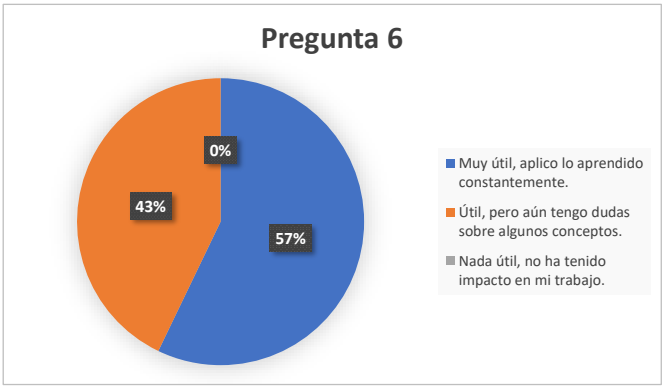


6. ¿Qué tan útil fue el curso de QA para aplicarlo en el día a día?

   En el diagrama se puede observar que el 57 % de los desarrolladores consideran que es muy útil aplicarlo lo aprendido día a día, mientras que el 43 % considera que es útil, pero aun tienen dudas con ciertos conceptos.

Figura 5.11

Gráfico de la pregunta número 6



CAPÍTULO V: PROPUESTA DE INVESTIGACIÓN

6.1. Tema

La presente investigación propone la integración de un sistema de QA Testing en el desarrollo de software, enfocado en optimización del código mediante la integración de prácticas de QA Este enfoque contempla la participación de especialistas en pruebas de calidad, quienes serán responsables de evaluar y verificar la funcionalidad y fiabilidad del código desarrollado.

6.2. Objetivo

Este programa abarca áreas clave como pruebas automatizadas, pruebas de integración, pruebas de rendimiento y seguridad. El objetivo es establecer estándares elevados de calidad mediante la implementación de mejores prácticas en el ciclo de desarrollo, lo que permitirá minimizar errores en etapas tempranas, optimizar el rendimiento de la aplicación y reducir los costos asociados a la corrección de defectos. Las metodologías de prueba actualizadas incluirán el uso de herramientas modernas como Cypress y GitLab, así como la adopción de prácticas ágiles que faciliten la detección temprana de problemas y aseguren la calidad continua del producto. Además de la implementación de un sistema de QA Testing de guía especifico para la aplicación ANI

6.3. Entorno de la aplicación

6.3.1. GitLab(Backend)

La arquitectura del proyecto contempló el establecimiento de un marco de validación riguroso, fundamentado en la sinergia entre las plataformas GitLab y SonarQube. Esta integración técnica facilita la verificación sistemática del código y la validación exhaustiva de funcionalidades mediante procesos automatizados de análisis. La estructuración del flujo operativo se diseñó priorizando la excelencia técnica en cada fase, desde el desarrollo inicial hasta el despliegue en ambientes productivos.

6.3.2. Cypress(Frontend)

Las pruebas automatizadas se realizó utilizando Cypress, una herramienta de automatización enfocada en probar el Frontend de la aplicación. Se utilizó para validar la correcta visualización de los componentes, la navegación entre páginas y la interacción del usuario con la interfaz. Gracias a su compatibilidad con TypeScript, fue posible simular acciones como clics, ingreso de datos y verificación de elementos, asegurando que el comportamiento del Frontend sea consistente y funcional en diferentes escenarios.

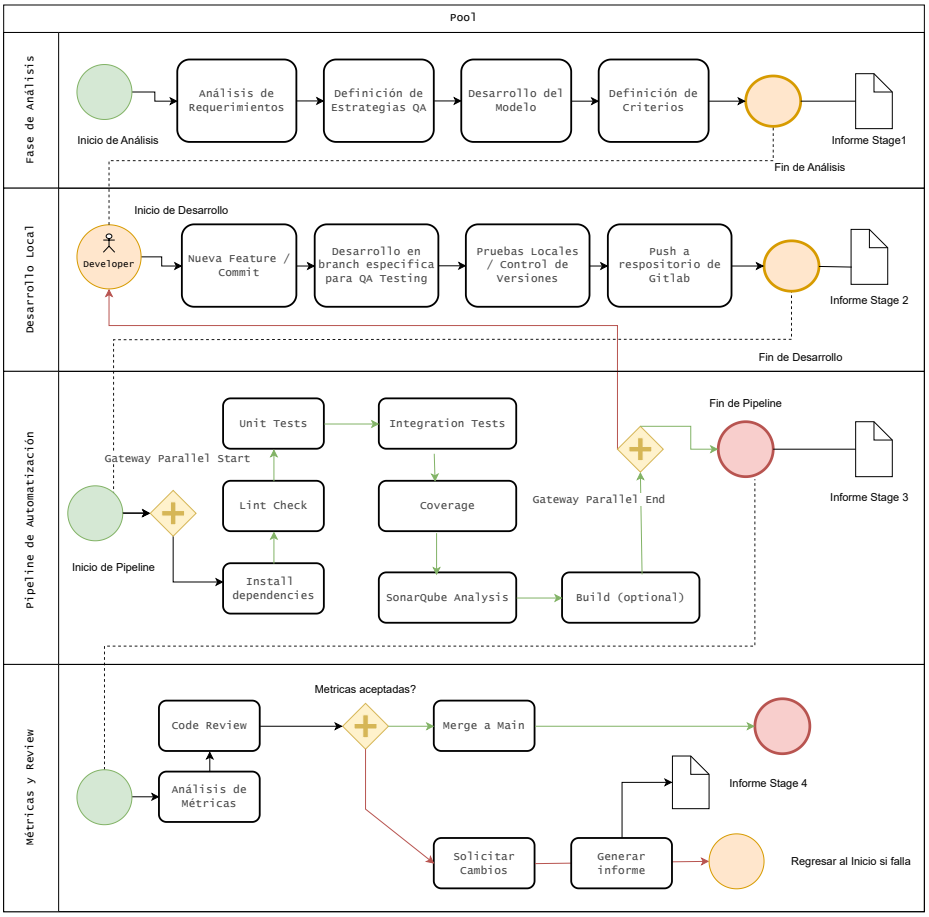
6.4. Estructura de la aplicación

6.4.1. Diagrama GitLab(Backend)

En la figura 6.1, se muestra que al desarrollar localmente una nueva funcionalidad, el código pasa a un proceso automatizado en GitLab. Las dependencias se instalan y se ejecutan pruebas (unitarias, integración, estilo con Lint, cobertura con SonarQube, construcción). Si todas las validaciones pasan, se obtienen métricas de calidad (cobertura, rendimiento y seguridad). En caso de falla, el pipeline se detiene y se solicitan correcciones. Al cumplir los estándares, se crea una Merge Request para revisión y, al ser aprobada, el código se fusiona a la rama principal. Esto garantiza la calidad continua del software y un flujo claro para la entrega de soluciones robustas.

Figura 6.1

Gráfico Business Process Model and Notation de la propuesta. Para una mejor visualización, visite: https://isystems.digital/bpmn/



6.5. Implementación de Control de Calidad en Backend NestJS

El documento proporciona una descripción detallada de la configuración y operación de un sistema de control de calidad para un proyecto de desarrollo de software utilizando NestJS y TypeScript. Se comienza estableciendo un entorno de desarrollo sólido con la instalación de tecnologías clave como Node.js y npm, que constituyen la base para el despliegue de herramientas adicionales y la gestión de dependencias necesarias para asegurar la calidad del código.

El enfoque principal del documento es la implementación de un pipeline de integración y entrega continua (CI/CD) utilizando GitLab. Este pipeline está estructurado en múltiples fases: instalación de dependencias, linting del código, ejecución de pruebas unitarias, generación de informes

de cobertura y análisis de calidad del código a través de SonarQube. Cada fase se define claramente en un archivo '.gitlab-ci.yml', asegurando que cada etapa contribuya eficazmente a la detección y corrección temprana de errores y vulnerabilidades en el código.

Se describe cómo se ejecutó SonarQube localmente sin el uso de Docker, facilitando un análisis exhaustivo y continuo de la calidad del código. La configuración de SonarQube se realiza a través de un archivo 'sonar-project.properties', que detalla los parámetros necesarios para el análisis, como exclusiones, inclusiones y rutas de los informes de cobertura. Este análisis es fundamental para identificar proactivamente vulnerabilidades, errores y çode smells", mejorando significativamente la calidad del software.

El documento concluye con una evaluación de los resultados obtenidos y los desafíos técnicos enfrentados durante la implementación. Resalta la efectividad del sistema de control de calidad para automatizar pruebas, incrementar la transparencia sobre la calidad del software y detectar anticipadamente problemas potenciales, estableciendo una metodología robusta para el mantenimiento y mejora continua de la calidad del código a lo largo del ciclo de vida del proyecto.

La implementación de herramientas de control de calidad en un proyecto de desarrollo es fundamental para mantener estándares altos y detectar problemas tempranamente. En este proceso, se implementó una solución completa de control de calidad para el backend desarrollado en NestJS, integrando pruebas unitarias, análisis estático de código y métricas de cobertura.

6.6. Guía de Informes y Procesos Completos (ANI)

Ver guía completa del Quality Assurance (ANI)

6.6.1. Requisitos Previos

Para implementar un sistema robusto de control de calidad, fue necesario contar con:

- Repositorio de código en GitLab
- Proyecto NestJS/TypeScript
- Docker instalado (para SonarQube)

- Node.js y npm configurados correctamente

Estos elementos constituyeron la base sobre la cual se construyó todo el ecosistema de calidad. Informe Completo

6.6.2. Implementación del Pipeline de CI/CD

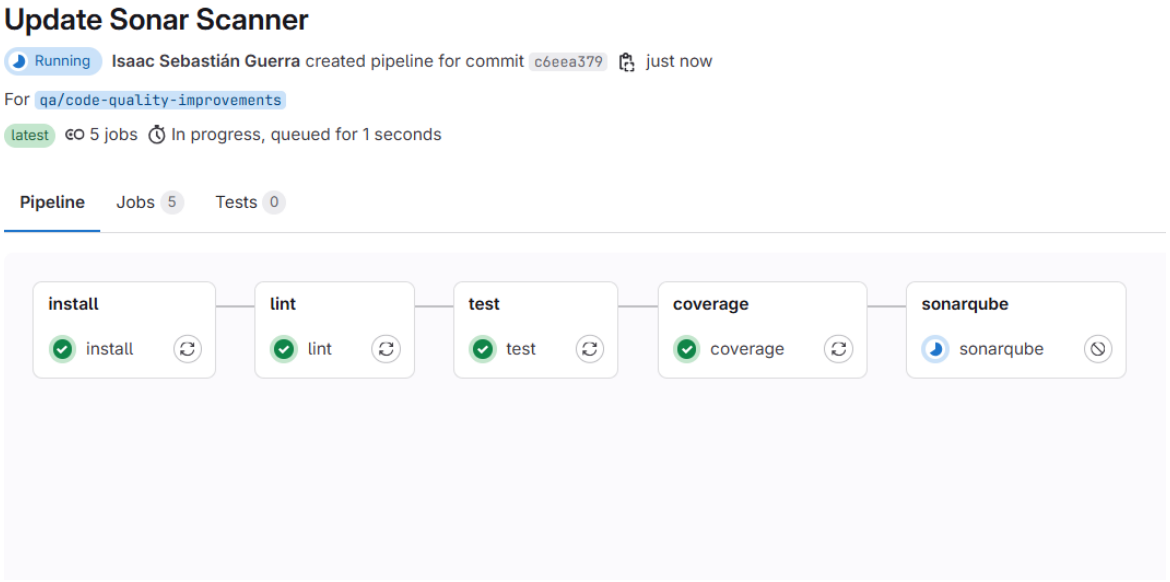
El proceso comenzó con la configuración de un pipeline de CI/CD en GitLab que incluye múltiples etapas, cada una enfocada en un aspecto específico de la calidad del código:

1. **Instalación de dependencias**: Preparación del entorno necesario
2. **Linting**: Análisis estático básico del código
3. **Testing**: Ejecución de pruebas unitarias
4. **Coverage**: Generación de reportes de cobertura
5. **SonarQube**: Análisis de calidad de código

La implementación se realizó a través de un archivo '.gitlab-ci.yml' que define estas etapas y sus respectivas acciones.

Figura 6.2

Ejecución del pipeline mostrando las etapas completadas



6.6.3. Configuración de Tests

El proyecto se estructuró con tests unitarios organizados en carpetas dedicadas, siguiendo las mejores prácticas de desarrollo:

```
ani-web-api/
|- src/
 test/
    unit/
        auth/
        users/
 coverage/
```

Los tests fueron implementados usando Jest, configurado para generar reportes de cobertura en varios formatos (lcov, text, cobertura) necesarios para su integración con herramientas de análisis.

Figura 6.3

Resultados de la ejecución de pruebas unitarias

6.6.4. Integración con SonarQube

Para el análisis detallado de calidad de código, se implementó SonarQube, una plataforma que permite detectar vulnerabilidades, bugs y code smells:

Instalación y Configuración

1. **Instalación de SonarQube**: Se utilizó Docker para ejecutar SonarQube localmente
2. **Configuración del proyecto**: Se creó un proyecto en SonarQube y se generó un token de acceso

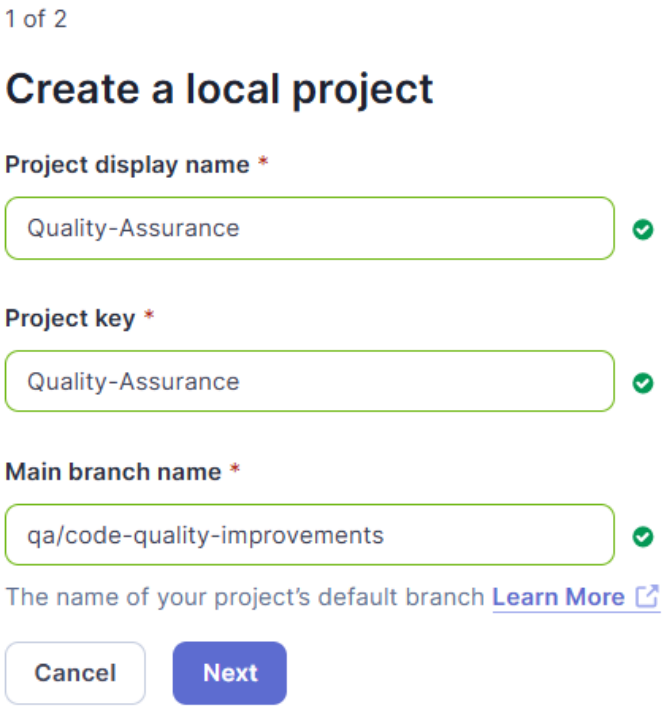
6.7. Informe de Configuración Sonarqube

Ver Sección Instalación y Configuración de Sonarqube

3. **Configuración de SonarScanner**: Se añadió un archivo `sonar-project.properties` en la raíz del proyecto

Figura 6.4

Interfaz de SonarQube mostrando la creación del proyecto



Archivo de Configuración de SonarQube

El archivo `sonar-project.properties` fue configurado con los siguientes parámetros:

```
# Información del proyecto
sonar.projectKey=Quality-Assurance
sonar.projectName=Quality-Assurance
sonar.projectVersion=1.0

# Rutas principales
sonar.sources=src
sonar.tests=test/unit
```

Figura 6.5

Interfaz de SonarQube corriendo localmente

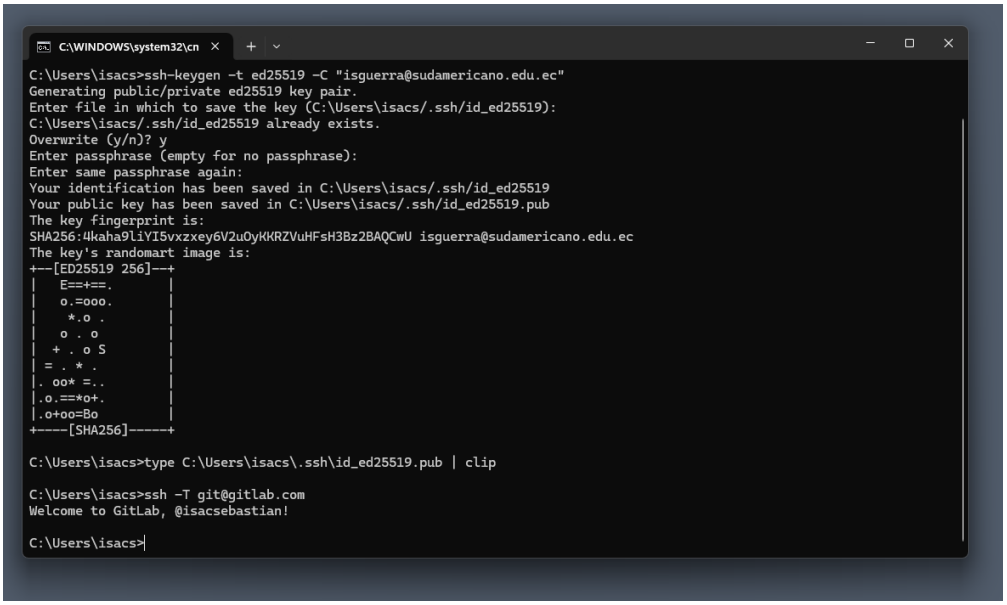
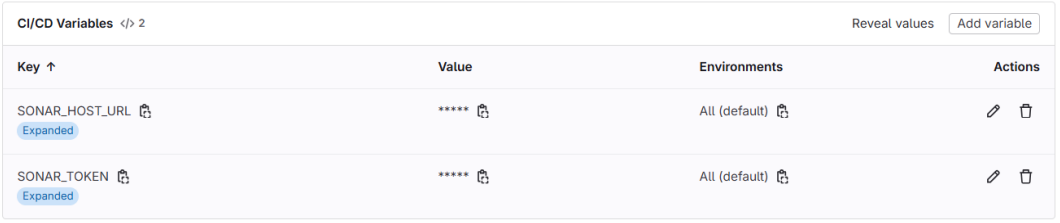


Figura 6.6

Configuración del token de acceso para SonarQube




```
# Exclusiones y patrones
sonar.exclusions=node_modules/**,coverage/**,dist/**
sonar.test.inclusions=test/unit/**/*.spec.ts
sonar.coverage.exclusions=test/**/*,src/**/*.spec.ts

# Reportes
sonar.typescript.lcov.reportPaths=coverage/lcov.info
sonar.javascript.lcov.reportPaths=coverage/lcov.info
```

```
# Configuración adicional
sonar.verbose=true
sonar.qualitygate.wait=true
```

Figura 6.7

Dashboard mostrando las métricas del proyecto con Jest

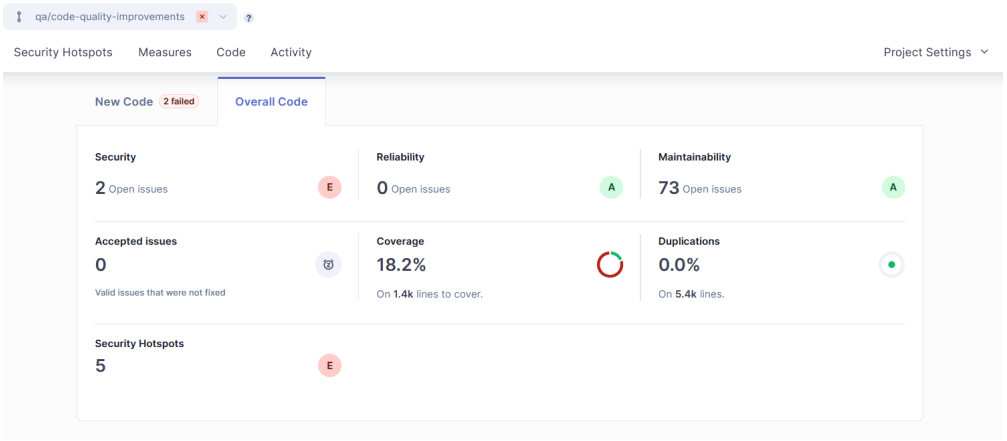
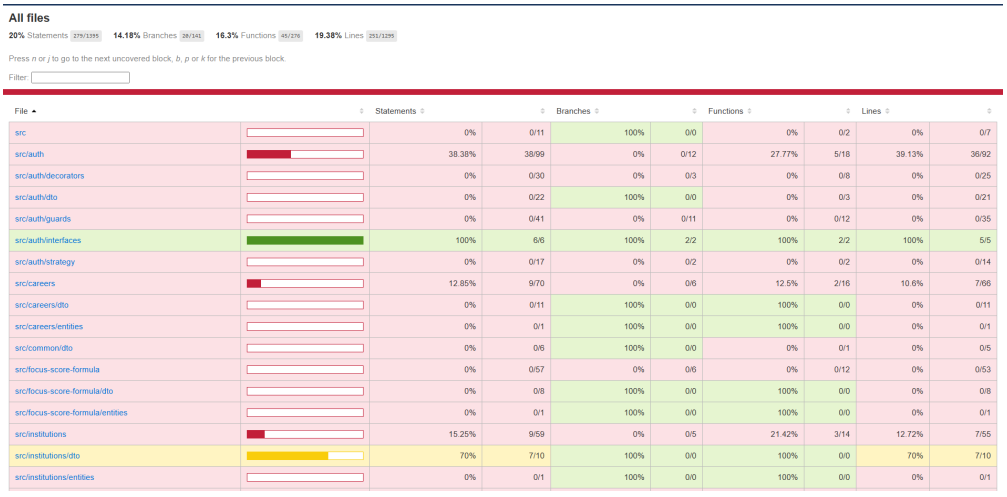


Figura 6.8

Dashboard de SonarQube mostrando las métricas del proyecto



6.7.1. Resultados y Métricas

El pipeline completo logró implementar un sistema integral de calidad, obteniendo:

- Pruebas unitarias automatizadas (69 tests pasando)
- Cobertura de código del 20 % del total de la aplicación
- Análisis de calidad de código con métricas detalladas
- Detección temprana de code smells y deuda técnica

6.7.2. Desafíos y Soluciones

Durante la implementación se enfrentaron diversos desafíos técnicos que requirieron soluciones creativas:

6.7.3. Conclusiones

La implementación de un sistema de control de calidad en el proyecto backend de NestJS ha proporcionado:

- Un proceso automatizado de validación de código
- Mayor visibilidad sobre la calidad del software
- Detección temprana de problemas potenciales
- Base sólida para el mantenimiento continuo

Esta implementación establece una metodología robusta para mantener la calidad del código a lo largo del ciclo de vida del proyecto.

6.8. Fases de Informes con Cypress(Frontend)

En la estructura de la aplicación Cypress se desarrolla un flujo sistemático como se muestra en la figura 6.9. El proceso consta de varias fases por cumplir, comenzando con:

- La primera fase consta de un análisis detallado de los requerimientos de la aplicación,luego se identifica las partes de la aplicación que serán probadas, también se define aquellas pruebas fuera de alcance de Cypress, es decir, pruebas no compatibles con la herramienta Cypress. Además, las estrategias de pruebas automatizadas en este caso es la herramienta que se utiliza para el testing de la aplicación utilizando Cypress con Mocha y finalmente se presenta un resultado sobre todo lo que se va analizar en la aplicación. Toda esta información se documenta en un informe detallado correspondiente a la fase 1, el cual incluye las especificaciones mencionadas.

  Hacer click para más detalles, consulte el Informe Técnico de Cypress con la aplicación de ANI

- La segunda fase consta del desarrollo de pruebas automatizadas, primero se configura el entorno de herramienta que se utilizara en este caso Cypress con Mocha. Además, se configura el código para comenzar las pruebas automatizadas de la aplicación, cuando este listo el código se ejecuta el test de prueba y cumple una condición que si pasa el test se realiza un informe detallado sobre la funcionalidad correcta de la aplicación. En el siguiente link puede observar un informe detallado de la fase 2. Configurando las herramientas y el código para las pruebas automatizadas.

  Hacer click para más detalles, consulte el Informe de Validación de Pruebas Automatizadas

- La tercera fase se activa cuando no se cumple con la condición establecida en la segunda fase, quiere decir, cuando el test ejecutado no cumple con las validaciones asignadas pasa por una fase de errores detallados, el cual se captura imágenes del error del código y al final de verificar el error encontrado se realiza el informe detallado del error encontrado.

En el siguiente link puede observar un informe detallado de la fase 3, que documenta los errores en las pruebas automatizadas de la aplicación.

Hacer click para más detalles, consulte el Informe Detallado de Errores en Pruebas Automatizadas

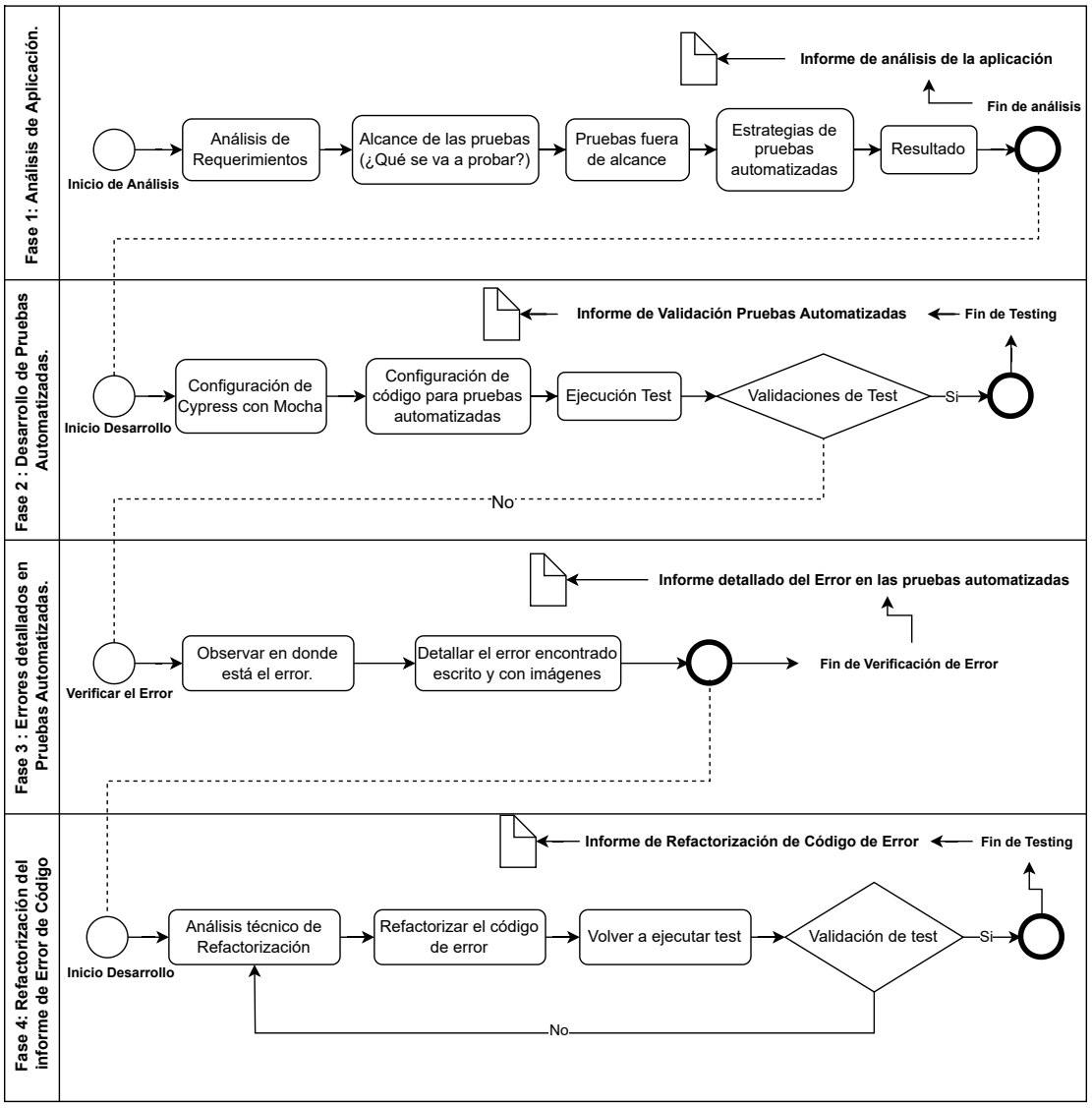
- La cuarta fase es la refactorización del código de error, se inicia con un análisis técnico del código, luego se re-factoriza el código de error, para luego ejecutar y verificar el cumplimiento de las condiciones establecidas. Una vez validad la refactorización, se manda un informe que documenta las correcciones implementadas y permitiendo así continuar con los demás tests. En caso de que el test no sea valido, tendrá que volver a empezar por analizar los requerimientos de la aplicación.

  En el siguiente link puede obervar un informe detallado de la fase 4, que documenta la refactorización del código.

  Hacer click para más detalles, consulte el Informe de Refactorización del Código

Figura 6.9

Fases de informes en pruebas automatizadas de Cypress Ayavaca (2025)



6.9. Configuración de Cypress para entornos de pruebas automatizadas

Se implementó una configuración especifica para la aplicación ANI, destinada para la ejecución de pruebas automatizadas. Cypress y el framework Mocha permitirán verificar el cumplimiento de las validaciones establecidas en los requerimientos de la aplicación.

En la figura 6.10 se observa el archivo package.json que representa la configuración principal del proyecto, donde se establece “ani_web” como nombre del proyecto en su versión “1.0.0”, utilizando TypeScript como lenguaje principal. La configuración específica para las pruebas auto-

matizadas se define en la sección “scripts”, donde se especifica el comando “test-chrome1” que ejecuta Cypress con parámetros específicos para realizar pruebas end-to-end (e2e) en el navegador Chrome, enfocándose en el archivo de prueba “loginUser.cy.ts”. El proyecto es desarrollado bajo la licencia *Internet Systems Consortium* (ISC), y utiliza como dependencia de desarrollo Cypress en su versión “14.0.0”, la cual es fundamental para la ejecución de las pruebas automatizadas en el entorno configurado.

Figura 6.10

Configuración de archivo Cypress

package.json > ...

```
{
  "name": "ani_web",
  "version": "1.0.0",
  "main": "ts",
  Debug
  "scripts": {
    "test-chorme1": "cypress run --spec cypress/e2e/loginUser.cy.ts --browser chrome "
  },

  "author": "Team Dev",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "cypress": "^14.0.0"
  }
}
```

En la figura 6.11 tiene un script de pruebas automatizadas desarrollado en Cypress para un sistema de autenticación y registro de usuarios, donde se definen dos bloques principales de prueba: el primero está enfocado en la funcionalidad de inicio de sesión, que verifica la interacción con el formulario de acceso validando la visibilidad de los campos de correo institucional y contraseña, permitiendo el ingreso de credenciales específicas; el segundo bloque prueba la interfaz de usuario, que realiza una secuencia de acciones para el registro de un nuevo usuario, incluyendo la navegación a la página de usuarios, el ingreso de datos personales como nombre, apellido, edad, la selección del género y la confirmación del registro mediante botones específicos, que permiten interactuar con los elementos de la interfaz web.

Figura 6.11

Script de Pruebas Automatizadas

```
cypress > e2e > loginUser.cy.ts > ...
describe('Login Page Testing', () => {

  it('login User', () => {
    cy.visit('http://localhost:5173/login')
    cy.get('#login_institutionalEmail').should("be.visible").type("kmquito@sudamericano.edu.ec")
    cy.get('#login_password').should("be.visible").type("kevin1234")

  })

  it("Interfaz-Usuario", () => {

    cy.visit("http://localhost:5173/users")
    cy.get('.ant-btn-link').should("be.visible").click()
    cy.get('#firstName').should("be.visible").type("Isaac")
    cy.get('#lastName').should("be.visible").type("Guerra")
    cy.get('#age').type("19")
    cy.get('#sex').click()
    cy.get('.ant-select-item-option').contains('Male').click()
    cy.get('.steps-action > .ant-btn').click()
  })


})
```

6.9.1. Cypress

En la siguiente Figura 6.12 se observa el levantamiento de la aplicación del proyecto de investigación.

Figura 6.12

Levantamiento de Proyecto de Investigación

```
PS C:\Users\TUF GAMING\OneDrive\Escritorio\ani-web-app-QA-Test-Env> npm run dev
>>

> frontend_2@0.0.0 dev
> vite


  VITE v5.4.11  ready in 1626 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

En la siguiente Figura 6.13 se observa el levantamiento del framework de Cypress para comenzar con las pruebas automatizadas.

Figura 6.13

Levantamiento de framework Cypress

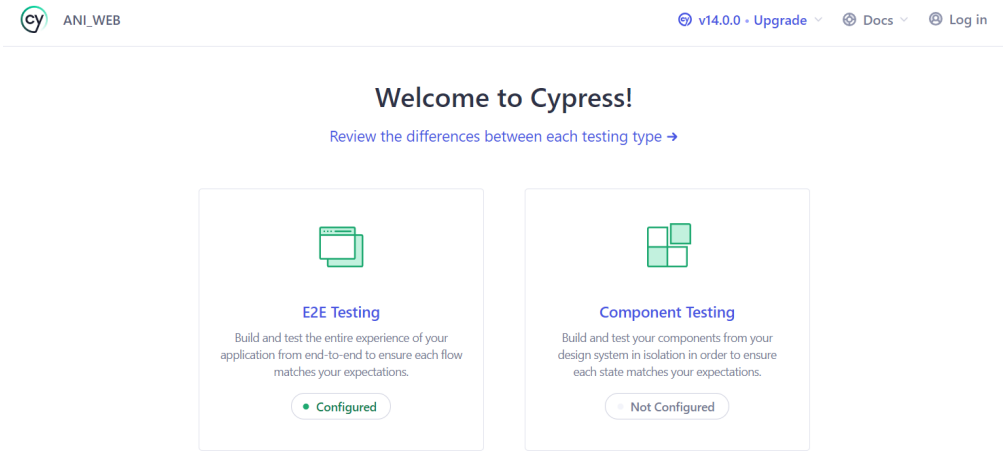
```
PS C:\Users\TUF GAMING\OneDrive\Escritorio\ANI_WEB> npx cypress open


DevTools listening on ws://127.0.0.1:49800/devtools/browser/459808f4-4793-4488-8a06-991b99981993
Missing baseUrl in compilerOptions. tsconfig-paths will be skipped
```

Al ejecutar Cypress, la Figura 6.14, se visualiza la pantalla de inicio de Cypress.

Figura 6.14

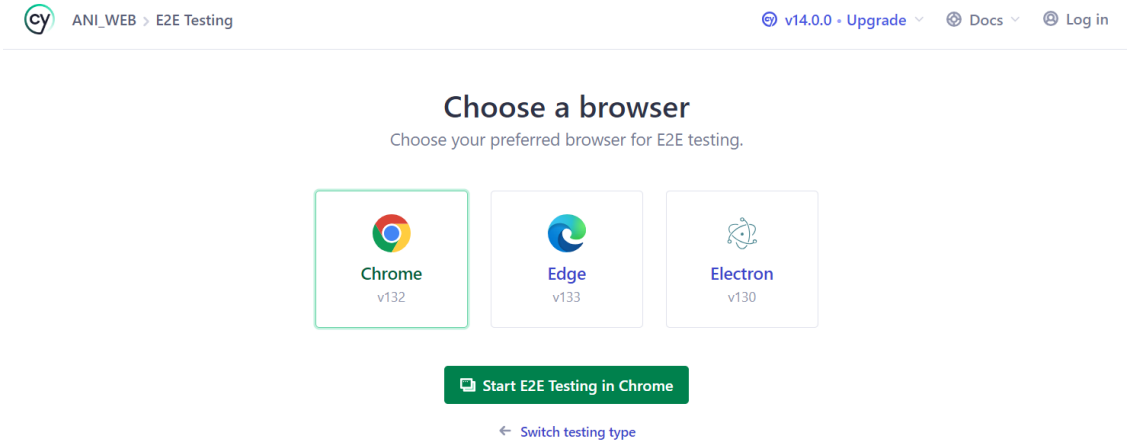
Página inicial Cypress.



La siguiente Figura 6.15 observa el tipo de navegador que se utilizara, en este caso se utiliza Chorme.

Figura 6.15

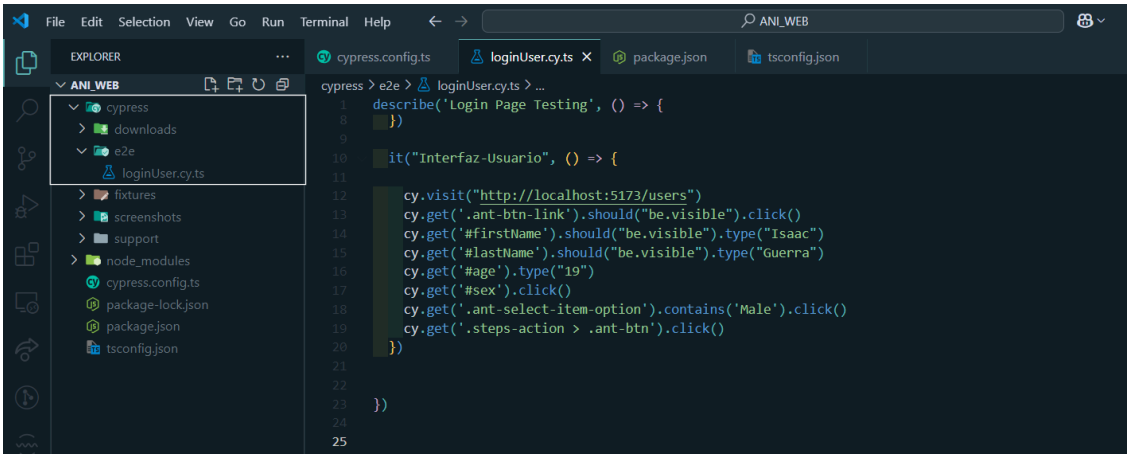
Selección de navegador.



En la Figura 6.16 se muestra un archivo creado dentro de una ruta de carpetas, para configurar el entorno de pruebas automatizadas.

Figura 6.16

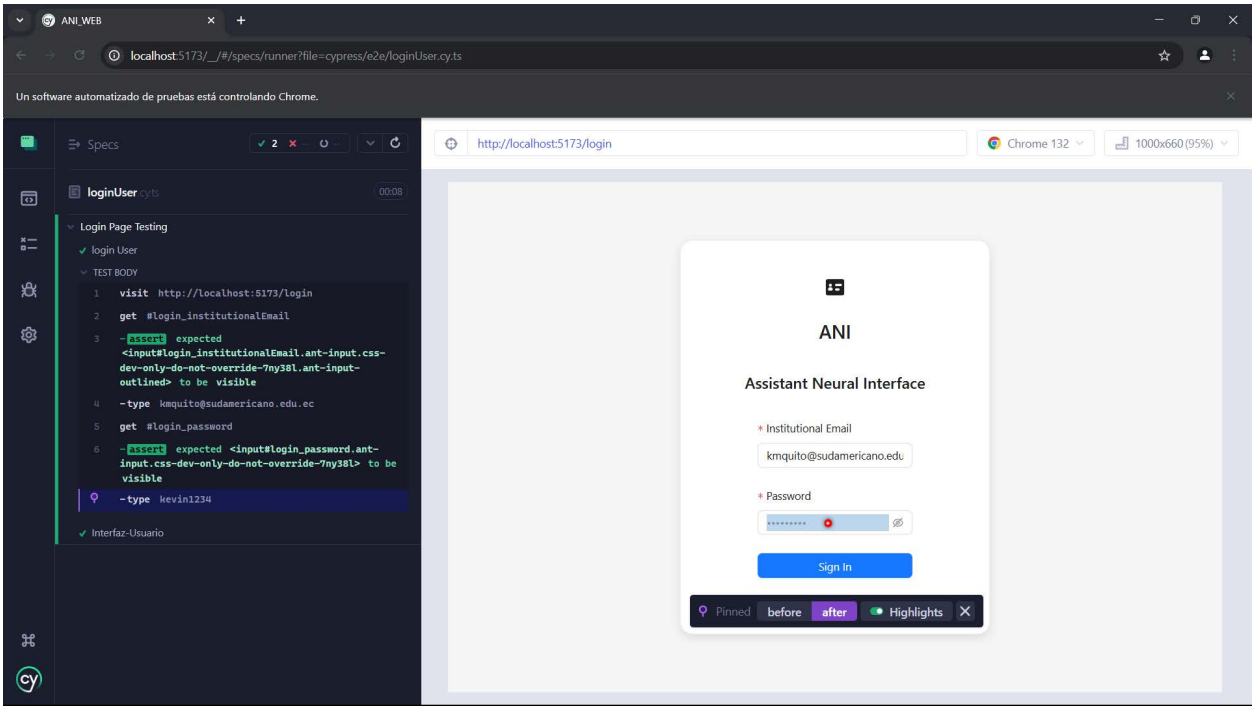
Archivo de Test Cypress



La siguiente Figura 6.17, se analiza la prueba automatizada del inicio de sesión. Que cumple con las condiciones y seguridad para el inicio de sesión del usuario.

Figura 6.17

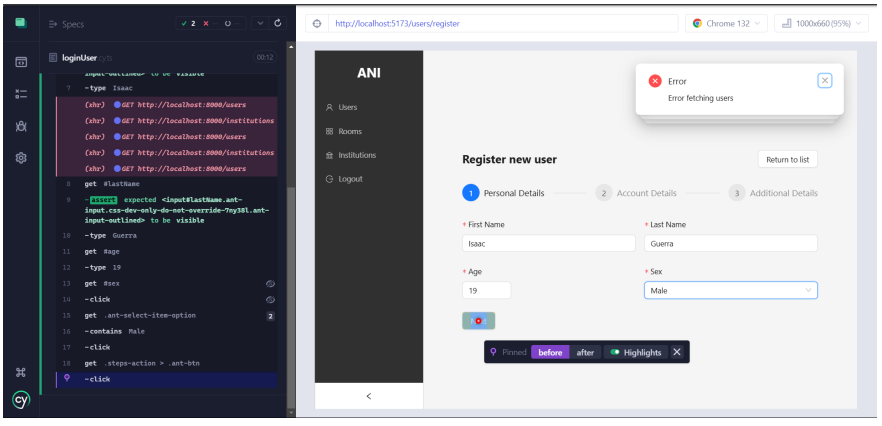
Prueba automatizada de Inicio de Sesión.



En la siguiente Figura 6.18 se muetra en pantalla que realiza las pruebas automatizadas, cumpliendo así las condiciones que tiene la aplicación para el registro de un nuevo usuario.

Figura 6.18

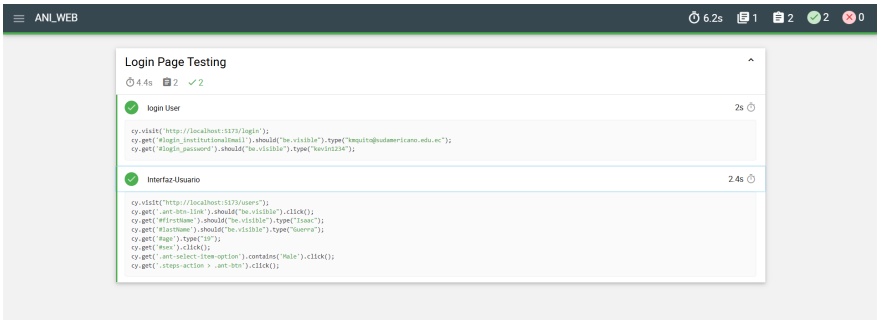
Prueba automatizada de Crear un Nuevo Usuario.



En la siguiente Figura 6.19 se muestra el resultado de Cypress con Mocha, lo que cumplió con el sistema de pruebas unitarias automatizadas y muestra el signo de aprobado las pruebas del test.

Figura 6.19

Prueba automatizada de Crear un Nuevo Usuario.



6.10. Guía Quality Assurance (ANI)

Hacer click de la guía completa del Quality Assurance (ANI)

CRONOGRAMA DE ACTIVIDADES

El gráfico de la Figura 7.1 muestra el cronograma completo de actividades desarrolladas en 21 semanas.

Figura 7.1

Para una mejor visualización, visite: https://isystems.digital/gantt/

Actividad	Inicio	Final
Fase 1: Investigación y Planificación		
Análisis inicial del estado actual	Oct 1	Oct 7
Revisión de literatura y mejores prácticas	Oct 8	Oct 14
Selección de herramientas QA	Oct 15	Oct 21
Definición del plan de implementación	Oct 22	Oct 28
Fase 2: Desarrollo de Guías y Material		
Desarrollo inicial de la guía QA	Oct 29	Nov 4
Creación de templates y estándares	Nov 5	Nov 11
Preparación material de capacitación	Nov 12	Nov 18
Revisión y ajuste de documentación	Nov 19	Nov 25
Fase 3: Implementación Técnica		
Configuración del ambiente de pruebas	Nov 26	Dic 2
Implementación de pruebas unitarias	Dic 3	Dic 9
Desarrollo de pruebas de integración	Dic 10	Dic 16
Automatización inicial de pruebas	Dic 17	Dic 23
Fase 4: Capacitación y Ajustes		
Preparación final del material	Dic 24	Dic 30
Sesión de capacitación 1	Dic 31	Ene 6
Sesión de capacitación 2	Ene 7	Ene 13
Retroalimentación y ajustes	Ene 14	Ene 20
Documentación de resultados iniciales	Ene 21	Ene 27
Fase 5: Evaluación y Cierre		
Evaluación de implementación	Ene 28	Feb 3
Análisis de resultados	Feb 4	Feb 10
Preparación de informe final	Feb 11	Feb 17
Cierre y presentación de resultados	Feb 18	Feb 24

CONCLUSION

El proyecto inició con una revisión completa de la aplicación y un previo análisis mediante de encuestas al equipo técnico del Instituto Tecnológico Particular Sudamericano, evidenciando la necesidad de potenciar y mejorar el conocimiento sobre calidad de software QA. Para atender la situación, se estructuró e impartió una capacitación específica en QA, facilitando herramientas y bases teóricas para la correcta implementación de pruebas previamente realizadas, para lo cual se obtuvo una notable mejora al desarrollar con metodologías de calidad, por parte del equipo, y a su vez en el manejo de herramientas contemporáneas como Cypress para automatización de pruebas, modificaciones y buenas practicas dentro del código y GitLab para gestión de código e integración continua.

El proceso completo consiguió una metodología robusta de aseguramiento de calidad y resultaron esenciales para identificar fallos en etapas tempranas y mejorar en gran manera los flujos de trabajo. Los procedimientos establecidos en la metodología fundamentados en investigación documental y experiencias exitosas dentro del ciclo de vida del desarrollo SDLC, permitieron construir un marco metodológico en Notion y en entornos dentro de repositorios propios de la aplicación que se materializó en documentación práctica para el equipo.

El aseguramiento de calidad efectuado reveló varios puntos de optimización específicos, consiguiendo para el equipo sugerencias técnicas validadas por docentes caacitadas en la metodología y tecnologías. El desarrollo y aproximación sistemático, sustentada en la robustez de tecnologias como SonarQube, GitLab y Cyrpress, además de optimizar la calidad del código y sus procesos enlazados, estableció cimientos sólidos para futuras implementaciones de QA en el Instituto Sudamericano. Los sistemas de automatización de pruebas demostraron su valor en la detección preventiva de errores y la confiabilidad del desarrollo, respaldado por el conjunto de pruebas implementado, marcaron y aseguraron calidad en el proceso de desarrollo teniendo un referente para la mejora continua de futuros proyectos institucionales.

RECOMENDACIONES

El desarrollo moderno de aplicaciones requiere una guía y un enfoque estructurado en la automatización de pruebas. La experiencia del proyecto propone implementar una organización metódica al realizar QA Testing con Cypress, GitLab, SonarQube, etc. contemplando próximos ajustes en los casos de prueba según las necesidades emergentes del equipo técnico y de la solicitud del cliente.

El conocimiento que debe tener el equipo de desarrollo, representa un punto fundamental para el éxito del proceso. Es de suma importancia proporcionar Capacitación específica en el manejo de herramientas como Cypress y GitLab, profundizando estos procesos de técnicas avanzadas de aseguramiento de calidad. Esta formación técnica reduce obstáculos durante la fase de implementación y desarrollo, mejorando la comprensión de las estructuras de prueba y mejorando la precisión en la identificación temprana de defectos. La facilidad y versatilidad en la implementación debe tomar en cuenta diversos entornos técnicos, respaldando procesos de integración continua.

El equipo de desarrollo, junto con la dirección del proyecto de los docentes capacitados, necesita manipular las pruebas automatizadas con la infraestructura actual, implementando programas de actualización en metodologías QA. La experiencia demuestra y corrobora la importancia de mantener registros técnicos actualizados y documentados, implementar protocolos rigurosos de versionamiento en GitLab y efectuar evaluaciones sistemáticas del código fuente al hacer refactoring. Esp garantiza niveles altos de calidad, asegurando la robustez del sistema a largo plazo. La adopción de una metodología en la estructuración de pruebas mediante Cypress, combinada con prácticas consistentes de control de versiones en GitLab, y buenas prácticas de SonarQube emerge como factor esencial del éxito. La configuración apropiada de entornos de desarrollo y pipelines de CI / *Continuous Delivery* (CD) requiere atención cuidadosa, asegurando así la eficacia del proceso de pruebas y la optimización continua del desarrollo dentro de la aplicación.

Bibliografía

Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.

Aniche, M., Maziero, E., Durelli, R., & Durelli, V. H. (2022). The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring. *IEEE Transactions on Software Engineering*, *48*, 1432-1450. https://doi.org/10.1109/TSE.2020.3021736

Ayavaca, K. M. Q. (2025). *Fases de informes en pruebas automatizadas de Cypress*.

Bakharev, N. (2023). Unit Testing: Definition, Examples, and Critical Best Practices. *Viitattu*, *9*, 2024.

Baldassarre, M. T., Lenarduzzi, V., Romano, S., & Saarimäki, N. (2020). On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube. *Information and Software Technology*, *128*. https://doi.org/10.1016/j.infsof.2020.106377

Bhanushali, A. (2023). INTERNATIONAL JOURNAL OF ADVANCES IN ENGINEERING RESEARCH Ensuring Software Quality Through Effective Quality Assurance Testing: Best Practices and Case Studies. *International Journal of Advances in Engineering Research (IJAER) 2023*, *26*. http://www.ijaer.com

Buckley, F. J., & Poston, R. (1984). Software Quality Assurance. *IEEE Transactions on Software Engineering*, *SE-10*, 36-41. https://doi.org/10.1109/TSE.1984.5010196

Choudhury, P., Crowston, K., Dahlander, L., Minervini, M. S., & Raghuram, S. (2020). GitLab: work where you want, when you want. *Journal of Organization Design*, *9*. https://doi.org/10.1186/s41469-020-00087-8

Colina, F. J. G., Hernández, S. C. J., & García, L. S. (2018). Gestión escolar y calidad educativa. *http://scielo.sld.cu/scielo.php?script=sci$_a$rttextpid* $= S0257 - 43142018000200016$, *37*(2), 89-109.

Goericke, S. (2019, enero). *The Future of Software Quality Assurance*. Springer International Publishing. https://doi.org/10.1007/978-3-030-29509-7

Gowda, P. G. A. N. (2019). TypeScript vs. JavaScript: A Comparative Analysis.

Kaur, G., & Singh, B. (2017). Improving the quality of software by refactoring. *Proceedings of the 2017 International Conference on Intelligent Computing and Control Systems, ICICCS 2017*, *2018-January*, 185-191. https://doi.org/10.1109/ICCONS.2017.8250707

KC, P. (2019). *NETWORK TEST SYSTEM FOR CONTINUOUS INTEGRATION* [Tesis doctoral, BMS COLLEGE OF ENGINEERING].

Kitchenham, B. A., Hughes, R. T., & Linkman, S. G. (2001). Modeling software measurement data. *IEEE Transactions on Software Engineering*, *27*, 788-804. https://doi.org/10.1109/32.950316

Masso, J., Pino, F. J., Pardo, C., García, F., & Piattini, M. (2020, agosto). Risk management in the software life cycle: A systematic literature review. https://doi.org/10.1016/j.csi.2020.103431

Mishra, A., & Otaiwi, Z. (2020, noviembre). DevOps and software quality: A systematic mapping. https://doi.org/10.1016/j.cosrev.2020.100308

Monje Morales, A. (2023, septiembre). *Automated Front-End Website Testing with Cypress* [Bachelor's Thesis]. Centria University of Applied Sciences. https://www.theseus.fi/bitstream/handle/10024/806779/Morales_Alejandro.pdf

Nguyen, N. (2022). Creating a modern web user interface using react and typescript. *TypeScript vs. JavaScript: A Comparative Analysis."*

Ozkaya, I. (2021, mayo). Can We Really Achieve Software Quality? https://doi.org/10.1109/MS.2021.3060552

Rios, J. C. C., Kopec-Harding, K., Eraslan, S., Page, C., Haines, R., Jay, C., & Embury, S. M. (2019). A Methodology for using gitlab for software engineering learning analytics. *Proceedings - 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2019*, 3-6. https://doi.org/10.1109/CHASE.2019.00009

Rodríguez-Martínez, M., Roussopoulos, N., McGann, J. M., Kelley, S., Mokwa, J., White, B., & Jájá, J. (2001). Integrating distributed scientific data sources with MOCHA and XRoaster.

Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM, 263-266. https://doi.org/10.1109/SSDM.2001.938560

Runeson, P. (2006). A survey of unit testing practices. *IEEE Software*, *23*, 22-29. https://doi.org/10.1109/MS.2006.91

Tasnim Taky, M. (2021). *Automated Testing with Cypress* [Thesis] [3 Appendices].

Underwood, S. (2016). *Exploring Organizations' Software Quality Assurance Strategies* [Doctoral Study]. Walden University. https://www.waldenu.edu