Implementación de pruebas automatizadas en el ciclo de vida del software para optimizar la calidad y continuidad del servicio en la EPSI AIC

Anderson Alexandry Lopez Yace

Proyecto de grado entregado para obtener el título de
Magister en Ingeniería de Software

Dirigida por
MBA Juan Pablo García

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Maestría en Ingeniería de Software
Santiago de Cali
11 de julio de 2026

Santiago de Cali, 11 de julio de 2026.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Implementación de pruebas automatizadas en el ciclo de vida del software para optimizar la calidad y continuidad del servicio en la EPSI AIC" realizado por el estudiante de Magister en Ingeniería de Software Anderson Alexandry Lopez Yace (cod: 9032114), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

Juan Pablo García Cifuentes
Firmado digitalmente por Juan Pablo García Cifuentes
Fecha: 2026.07.14 09:36:09 -05'00'

MBA Juan Pablo García

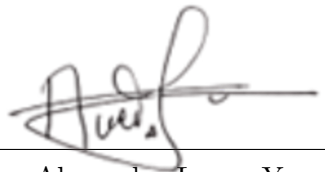
Santiago de Cali, 11 de julio de 2026.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Implementación de pruebas automatizadas en el ciclo de vida del software para optimizar la calidad y continuidad del servicio en la EPSI AIC" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,



Anderson Alexandry Lopez Yace
Código: 9032114

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Implementación de pruebas automatizadas en el ciclo de vida del software para optimizar la calidad y continuidad del servicio en la EPSI AIC

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Pruebas de Software
3. Tipo de proyecto: Aplicado
4. Estudiante: Anderson Alexandry Lopez Yace
5. Correo electrónico: andersonlopez@javerianacali.edu.co
6. Dirección y teléfono: Calle 56 BN No. 10-64 - 3122646967
7. Director: Juan Pablo García
8. Vinculación del director: Planta
9. Correo electrónico del director: jpgarcia@javerianacali.edu.co
10. Palabras clave: Aseguramiento de calidad, pruebas automatizadas, sistemas de información en salud, ISTQB, Pruebas de Software, técnicas de caja negra, continuidad operativa, EPSI.
11. Fecha de inicio: Enero 15 de 2026
12. Resumen: Este proyecto de grado propuso el diseño e implementación de un proceso de aseguramiento de calidad con automatización de pruebas para los sistemas de información con módulos misionales de la EPSI AIC (Entidad Promotora de Salud Indígena, Asociación Indígena del Cauca), entidad responsable de garantizar servicios de salud a comunidades indígenas en Colombia. La iniciativa respondió a la ausencia total de procesos formales de testing en la organización, situación evidenciada por 0 % de cobertura en pruebas automatizadas, 100 % de dependencia en validaciones manuales ejecutadas por un único responsable, y ausencia de métricas de calidad y trazabilidad de defectos según estimaciones declarativas. Esta deficiencia generaba incidentes recurrentes en producción que comprometían la continuidad operativa de servicios críticos de salud.

    El objetivo fue diseñar un proceso formal de aseguramiento de calidad que incluyera estructura organizacional, criterios de priorización de funcionalidades críticas, diseño sistemático de casos de prueba aplicando técnicas de caja negra (particiones de equivalencia, tablas de decisión y arreglos ortogonales), e implementación de automatización mediante un marco de trabajo escalable y mantenible. La metodología se enmarcó en Design Science Research (DSR), el cual

se estructuró en tres ejes: Investigación del Problema, Diseño del Tratamiento y Validación del Tratamiento, desarrollados a través de cuatro fases operativas secuenciales durante cinco meses.

Los resultados incluyeron un proceso documentado de aseguramiento de calidad adaptado al contexto organizacional de la EPSI AIC, casos de prueba diseñados formalmente con trazabilidad completa, un marco de automatización implementado y funcionando, y evidencia cuantitativa del impacto mediante indicadores de proceso (cobertura, productividad) y de producto (correctitud, confiabilidad). Este proyecto contribuyó al conocimiento en la implementación de prácticas de aseguramiento de calidad en organizaciones de salud con baja madurez en procesos de testing, generando lineamientos transferibles a contextos similares del sector.

Agradecimientos

El desarrollo de este trabajo de grado no habría sido posible sin el apoyo invaluable de diversas personas e instituciones que me acompañaron a lo largo de este proceso. A todos ellos, expreso mi más profundo y sincero agradecimiento.

En primer lugar, agradezco a **Dios** por guiar mis pasos y darme la fortaleza necesaria para culminar esta etapa. De igual forma, dedico este logro a mis padres, **Niray Lopez** e **Ismenia Yace**; sin su apoyo incondicional, sus enseñanzas y los valores que me han inculcado, no estaría hoy celebrando esta meta académica. De manera muy especial, agradezco a **Sandra Lorena Cuene Corpus**, no solo por ser mi incondicional soporte emocional a lo largo de este camino, sino también por su invaluable dedicación para ayudarme a resolver mis dudas y su apoyo constante en los momentos de mayor exigencia. Su acompañamiento, su paciencia y su respaldo integral en cada uno de los altibajos fueron el pilar que me permitió mantener la motivación y culminar este proyecto.

Agradezco y expreso mi profunda gratitud a mi director de trabajo de grado, **Juan Pablo García**, por su orientación, disposición y acompañamiento. Sus observaciones, conocimientos y compromiso fueron esenciales para fortalecer el rigor técnico y metodológico de la investigación. Asimismo, extiendo mi agradecimiento al comité evaluador por el tiempo dedicado a la revisión de este documento y por sus valiosas observaciones, las cuales enriquecieron el resultado final.

Quiero reconocer también la apertura y colaboración de los líderes y el equipo técnico de la **EPSI AIC**. Su disposición para abrir las puertas de la organización y participar activamente en el diagnóstico e implementación fue clave para llevar esta investigación a un entorno real y aplicable.

Finalmente, este trabajo de grado representa para mí mucho más que la culminación de un ciclo académico; ha sido un proceso de profundo crecimiento personal y profesional. Cada desafío superado durante esta investigación ha forjado en mí una nueva visión técnica y humana que me acompañará en mis futuros retos. Gracias a todos por creer en mí y ser parte de este viaje.

Resumen

Este proyecto de grado propuso el diseño e implementación de un proceso de aseguramiento de calidad con automatización de pruebas para los sistemas de información con módulos misionales de la EPSI AIC (Entidad Promotora de Salud Indígena, Asociación Indígena del Cauca), entidad responsable de garantizar servicios de salud a comunidades indígenas en Colombia. La iniciativa respondió a la ausencia total de procesos formales de testing en la organización, situación evidenciada por 0 % de cobertura en pruebas automatizadas, 100 % de dependencia en validaciones manuales ejecutadas por un único responsable, y ausencia de métricas de calidad y trazabilidad de defectos según estimaciones declarativas. Esta deficiencia generaba incidentes recurrentes en producción que comprometían la continuidad operativa de servicios críticos de salud.

El objetivo fue diseñar un proceso formal de aseguramiento de calidad que incluyera estructura organizacional, criterios de priorización de funcionalidades críticas, diseño sistemático de casos de prueba aplicando técnicas de caja negra (particiones de equivalencia, tablas de decisión y arreglos ortogonales), e implementación de automatización mediante un marco de trabajo escalable y mantenible. La metodología se enmarcó en Design Science Research (DSR), el cual se estructuró en tres ejes: Investigación del Problema, Diseño del Tratamiento y Validación del Tratamiento, desarrollados a través de cuatro fases operativas secuenciales durante cinco meses.

Los resultados incluyeron un proceso documentado de aseguramiento de calidad adaptado al contexto organizacional de la EPSI AIC, casos de prueba diseñados formalmente con trazabilidad completa, un marco de automatización implementado y funcionando, y evidencia cuantitativa del impacto mediante indicadores de proceso (cobertura, productividad) y de producto (correctitud, confiabilidad). Este proyecto contribuyó al conocimiento en la implementación de prácticas de aseguramiento de calidad en organizaciones de salud con baja madurez en procesos de testing, generando lineamientos transferibles a contextos similares del sector.

Palabras clave: Aseguramiento de calidad, pruebas automatizadas, sistemas de información en salud, ISTQB, pruebas de software, técnicas de caja negra, continuidad operativa, EPSI.

Abstract

This master's thesis proposed the design and implementation of a quality assurance process with test automation for information systems with core modules at EPSI AIC (indigenous Health Promoting Entity, Indígena Association of Cauca), the entity responsible for guaranteeing healthcare services to indigenous communities in Colombia. The initiative addressed the total absence of formal testing processes in the organization, a situation evidenced by 0 % coverage in automated tests, 100 % dependence on manual validations executed by a single individual, and a complete lack of quality metrics and defect traceability based on declarative estimates. This deficiency generated recurring incidents in production that compromised the operational continuity of critical healthcare services.

The objective was to design a formal quality assurance process including an organizational structure, prioritization criteria for critical functionalities, systematic test case design applying black-box techniques (equivalence partitioning, decision tables, and orthogonal arrays), and the implementation of automation through a scalable and maintainable framework. The methodology was framed within Design Science Research (DSR), structured into three core axes: Problem Investigation, Treatment Design, and Treatment Validation, which were developed across four sequential operational phases over a five-month period.

The results included a documented quality assurance process adapted to the organizational context of EPSI AIC, formally designed test cases with complete traceability, an implemented and functional automation framework, and quantitative evidence of the impact through process indicators (coverage, productivity) and product indicators (correctness, reliability). This project contributed to the body of knowledge on the implementation of quality assurance practices in healthcare organizations with low testing process maturity, generating transferable guidelines for similar contexts within the sector.

Keywords: Quality assurance, automated testing, health information systems, ISTQB, software testing, black-box techniques, operational continuity, EPSI.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La transformación digital en el sector salud ha generado una creciente dependencia de sistemas de información robustos y confiables para garantizar la continuidad de servicios críticos. En este contexto, las organizaciones del sector enfrentan el desafío de implementar prácticas efectivas que aseguren no solo la funcionalidad de sus sistemas, sino también su estabilidad y seguridad operativa. La ausencia de procesos formales de aseguramiento de calidad puede derivar en consecuencias que trascienden lo técnico, afectando directamente la prestación de servicios esenciales para la población.

La EPSI AIC, entidad responsable de garantizar el acceso a servicios de salud para comunidades indígenas del departamento del Cauca, opera bajo un modelo donde los sistemas de información constituyen el eje central de sus procesos misionales. Sin embargo, un diagnóstico preliminar reveló deficiencias estructurales en sus prácticas de desarrollo y despliegue de software. La organización presenta una cobertura nula en pruebas automatizadas, dependencia total de validaciones manuales ejecutadas por un único responsable, y ausencia de métricas que permitan cuantificar la calidad del software o rastrear defectos de manera sistemática. Esta situación genera incidentes recurrentes en producción que comprometen la estabilidad de los sistemas y, por extensión, la continuidad de los servicios de salud que atienden a poblaciones vulnerables.

El proyecto tuvo como objetivo diseñar e implementar un proceso de aseguramiento de calidad basado en pruebas automatizadas, adaptado al contexto organizacional y tecnológico de la EPSI AIC. La investigación se desarrolló bajo el paradigma de Design Science Research (DSR), mediante cuatro fases operativas: diagnóstico (Fase A), priorización y diseño de casos de prueba (Fase B), implementación del marco de automatización (Fase C) y evaluación del impacto (Fase D); enmarcadas en los tres ejes del ciclo DSR . Se definieron criterios de priorización basados en riesgo y criticidad, se diseñaron casos de prueba aplicando técnicas de caja negra reconocidas por el International Software Testing Qualifications Board (ISTQB) y se implementó un marco de automatización en módulos misionales de alto riesgo.

Los resultados incluyeron la documentación de un proceso formal de aseguramiento de calidad, la implementación de un marco de automatización funcional, la generación de casos de prueba con trazabilidad completa y evidencia cuantitativa del impacto mediante indicadores de cobertura de pruebas, productividad y reducción de defectos. La propuesta contribuyó al fortalecimiento de la continuidad operativa de los sistemas de información y aportó lineamientos transferibles para la implementación de prácticas de calidad en organizaciones del sector salud con baja madurez en procesos de testing.

1.1. Definición del problema

1.1.1. Planteamiento del problema

Actualmente, las organizaciones del sector salud enfrentan una creciente necesidad de modernizar sus procesos tecnológicos para brindar servicios más confiables y seguros, en respuesta a la creciente demanda de los usuarios y a las exigencias normativas. No basta con contar con sistemas robustos, sino que es necesario implementar prácticas efectivas de aseguramiento de la calidad que permitan reducir errores y asegurar la continuidad del servicio.

En la EPSI AIC, entidad responsable de garantizar el acceso a servicios de salud para comunidades indígenas, los sistemas de información son esenciales para gestionar y apoyar los procesos misionales de salud. Sin embargo, se ha identificado una deficiencia crítica en el proceso de desarrollo y despliegue de software: la carencia de pruebas automatizadas dentro del ciclo de vida del software.

Un diagnóstico organizacional preliminar, construido a partir de entrevistas estructuradas al equipo de desarrollo y revisión del proceso operativo, permitió estimar la magnitud de esta deficiencia. Los resultados diagnósticos estimados indican:

1. 0 % de cobertura en pruebas automatizadas a nivel unitario, de integración y regresión.
2. 100 % de dependencia en validaciones manuales ejecutadas por un único responsable, constituyendo un punto crítico de falla operacional.
3. Ausencia total de métricas de calidad, documentación de casos de prueba y trazabilidad de defectos.
4. Carencia completa de registros históricos que permitan cuantificar el impacto de los errores en producción.

Debido a la inexistencia de métricas históricas formales relacionadas con pruebas de software dentro de la organización, estas cifras deben interpretarse como una línea base diagnóstica construida a partir de evidencia declarativa del equipo técnico y análisis del proceso actual, práctica aceptada en estudios de Design Science Research cuando no existen registros cuantitativos previos.

Actualmente, no existe un proceso formal de diseño de pruebas. Las validaciones se ejecutan manualmente y se limitan a pruebas de usuario, donde los responsables de definir los requerimientos verifican su cumplimiento una vez que el desarrollo se entrega en un ambiente de pruebas. Esta práctica resulta insuficiente porque no incluye pruebas unitarias, de integración ni de regresión, lo que incrementa el riesgo de que errores no detectados lleguen a producción y afecten la operación normal de los sistemas.

Esta situación genera consecuencias documentadas tanto en la literatura especializada como en el diagnóstico organizacional realizado:

- Incidentes recurrentes en producción que afectan la disponibilidad de los sistemas y, por ende, la continuidad de los servicios de salud, situación que según (Knight and Wika, 1995) representa riesgos críticos en sistemas de salud donde la seguridad es prioritaria.

- Retrasos significativos en la detección y corrección de errores, elevando los costos operativos en una proporción que la literatura estima entre 3–5 veces superior comparada con organizaciones con procesos maduros de testing (Boehm, 1988).
- Dificultades para mantener la trazabilidad de los cambios y garantizar la calidad del software, limitando la capacidad de cumplimiento de la Resolución 866 de 2021 sobre seguridad de la información en salud (Ministerio de Salud y Protección Social, 2021).
- Riesgos de incumplimiento normativo, sanciones por errores en el manejo de información sensible, particularmente crítico en el contexto de datos de poblaciones indígenas.
- Pérdida de confianza entre usuarios, prestadores y entes de control en la capacidad tecnológica de la EPSI, evidenciada en las entrevistas realizadas al equipo de desarrollo.
- Dependencia operacional crítica del conocimiento no documentado de un único responsable de pruebas, generando un punto de falla que puede paralizar completamente los procesos de despliegue.

En el contexto actual, la EPSI AIC enfrenta desafíos relacionados con la confiabilidad y la calidad del software utilizado para la gestión de los procesos misionales en salud. Aunque la organización ha avanzado en la implementación de herramientas tecnológicas y en la integración de sistemas de información, persisten dificultades para garantizar la estabilidad de las aplicaciones en producción. Por lo tanto, el problema principal que enfrenta la EPSI AIC es:

> *"La EPSI AIC carece de procesos de pruebas automatizadas en el ciclo de vida del software, lo que limita el aseguramiento de la calidad, aumenta la probabilidad de fallos en producción y pone en riesgo la continuidad operativa de los sistemas de información en salud."*

Para ilustrar de manera clara la relación entre causas y efectos, se presenta a continuación el árbol del problema (Figura 1.1).

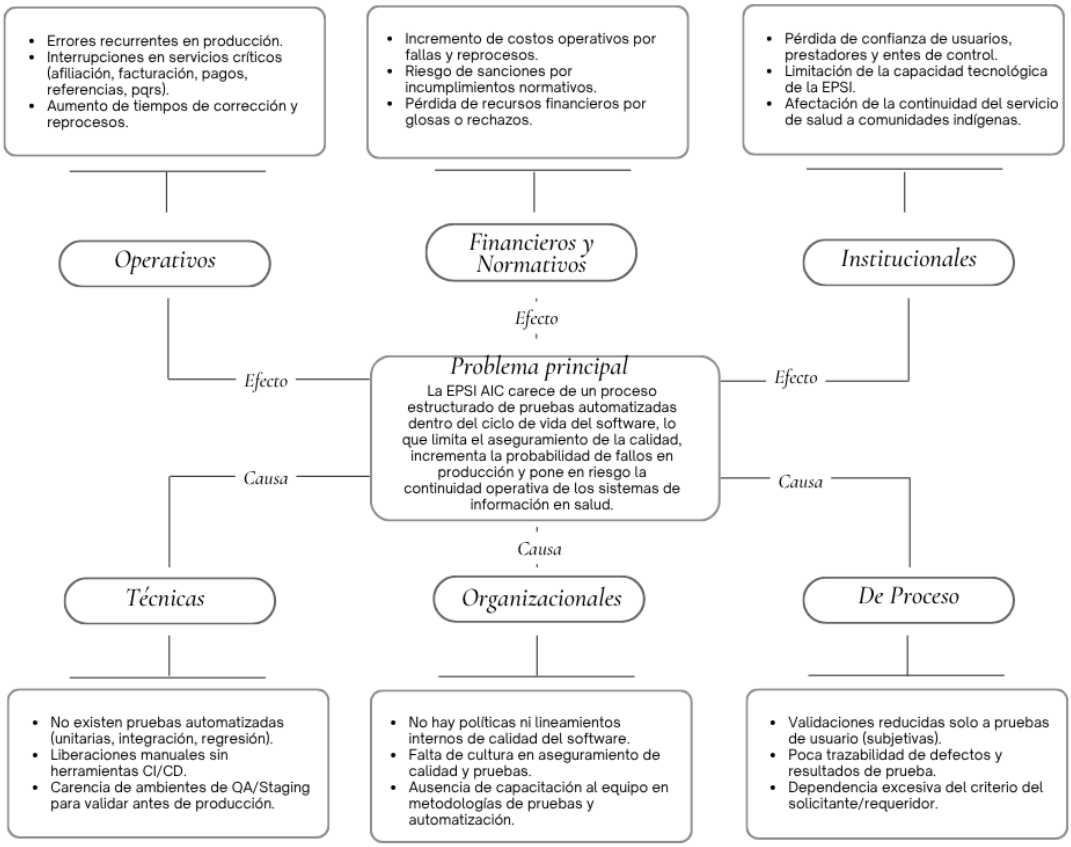


Figura 1.1: Árbol del problema. Elaboración propia

La Figura 1.1 ilustra cómo la ausencia de marcos de pruebas definidos genera efectos en cascada que comprometen la operación organizacional. La ausencia cuantificada de pruebas automatizadas (0 % de cobertura) limita la detección temprana de errores y crea una dependencia crítica en validaciones manuales realizadas por un único recurso humano. Esta situación, evidenciada mediante el diagnóstico participativo inicial, provoca que los ambientes de prueba carezcan de confiabilidad y que los despliegues a producción se ejecuten con alto riesgo de fallas en módulos críticos. La falta de un marco sistemático de pruebas impide la estandarización, el mantenimiento y la evolución de los procesos de aseguramiento de calidad, incrementando los costos operacionales y afectando directamente la continuidad del servicio en un contexto donde la estabilidad de los sistemas de salud es fundamental.

1.1.2. Formulación del problema

1. ¿Cómo puede diseñarse un proceso de aseguramiento de la calidad del software y la estructura organizacional que lo soporte, adaptados al contexto y restricciones de la EPSI AIC?

2. ¿Qué criterios deben establecerse para priorizar las funcionalidades y los casos de prueba a automatizar dentro del ciclo de vida del software?

3. ¿Qué técnicas de diseño de pruebas de caja negra resultan más adecuadas para definir casos de prueba efectivos y trazables para las funcionalidades priorizadas?

4. ¿Cómo puede implementarse la automatización de los casos de prueba priorizados para fortalecer el proceso de aseguramiento de la calidad?

5. ¿Cuál es el impacto del proceso de aseguramiento de calidad, soportado por pruebas automatizadas, en los indicadores de proceso (cobertura, productividad) y de producto (correctitud, confiabilidad) para la continuidad operativa de los sistemas de información en salud de la EPSI AIC?

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Diseñar e implementar un marco de aseguramiento de calidad basado en pruebas automatizadas para fortalecer la continuidad operativa en los sistemas de información de la EPSI AIC.

1.2.2. Objetivos específicos

1. Diseñar un proceso de aseguramiento de la calidad del software y definir la estructura organizacional que lo soporte, adaptado al contexto y restricciones de la EPSI AIC.

2. Establecer criterios de priorización para seleccionar las funcionalidades críticas y los casos de prueba a automatizar, basados en riesgo, criticidad y frecuencia de uso.

3. Diseñar casos de prueba efectivos y trazables para las funcionalidades priorizadas, aplicando técnicas de caja negra (particiones de equivalencia, tablas de decisión o arreglos ortogonales).

4. Implementar la automatización de los casos de prueba priorizados mediante un marco de trabajo escalable y mantenible.

5. Evaluar el impacto del proceso de aseguramiento de calidad mediante indicadores de proceso (cobertura, productividad) y de producto (correctitud, confiabilidad), para validar su contribución a la continuidad operativa de los sistemas de información en salud.

1.3. Delimitaciones y alcances

Este proyecto se enfocó en la implementación práctica de un marco de aseguramiento de calidad basado en pruebas automatizadas dentro del ciclo de vida del software de la EPSI AIC, orientado a fortalecer los procesos de aseguramiento de la calidad y a garantizar la continuidad operativa de los sistemas de información en salud.

1.3.1. Actividades incluidas en el alcance

El alcance comprende la realización de actividades específicas tales como:

- Realizar un diagnóstico del estado actual de las prácticas de pruebas y aseguramiento de calidad en la EPSI AIC para definir criterios de priorización de funcionalidades y casos de prueba.
- Diseñar casos de prueba efectivos y trazables para las funcionalidades priorizadas, aplicando técnicas de diseño adecuadas (por ejemplo, caja negra).
- Implementar un piloto del marco de aseguramiento de calidad mediante la automatización de pruebas en módulos misionales, evaluando su viabilidad técnica y funcionalidad.
- Evaluar el impacto del piloto mediante indicadores de proceso y producto (cobertura, productividad, correctitud, confiabilidad y continuidad operativa), para generar recomendaciones de escalamiento futuro.

1.3.2. Exclusiones del alcance

- No se realizará la automatización completa de todos los módulos y sistemas de la EPSI, dado que el proyecto tiene un carácter aplicado y piloto.
- No se abordará la modernización completa del ciclo DevOps, integración y despliegue continuo (CI/CD), ni la migración de infraestructura a la nube, aunque este proyecto puede sentar las bases para futuras iniciativas en estas áreas.
- No se contempla un plan de capacitación masiva para todos los equipos de la EPSI, limitándose a la socialización de resultados y lineamientos básicos para la adopción del marco implementado.

1.4. Justificación del trabajo de grado

En el contexto del sector salud, la calidad, seguridad y continuidad en los sistemas de información son elementos críticos para garantizar la prestación adecuada de los servicios a la población. La automatización de pruebas en el ciclo de vida del software se presenta como una herramienta indispensable para alcanzar estos objetivos, especialmente en organizaciones como la EPSI AIC, que gestionan sistemas de información con alto impacto en la salud de comunidades indígenas.

Al iniciar este proyecto, la EPSI AIC no contaba con procesos de pruebas de software automatizados. Su esquema de validación dependía enteramente de una sola persona, lo que incrementaba el riesgo de fallas en los sistemas de información en salud y afectaba la calidad y la continuidad del servicio. Esta situación generaba desconfianza en usuarios y entes de control, y tenía un impacto negativo en los procesos financieros de la organización, expuesta a reprocesos y observaciones.

Por ello, el proyecto se propuso implementar un marco de aseguramiento de calidad basado en pruebas automatizadas, integrando criterios de priorización, diseño de casos de prueba, automatización y evaluación del impacto. Este enfoque se alineó con normas de calidad del software como ISO/IEC 25010 (ISO, 2011) y con las tendencias de las metodologías ágiles y DevOps, en las que las pruebas automatizadas permiten la detección temprana de errores, mejoran la confiabilidad del software y fortalecen la continuidad operativa (Ambur and Devaraj, 2018; Shahin et al., 2017).

A nivel operativo, la disposición favorable del proceso de Tecnologías de la Información y de los líderes de la EPSI facilitó la ejecución del proyecto, lo que aseguró un desarrollo ágil y garantizó su viabilidad técnica, sostenibilidad y escalabilidad.

De esta manera, el proyecto respondió a una necesidad institucional prioritaria y se enmarcó en las estrategias de mejora tecnológica y operativa de la EPSI AIC. Su ejecución permitió fortalecer la capacidad y la calidad de los sistemas de información en salud, en beneficio directo de las comunidades atendidas, y aseguró la sostenibilidad y la escalabilidad requeridas en un contexto de alta demanda y complejidad.

1.5. Metodología de la Investigación

Para dar cumplimiento a los objetivos establecidos en el proyecto, la metodología de este proyecto de grado tomó como referencia la metodología *Design Science Research* (DSR) propuesta por Hevner, March, Park y Ram (Hevner et al., 2004), un paradigma de investigación que busca generar conocimiento mediante la construcción de artefactos que resuelven problemas prácticos y relevantes en el dominio de las Tecnologías de la Información. En este contexto, el artefacto principal desarrollado corresponde a un Marco de Pruebas Automatizadas para la EPSI AIC.

La elección de DSR sobre otras metodologías de investigación aplicada, como *Action Research* o *Case Study*, se fundamenta en que estas últimas se orientan principalmente a la comprensión o intervención de fenómenos organizacionales, mientras que DSR tiene como propósito central la creación y evaluación de soluciones tecnológicas. Dado que el objetivo de este proyecto no se limitó al análisis del problema, sino que incluye el diseño, implementación y validación de un artefacto, DSR resulta más adecuada para abordar el problema planteado.

El ciclo DSR adoptado operó sobre tres ejes fundamentales: Investigación del Problema, Diseño del Tratamiento y Validación del Tratamiento. Las cuatro fases del proyecto se distribuyeron sobre estos tres ejes de la siguiente manera: la Fase A (Diagnóstico y Diseño del Proceso de *QA*) corresponde al eje de Investigación del Problema; las Fases B y C (Priorización y Diseño de Casos de Prueba, e Implementación de la Automatización) corresponden conjuntamente al eje de Diseño del Tratamiento, dado que el diseño de casos de prueba y la implementación del marco de automatización constituyen dos etapas secuenciales e interdependientes del mismo proceso de construcción del artefacto; y la Fase D (Evaluación de Impacto) corresponde al eje de Validación del Tratamiento.

Eje DSR	Alcance / Enfoque	Fase del proyecto
Investigación del Problema	Comprensión del contexto, diagnóstico y definición del problema	Fase A
Diseño del Tratamiento	Diseño del artefacto (proceso *QA*, casos de prueba, marco de automatización)	Fases B y C
Validación del Tratamiento	Evaluación del impacto del artefacto en el contexto real	Fase D

Tabla 1.1: Correspondencia entre los ejes del ciclo DSR y las fases del proyecto. Fuente: Elaboración propia con base en (Hevner et al., 2004)

Fases del Proyecto

Fase A: Diagnóstico y Diseño del Proceso de Aseguramiento de Calidad

Esta fase comprendió el diagnóstico del estado actual de los procesos de pruebas en la EPSI AIC y diseñar un proceso formal de aseguramiento de calidad adaptado a su contexto organizacional y tecnológico.

Obj. Específico	Actividades Clave
O.E. 1	**A.1. Diagnóstico del Estado Actual:** Caracterizar prácticas actuales de pruebas mediante entrevistas y análisis documental, mapear roles y flujos de trabajo, y establecer línea base cuantitativa (fuga de defectos, tiempos de ejecución, métricas disponibles).
	A.2. Diseño del Proceso de QA (TO-BE): Definir proceso formal de aseguramiento de calidad incluyendo políticas, flujos de trabajo, puntos de control e integración con Scrum.
	A.3. Definición de Estructura Organizacional: Establecer roles, responsabilidades, matriz de responsabilidades RACI (Responsable, Aprobador, Consultado e Informado) y plan de transición al modelo propuesto.

Fase B: Priorización y Diseño de Casos de Prueba

Esta fase se enfoca en establecer criterios sistemáticos para seleccionar funcionalidades críticas y diseñar casos de prueba aplicando técnicas formales de caja negra.

Obj. Específico	Actividades Clave
O.E. 2	**B.1. Definición de Criterios de Priorización:** Establecer un modelo con tres criterios: (1) riesgo para la continuidad del servicio, (2) criticidad del proceso de negocio según la Resolución 866 de 2021, y (3) frecuencia de uso del módulo o funcionalidad.
	B.2. Aplicación de Criterios: Evaluar los módulos del sistema con la matriz de priorización, validar los resultados con los equipos clave y generar el listado priorizado de funcionalidades a probar.
O.E. 3	**B.3. Selección de Técnicas de Caja Negra:** Elegir la técnica según la funcionalidad: Particiones de Equivalencia para entradas, Tablas de Decisión para lógica compleja y Arreglos Ortogonales para combinaciones de parámetros.
	B.4. Diseño de Casos de Prueba: Diseñar casos detallados con precondiciones, pasos, datos y resultados esperados, asegurando trazabilidad con los requisitos y funcionalidades.
	B.5. Validación de Cobertura: Verificar que los casos cubren los escenarios críticos y validar con expertos del dominio.

Fase C: Implementación de la Automatización

Esta fase consiste en seleccionar herramientas, diseñar la arquitectura del marco de automatización e implementar los casos de prueba priorizados.

Obj. Específico	Actividades Clave
O.E. 4	**C.1. Selección de Herramientas y Stack Tecnológico:** Elegir herramientas de automatización considerando compatibilidad, costo, curva de aprendizaje, soporte y capacidad de integración, justificando la selección mediante una matriz de decisión.
	C.2. Diseño de Arquitectura del Marco: Implementar un diseño modular (por ejemplo, *Page Object Model*) que separe la lógica de negocio de la interfaz, incluyendo componentes para gestión de datos, utilidades, reportería y trazabilidad.
	C.3. Desarrollo del Sistema de Reportería: Codificar el módulo de reportería automatizada que genere evidencia de fallas (capturas, logs, información del ambiente) en formatos HTML y PDF, vinculada a los casos de prueba ejecutados.
	C.4. Implementación Piloto: Automatizar los casos de prueba de mayor riesgo, realizando pruebas de concepto para validar el funcionamiento y la estabilidad del marco.
	C.5. Documentación Técnica: Generar guías de instalación, configuración, ejecución y mantenimiento del marco automatizado.

Fase D: Evaluación de Impacto

Esta fase final evalúa el impacto del proceso de aseguramiento de calidad implementado mediante indicadores cuantitativos y cualitativos.

Obj. Específico	Actividades Clave
O.E. 5	**D.1. Medición de Indicadores de Proceso:** Evaluar la cobertura de pruebas y la productividad, considerando el porcentaje de funcionalidades críticas automatizadas y la reducción en los tiempos de ejecución.
	D.2. Medición de Indicadores de Producto: Medir la correctitud, como la proporción de casos de prueba ejecutados de forma exitosa sobre el total ejecutado, y la confiabilidad, como el complemento de la proporción de no conformidades detectadas sobre el total de casos de prueba ejecutados, conforme a las definiciones del ISTQB.
	D.3. Evaluación de Continuidad Operativa: Analizar el impacto en la continuidad del servicio, incluyendo la reducción de incidentes, tiempos de respuesta y el cumplimiento de los *Service Level Agreements* (SLAs).
	D.4. Análisis Cualitativo: Recolectar la percepción del equipo de desarrollo y QA sobre la confianza en los despliegues, la facilidad de diagnóstico de errores y la adopción del proceso de aseguramiento de calidad.
	D.5. Análisis Comparativo y Recomendaciones: Comparar las métricas antes y después de la implementación, identificar logros, limitaciones y proponer recomendaciones para el escalamiento y mejora continua del proceso.

1.6. Resultados obtenidos

El proyecto produjo cuatro resultados principales: un proceso formal de aseguramiento de calidad *TO-BE* adaptado al contexto de la EPSI AIC, con estructura de roles, políticas de calidad y mecanismo de *Quality Gate*; un conjunto de casos de prueba con trazabilidad completa diseñados mediante técnicas de caja negra ISTQB sobre el Módulo de Referencia 24/7, sistema misional que gestiona el traslado oportuno (remisión) de pacientes entre niveles de atención; un marco de automatización funcional con cuatro capas de verificación (Pytest, Newman, Vitest y Playwright) integradas en un *pipeline* CI/CD sobre infraestructura local; y evidencia cuantitativa del impacto, con cobertura automatizada del 41,1 % en *backend* y 21,9 % en *frontend*, reducción del 76,5 % en el tiempo de validación de regresión del *backend*, ahorro acumulado de 33,3 horas de esfuerzo manual durante el piloto y satisfacción general del equipo de 4,5 sobre 5.

Capítulo 2

Marco de referencia

En este capítulo se presentan las bases teóricas y los trabajos previos que sirven como base para la realización de este proyecto.

2.1. Marco Teórico

El presente marco teórico articula los fundamentos normativos, metodológicos y técnicos necesarios para la implementación de pruebas automatizadas en el sector salud. Se estructura en seis dimensiones complementarias que sustentan la propuesta: marcos normativos aplicables, fundamentos de calidad de software, metodologías de desarrollo, automatización de pruebas, herramientas y marcos de trabajo, y criterios de evaluación de impacto.

2.1.1. Bases Teóricas

En esta sección se describen los fundamentos teóricos que sustentan el trabajo de investigación.

2.1.1.1. Marcos normativos en Salud

La Resolución 866 de 2021, expedida por el Ministerio de Salud y Protección Social de Colombia, establece los lineamientos para la gestión de la seguridad de la información en el sector salud. Su objetivo es garantizar que las entidades que procesan datos relacionados con la atención en salud adopten medidas integrales de protección que aseguren la confidencialidad, integridad, disponibilidad y trazabilidad de la información (Ministerio de Salud y Protección Social, 2021).

La norma obliga a las instituciones a implementar políticas, procesos y controles que permitan minimizar riesgos asociados con incidentes de seguridad como accesos indebidos, pérdida, alteración o divulgación no autorizada de los datos. En este sentido, se articula con los principios de protección de datos personales y con la normatividad vigente en el país en materia de *habeas data*, pero adaptada al contexto específico del sector salud, donde la información clínica es considerada especialmente sensible.

Asimismo, la resolución enfatiza en la necesidad de alinear la gestión de seguridad de la información con estándares y buenas prácticas internacionales de seguridad informática y ciberseguridad. Esto implica que las instituciones deben diseñar planes de gestión del riesgo, mecanismos de monitoreo, auditorías internas y procesos de mejora continua orientados a salvaguardar los sistemas de información en salud (Ministerio de Salud y Protección Social, 2021).

En el caso de instituciones de salud, la implementación de pruebas automatizadas en el ciclo de vida del software es clave para cumplir con estos lineamientos, ya que permite verificar de forma

constante la correcta gestión de permisos, el cifrado de datos y la resistencia de los sistemas frente a vulnerabilidades.

2.1.1.2. Fundamentos de Calidad de Software y Aseguramiento de la Calidad (QA)

La calidad del software es un concepto fundamental que va más allá de su simple funcionalidad. Se define por un conjunto de atributos que aseguran que el producto final cumple con las necesidades del usuario y los estándares técnicos (Pressman and Maxim, 2020). La norma internacional ISO/IEC 25010:2011 establece un modelo de calidad riguroso que sirve de referencia, centrándose en características como la fiabilidad, seguridad, usabilidad y mantenibilidad (ISO, 2011). Las prácticas de pruebas automatizadas contribuyen a mejorar directamente estos atributos, garantizando que el software funcione de manera consistente, proteja la información sensible y sea fácil de adaptar a futuros cambios.

Es crucial diferenciar entre Aseguramiento de la Calidad (QA) y Control de Calidad (QC). Mientras que el QC es un proceso reactivo que busca encontrar defectos en el producto terminado, el QA es una estrategia proactiva orientada a prevenir errores desde las etapas iniciales del desarrollo (NIST, 2017). La automatización de pruebas se enmarca en las prácticas de QA, ya que su objetivo es fortalecer los procesos de desarrollo para evitar fallas, en lugar de simplemente corregirlas una vez que han ocurrido.

2.1.1.3. Ciclo de Vida del Desarrollo de Software (SDLC)

El desarrollo de software se guía por un Ciclo de Vida del Desarrollo de Software (SDLC). Una de las metodologías más utilizadas actualmente es Scrum, un marco de trabajo ágil que se caracteriza por su enfoque iterativo e incremental, donde el desarrollo se organiza en periodos cortos llamados *Sprints*. A diferencia de modelos secuenciales como el de cascada, Scrum prioriza la entrega de valor continua y la adaptación al cambio (Schwaber and Sutherland, 2020).

En este contexto, las pruebas no son una fase aislada al final del proceso, sino una actividad integrada y constante. La detección temprana de errores se convierte en una responsabilidad compartida que ocurre dentro de cada Sprint. La curva de costos de los defectos demuestra que corregir un error en las primeras etapas del desarrollo es significativamente menos costoso que hacerlo en producción (Boehm, 1988). Por lo tanto, la integración de pruebas en cada iteración es fundamental para garantizar que el software sea estable y confiable, mitigando los riesgos para los usuarios y los procesos de negocio.

2.1.1.4. Conceptos Clave de Pruebas de Software

Las pruebas de software constituyen el conjunto de actividades mediante las cuales se verifica que un sistema se comporta conforme a lo esperado y se detectan defectos antes de que lleguen a producción. En la práctica se distinguen dos modalidades de ejecución: la manual, en la que una persona ejecuta y evalúa los casos, y la automatizada, en la que scripts y herramientas los ejecutan de forma repetible. Cada una responde a necesidades distintas: la manual conserva su valor en

escenarios exploratorios o que exigen juicio humano, mientras que la automatizada resulta ventajosa cuando las pruebas son repetitivas y deben ejecutarse con frecuencia (Sommerville, 2016). Para el presente proyecto esta distinción es relevante, ya que el estado inicial de la EPSI AIC dependía por completo de la modalidad manual, con las limitaciones de repetibilidad y trazabilidad que ello implica.

Las pruebas también se organizan por niveles según el alcance de lo que verifican. Las pruebas unitarias validan componentes individuales de forma aislada; las de integración comprueban la correcta interacción entre módulos; las de regresión confirman que los cambios introducidos no afecten funcionalidades ya existentes; y las de aceptación evalúan el sistema frente a las expectativas del usuario o del negocio (Pressman and Maxim, 2020). De estos niveles, la regresión es el que mejor evidencia el beneficio de la automatización: en un entorno donde el código cambia de forma continua, revalidar manualmente todo lo previamente construido resulta costoso y propenso a error, lo que justifica delegar esa tarea a un conjunto de pruebas automatizado (Myers et al., 2004). Esta lógica es la que orienta el diseño del marco propuesto, que prioriza precisamente la automatización de la regresión sobre el módulo crítico de la organización.

2.1.1.5. Automatización de pruebas de software

La automatización de pruebas consiste en apoyar la ejecución de los casos en herramientas que los corren de manera desatendida, con el fin de ampliar la cobertura, acortar los tiempos de verificación y reducir la dependencia de la intervención humana (Fewster and Graham, 2019). No obstante, automatizar no es un objetivo en sí mismo: la literatura advierte que su conveniencia depende del caso y que resulta especialmente justificada cuando las pruebas son repetitivas, de alto volumen o críticas para la continuidad del servicio (Kaner et al., 1999). Este criterio de selectividad es el que se adopta en este trabajo, donde la automatización se concentra en las funcionalidades de mayor riesgo en lugar de aplicarse de manera indiscriminada.

La automatización rara vez se implementa de forma aislada; suele articularse con prácticas de desarrollo que la incorporan al ciclo de vida del software. Entre las más difundidas están el desarrollo guiado por pruebas (TDD), que antepone la escritura de pruebas unitarias a la codificación para que el diseño responda a los requisitos (Beck, 2003); el desarrollo guiado por comportamiento (BDD), que describe el comportamiento esperado en un lenguaje cercano al natural para acercar a desarrolladores y usuarios (North, 2006); y el desarrollo guiado por pruebas de aceptación (ATDD), que deriva las pruebas de los criterios de aceptación acordados con el cliente (Ambler, 2002). Más que técnicas de prueba, estas prácticas comparten un principio que el marco propuesto retoma: desplazar la verificación hacia etapas tempranas del desarrollo en lugar de relegarla al final del ciclo.

2.1.1.6. Buenas Prácticas de Pruebas según el ISTQB

El *International Software Testing Qualifications Board* (ISTQB) es un referente internacional en la estandarización de las prácticas de prueba, y su cuerpo de conocimiento ofrece un vocabulario común y un conjunto de técnicas sistemáticas para diseñar, ejecutar y analizar pruebas (Hamburg

and Roman, 2025). Para este proyecto, su principal aporte no es el esquema de certificación sino su catálogo de técnicas de *caja negra*, es decir, aquellas que verifican el *software* desde la perspectiva de sus entradas, salidas y requisitos, sin atender a la estructura interna del código. Esta orientación resulta pertinente en sistemas de salud, donde interesa sobre todo garantizar que las funciones críticas para la atención se comporten de forma correcta y trazable.

El ISTQB agrupa las técnicas de *caja negra* según el tipo de lógica que verifican: las basadas en datos comprueban el tratamiento de dominios de entrada; las basadas en comportamiento examinan los estados y flujos del sistema; y las basadas en reglas validan la correcta aplicación de la lógica de negocio. Esta clasificación fue determinante para el diseño de casos del presente trabajo, ya que permitió elegir la técnica en función de la naturaleza de cada funcionalidad: particiones de equivalencia para las validaciones de formato y rango, y tablas de decisión para los flujos de autorización con múltiples condiciones.

Más allá del catálogo, el enfoque del ISTQB se caracteriza por su orientación al riesgo y a la eficiencia: el diseño de pruebas no busca cubrirlo todo, sino seleccionar las técnicas que ofrezcan mayor detección de defectos con un esfuerzo razonable. Este principio conecta con la estrategia de priorización adoptada en este proyecto y sustenta la decisión de concentrar el diseño formal de casos en el módulo de mayor criticidad, en lugar de dispersar el esfuerzo de manera uniforme.

2.1.1.7. Criterios de Priorización Basados en Riesgo

La definición de criterios para priorizar los aplicativos y funcionalidades sobre los cuales se implementará un modelo de calidad de pruebas no es una decisión arbitraria, sino que se fundamenta en el enfoque de pruebas basadas en riesgo (*risk-based testing*) establecido por el ISTQB en su Syllabus Foundation Level v4.0 (International Software Testing Qualifications Board, 2023).

Dicho enfoque define que el nivel de riesgo de un componente de software se determina a partir de dos dimensiones: la probabilidad de ocurrencia de una falla (*risk likelihood*) y el impacto que dicha falla tendría sobre el negocio (*risk impact*). Este modelo bidimensional, alineado con los principios de gestión de riesgos de la norma ISO 31000 (International Organization for Standardization, 2018), proporciona el sustento teórico para seleccionar criterios de priorización que cubran ambas dimensiones de manera sistemática.

Desde esta perspectiva, los criterios de priorización deben operacionalizar las dos dimensiones del modelo. El impacto operativo se refleja en el riesgo de continuidad del servicio: una falla en un sistema misional compromete directamente la prestación del servicio que soporta, configurando un escenario de alto impacto en el negocio.

El impacto legal y regulatorio, por su parte, representa las consecuencias derivadas del incumplimiento de obligaciones normativas concretas y verificables; en este sentido, Knight y Wika (Knight and Wika, 1995) señalan que los sistemas médicos son intrínsecamente críticos para la seguridad, lo que exige enfoques de prueba más rigurosos que consideren explícitamente las consecuencias normativas de una falla.

La frecuencia de uso actúa como proxy de la probabilidad de exposición al defecto: a mayor volumen de uso de un módulo o funcionalidad, mayor es la probabilidad de que un defecto latente

se manifieste en producción.

La desagregación del eje de impacto en dos criterios diferenciados, operativo y normativo, responde a contextos organizacionales donde las consecuencias de una falla trascienden lo técnico y pueden derivar en sanciones regulatorias o afectación directa a los usuarios del servicio. Esta distinción, aunque no explícita en el modelo genérico del ISTQB, es coherente con su principio de adaptar el análisis de riesgos al contexto organizacional y sectorial en el que se aplica.

Finalmente, la ponderación específica asignada a cada criterio no se establece de manera apriorística, sino que debe emerger del diagnóstico organizacional previo. Siguiendo a Yin (Yin, 2018), en contextos organizacionales donde los procesos no están formalizados y el conocimiento reside principalmente en las personas, la evidencia cualitativa obtenida mediante entrevistas a actores clave constituye una fuente válida y metodológicamente sólida para informar decisiones de diseño.

2.1.1.8. Herramientas de Pruebas Automatizadas

El ecosistema de herramientas de automatización es amplio y suele especializarse por nivel de prueba. Para las pruebas de interfaz gráfica se emplean herramientas como Selenium, Cypress o Playwright; para las pruebas unitarias, marcos propios de cada lenguaje como JUnit, TestNG, Pytest o Vitest; para las pruebas de rendimiento, soluciones como Apache JMeter o Gatling; y para la validación de servicios web y APIs, herramientas como Postman y su ejecutor por línea de comandos Newman (Meszaros, 2007). Más que la herramienta en sí, lo determinante es su adecuación al contexto: la literatura señala como criterios de selección la compatibilidad con la infraestructura existente, el costo, la curva de aprendizaje, el soporte de la comunidad y la capacidad de integrarse a un *pipeline* de integración y entrega continua (Garousi et al., 2019). Estos criterios son los que guiaron la selección de herramientas del presente proyecto, que priorizó opciones de código abierto ya presentes o afines al ecosistema tecnológico de la EPSI AIC para reducir la curva de adopción.

2.1.1.9. Marcos de Pruebas Automatizadas

Un marco de pruebas (*test framework*) es la arquitectura que organiza y estandariza la automatización, de modo que los casos no se escriban de forma aislada sino sobre una base común que facilite la reutilización de componentes, la gestión de datos de prueba y la generación de reportes (Meszaros, 2007). Un marco bien diseñado se caracteriza por ser modular, mantenible y escalable, de manera que absorba los cambios del sistema sin obligar a reescribir masivamente los casos existentes (Dustin et al., 1999). Esta cualidad es la que distingue una práctica sostenible de un esfuerzo puntual: un marco consolidado trasciende los proyectos individuales y se incorpora a la estrategia de aseguramiento de calidad de la organización. Bajo esta premisa se concibió el marco propuesto en este trabajo, orientado no a automatizar un módulo puntual sino a establecer una base reutilizable para el resto de los sistemas de la EPSI AIC.

2.1.1.10. Automatización de Pruebas en Organizaciones de Salud

La automatización de pruebas en el sector salud presenta condicionantes propios que inciden en el diseño de cualquier marco de trabajo. Los sistemas médicos son críticos para la seguridad, lo que exige enfoques de verificación más rigurosos que los de un sistema convencional, incluida la validación de propiedades específicas de seguridad (Knight and Wika, 1995). A ello se suman requisitos particulares como la verificación del cumplimiento normativo (por ejemplo, la trazabilidad exigida por la Resolución 866 de 2021 (Ministerio de Salud y Protección Social, 2021)), la integridad de los datos clínicos y la disponibilidad del sistema durante operaciones sensibles. La literatura reporta que las organizaciones de salud que adoptan pruebas automatizadas mejoran la confiabilidad de sus sistemas y reducen los incidentes que afectan la atención de los pacientes (Healthcare Information and Management Systems Society, 2019). No obstante, el éxito de estas iniciativas depende de factores organizacionales tanto como técnicos: la capacitación del equipo, la definición de métricas propias del sector y la capacidad de sostener la automatización sin comprometer la respuesta ante emergencias o cambios regulatorios. Estas particularidades son las que justifican que el marco propuesto se haya diseñado a la medida del contexto de la EPSI AIC y no como una adaptación genérica.

2.1.1.11. Diseño organizacional aplicado a equipos de aseguramiento de calidad

El diseño organizacional es la disciplina que estudia cómo distribuir, coordinar y supervisar las tareas dentro de una organización para alcanzar sus objetivos de manera eficiente y sostenible (Mintzberg, 1979). En el contexto del desarrollo de software, la forma en que se estructura el equipo responsable del aseguramiento de calidad determina directamente la sostenibilidad del proceso, la distribución del conocimiento institucional y la capacidad de respuesta ante defectos. Una decisión organizacional inadecuada puede comprometer la efectividad de incluso el marco técnico más robusto, razón por la cual la literatura especializada reconoce el diseño organizacional como una dimensión crítica de cualquier iniciativa de *QA* (Dustin et al., 1999).

Mintzberg (Mintzberg, 1979) identificó que en equipos pequeños con tecnología compleja, las estructuras altamente centralizadas generan dependencias críticas en actores individuales, mientras que las estructuras distribuidas favorecen la transferencia de conocimiento y la resiliencia operativa ante la rotación de personal. En el ámbito específico de los equipos de software, Leite et al. (Leite et al., 2021) realizaron un estudio de teoría fundamentada (*grounded theory*) sobre la organización de células de desarrollo en organizaciones que buscan integrar prácticas de entrega continua (*continuous delivery*). Sus hallazgos determinaron que las estructuras más compatibles con la integración sostenible de *QA* son aquellas donde la responsabilidad de las pruebas se distribuye entre los miembros del equipo de desarrollo, en lugar de concentrarse en un departamento o rol independiente. Esta configuración, que los autores denominan equipo multifuncional, favorece la adopción cultural de las prácticas de calidad porque las pruebas dejan de ser una actividad externa al desarrollo y se convierten en parte integral del trabajo cotidiano de cada desarrollador.

Por su parte, Dustin et al. (Dustin et al., 1999) complementan esta perspectiva desde el ángulo de la adopción práctica, señalando que los marcos de automatización de pruebas más sostenibles

en organizaciones con baja madurez inicial son aquellos que no dependen de la contratación de roles exclusivos de *QA*, sino que aprovechan las capacidades técnicas existentes en el equipo de desarrollo mediante la designación de responsables de coordinación con dedicación parcial. Este enfoque reduce la barra de entrada para organizaciones que inician sus procesos de aseguramiento de calidad y minimiza el riesgo de abandono del proceso ante restricciones presupuestales o cambios de personal.

La literatura especializada coincide en que la elección del modelo organizacional para *QA* no es una decisión únicamente técnica, sino que debe responder a las condiciones reales de la organización: su tamaño, su nivel de madurez en procesos de *testing*, su capacidad presupuestal y su cultura de desarrollo. Un modelo organizacional que funciona en una empresa con alta madurez y recursos dedicados puede resultar inviable e insostenible en una organización que apenas inicia la formalización de sus prácticas de aseguramiento de calidad (Dustin et al., 1999).

2.1.1.12. Evaluación del Impacto de la Automatización de Pruebas

La implementación de pruebas automatizadas debe evaluarse con base en indicadores de calidad y eficiencia que permitan cuantificar su valor organizacional. La literatura especializada identifica múltiples dimensiones de evaluación que deben considerarse de manera integral.

Métricas Cuantitativas Entre las métricas cuantitativas más relevantes se encuentran la cobertura de pruebas, la reducción en tiempos de ejecución, la detección temprana de defectos y la disminución de costos asociados a fallas en producción (Fewster and Graham, 2019; Garousi et al., 2019). Específicamente, las organizaciones suelen medir:

- **Tasa de fuga de defectos:** Proporción de errores que llegan a producción después de implementar automatización.
- **Tiempo medio de detección:** Reducción en el tiempo transcurrido entre la introducción de un defecto y su identificación.
- **Cobertura de pruebas automatizadas:** Porcentaje del código y funcionalidades cubiertas por pruebas automatizadas.
- **Tiempo de ciclo de desarrollo:** Impacto en la velocidad de entrega de nuevas funcionalidades.

Métricas Cualitativas y Organizacionales La evaluación debe incluir aspectos cualitativos relacionados con la adopción y sostenibilidad de la automatización. Según Dustin et al. Dustin et al. (1999), los factores críticos incluyen:

- **Confianza del equipo de desarrollo:** Medida a través de encuestas de percepción sobre la seguridad en los despliegues.

- **Adopción organizacional:** Evaluación del grado de integración de las prácticas automatizadas en los procesos rutinarios.
- **Sostenibilidad técnica:** Capacidad del marco implementado para evolucionar con los cambios tecnológicos y organizacionales.

Metodologías de Evaluación Para organizaciones que implementan automatización y que carecen de procesos formalizados de automatización de pruebas, la literatura recomienda diseños cuasi-experimentales con medición pre/post implementación Cook and Campbell (1979). Este enfoque permite documentar el impacto incluso cuando no existen líneas base históricas, utilizando el mismo equipo como grupo de control y experimental en diferentes momentos temporales. La triangulación de fuentes (métricas objetivas, encuestas de percepción y observación directa) proporciona validez adicional a los hallazgos y permite identificar factores contextuales que influyen en el éxito de la implementación (Yin, 2018).

Síntesis del Marco Teórico El presente marco teórico articula los fundamentos normativos, metodológicos y técnicos necesarios para la implementación de pruebas automatizadas en el sector salud. Se estructura en cinco dimensiones complementarias que sustentan la propuesta: marcos normativos aplicables, fundamentos de calidad de software, metodologías de desarrollo, herramientas de automatización y criterios de evaluación de impacto.

2.2. Estado del Arte

El acelerado ritmo de la transformación digital exige que las organizaciones garanticen la calidad y confiabilidad de sus productos de software. En el sector de la salud, esta premisa se intensifica, ya que los fallos pueden trascender las pérdidas económicas y generar riesgos directos para la seguridad de los usuarios del sistema de seguridad social del país. La implementación de un marco de pruebas automatizadas no solo busca la eficiencia, sino que es una acción estratégica de mitigación de riesgos y corrección de la inmadurez del proceso de desarrollo. En el contexto de la EPSI AIC, la ausencia de dicho marco se traduce en efectos negativos concretos (defectos tardíos, sobrecostos, falta de fiabilidad, etc.) que deben ser impedidos de manera proactiva.

El presente Estado del Arte articula la investigación en cuatro áreas clave que sustentan la viabilidad y necesidad del proyecto: la evolución de las metodologías de testing y adopción tecnológica, las limitaciones de los enfoques tradicionales, la implementación de marcos de automatización en organizaciones de salud, y las soluciones técnicas para garantizar trazabilidad y control de defectos.

2.2.0.1. Evolución de las Metodologías de Testing y Adopción de IA

La evolución del testing de software ha experimentado una transformación significativa en las últimas décadas, transitando desde métodos manuales hacia enfoques automatizados impulsados por inteligencia artificial. (Saxena, 2025) documenta que el avance más significativo en testing durante la última década ha sido “la adopción generalizada de testing automatizado impulsado por

IA y ML, que ha aumentado significativamente la eficiencia, cobertura de pruebas y la capacidad de identificar rápidamente problemas durante el proceso de desarrollo". Esta evolución se caracteriza por la integración del testing en pipelines de DevOps, habilitando testing continuo y ciclos de *release* más rápidos.

La transformación metodológica se estructura en tres etapas principales: la era de herramientas robustas de automatización y software de código abierto (2000–2010), el periodo de metodologías ágiles y herramientas escalables (2010–2018), y la época actual de testing autónomo con *machine learning* e IA (2019 en adelante). Esta progresión ha sido impulsada por la necesidad de abordar las limitaciones inherentes a los métodos tradicionales.

2.2.0.2. Limitaciones de los Métodos Tradicionales

(Baqar and Khanda, 2024) identifican desafíos críticos en la generación y validación tradicional de casos de prueba: "los métodos tradicionales enfrentan varios desafíos críticos, siendo uno de los más apremiantes la naturaleza intensiva en tiempo de la creación manual de casos de prueba, especialmente para sistemas grandes y complejos". Esta dependencia en entrada humana introduce riesgo de error humano, donde los testers pueden pasar por alto escenarios críticos o fallar en contemplar casos límite, llevando a brechas en el proceso de testing.

(Zamli et al., 2007) ya documentaban estas problemáticas en el contexto de herramientas de testing automatizado, señalando que "las prácticas actuales de testing de software carecen de automatización y están aún principalmente basadas en procesos altamente manuales desde la generación de casos de prueba hasta la ejecución actual del test". Esta perspectiva histórica demuestra la persistencia de estos desafíos a lo largo del tiempo.

Las limitaciones de los enfoques manuales se agravan en organizaciones donde no existen procesos formalizados de testing. En estos contextos, la ausencia de documentación sistemática y métricas de calidad genera dependencias críticas en conocimiento individual y aumenta exponencialmente el riesgo de fallas en producción (Fewster and Graham, 2019).

2.2.0.3. Implementación de Marcos de Automatización en Organizaciones de Salud

En el ámbito de la salud, los riesgos derivados de la inmadurez metodológica son especialmente graves. El marco de pruebas debe ser, ante todo, un mecanismo de mitigación de riesgos enfocado en la verificación de la seguridad. La investigación de Knight and Wika (1995) destaca que los sistemas médicos son intrínsecamente críticos para la seguridad (*safety-critical*). Para mitigar el riesgo de fallas, la solución se desplaza de la prueba funcional a la verificación de propiedades de seguridad específicas. Esta necesidad ha impulsado metodologías avanzadas que requieren automatización sofisticada:

- La prueba automatizada con casos seleccionados es vital para demostrar que el software exhibe propiedades de fiabilidad (Knight and Wika, 1995).
- Se utilizan técnicas de *reversal checking* para automatizar la determinación de la corrección de los resultados, un problema notoriamente difícil en software complejo Knight and Wika

(1995).

(Saxena, 2025) complementa esta perspectiva enfatizando que "los métodos de seguridad de software tradicionales, que se enfocan principalmente en testing de funcionalidad básica, pueden ya no ser suficientes en el paisaje complejo de hoy". Para sistemas críticos de salud, esto implica la necesidad de enfoques más rigurosos que incluyan testing de vulnerabilidades de seguridad específicas.

2.2.0.4. Casos de Éxito y Lecciones Aprendidas

La literatura documenta casos de implementación exitosa de automatización en el sector salud. El proyecto MIDAS, una iniciativa de automatización de pruebas de servicios en el sector salud reportada en la literatura (Hillah et al., 2016) implementó automatización de pruebas de servicios específicamente para abordar la sobrecarga de casos de prueba y la complejidad de los datos clínicos. Este proyecto demostró que la mitigación de riesgos en salud requiere *frameworks* de alta complejidad técnica y metodológica, capaces de manejar la sensibilidad y criticidad de los datos médicos (Hillah et al., 2016).

AutoBOT es una herramienta desarrollada por el Sidia Institute of Science and Technology para automatizar pruebas funcionales de software en dispositivos Android. Su estudio de caso (Viana et al., 2024) proporciona evidencia cuantitativa del impacto de la automatización, demostrando un ahorro cuantificable de 2074 horas de esfuerzo manual en un año. Este tipo de hallazgos ilustra cómo la automatización puede mitigar sobrecostos y cuellos de botella del proceso manual, elementos particularmente críticos en organizaciones de salud como EPSI AIC, donde los recursos técnicos suelen ser limitados.

2.2.0.5. Factores Críticos de Éxito en Implementación

La literatura especializada identifica factores específicos que determinan el éxito de implementaciones de automatización en organizaciones con procesos poco maduros:

- **Adopción gradual y participativa:** Las implementaciones exitosas requieren la participación activa del equipo de desarrollo desde la fase de diseño, asegurando que el marco de automatización se adapte a las capacidades y limitaciones organizacionales existentes (Dustin et al., 1999).
- **Enfoque en sostenibilidad:** Los marcos de pruebas deben diseñarse considerando la capacidad de mantenimiento a largo plazo, especialmente en organizaciones con recursos técnicos limitados. Esto incluye la selección de herramientas con curvas de aprendizaje manejables y comunidades de soporte activas (Garousi et al., 2019).
- **Métricas específicas del contexto:** La evaluación del impacto debe incluir métricas relevantes para el sector salud, como tiempo de detección de errores críticos, disponibilidad del sistema durante operaciones médicas y cumplimiento de requisitos regulatorios específicos (**?**).

2.2.0.6. Soluciones de Trazabilidad y Control de Defectos

La implementación efectiva de marcos de automatización requiere soluciones técnicas que garanticen la trazabilidad completa y el control sistemático de defectos. Esta dimensión es particularmente crítica en organizaciones que carecen de procesos formalizados de gestión de incidentes.

(Khankhoje, 2022) propone una arquitectura de solución donde el *framework* automatizado funciona como un sistema de alerta temprana que integra múltiples herramientas para garantizar trazabilidad completa. En este modelo, el sistema:

- Emplea APIs de herramientas de gestión de proyectos (como Jira) para la creación automática de *issues* al detectar fallas en las pruebas, eliminando el registro manual y sus retrasos asociados.
- Vincula automáticamente las fallas detectadas con herramientas de gestión de pruebas (como TestRail) para actualizar el estado de la prueba y mantener la trazabilidad desde el caso de prueba hasta la resolución del defecto.

Esta aproximación resulta especialmente relevante para organizaciones como las EPSI, donde la ausencia de sistemas formales de trazabilidad puede generar pérdida de información crítica sobre defectos y su impacto en la operación.

2.2.0.7. Integración con Procesos Existentes

La literatura enfatiza que la implementación de soluciones de trazabilidad debe considerar los procesos organizacionales existentes, incluso cuando estos son informales. (Meszaros, 2007) señala que los marcos de pruebas exitosos son aquellos que se integran naturalmente con los flujos de trabajo actuales del equipo de desarrollo, minimizando la resistencia al cambio y maximizando la adopción.

En el contexto de organizaciones de salud, esto significa que las soluciones de automatización deben diseñarse para funcionar con herramientas básicas (como hojas de cálculo o sistemas de *tickets* simples) mientras proporcionan una ruta de evolución hacia herramientas más sofisticadas conforme la organización desarrolla mayor madurez en sus procesos.

La revisión de la literatura evidencia el consenso académico sobre la evolución necesaria desde métodos manuales hacia marcos automatizados de testing, especialmente en sistemas críticos como los del sector salud. La investigación documenta casos exitosos de implementación y proporciona fundamentos técnicos sólidos para el diseño de soluciones de automatización.

La presente investigación contribuye al conocimiento existente mediante la documentación detallada del proceso de implementación de un marco de pruebas automatizadas en una organización del sector salud que carece de procesos formalizados previos, desarrollando metodologías de evaluación apropiadas para este contexto y analizando los factores que influyen en la adopción y sostenibilidad de la solución implementada.

2.3. Resumen del capítulo

El presente capítulo articuló las bases y fundamentos normativos, metodológicos y teóricos, además de los trabajos previos necesarios para la implementación de pruebas automatizadas en el sector salud y el correcto desarrollo del proyecto. Como primera medida se abordaron cinco dimensiones teóricas que sustentan la propuesta: marcos normativos aplicables, fundamentos de calidad de software, metodologías de desarrollo, herramientas de automatización y criterios de evaluación de impacto.

Posteriormente, se analizaron distintos trabajos relacionados con el proyecto, lo cual permitió identificar distintas experiencias y técnicas desarrolladas que aportan de manera significativa el correcto desarrollo de las fases del proyecto. Todo esto proporciona una amplia mirada teórica que fundamente en el desarrollo de la solución implementada.

Capítulo 3

Diseño e Implementación del Marco de Aseguramiento de Calidad

Este capítulo documenta el desarrollo del artefacto central del proyecto, cubriendo las tres primeras fases operativas del ciclo Design Science Research adoptado: la Fase A de Investigación del Problema y las Fases B y C correspondientes al eje de Diseño del Tratamiento. Las secciones 3.1 y 3.2 presentan la Fase A, que comprende el diagnóstico del estado actual de las prácticas de calidad en la EPSI AIC. Se describe el enfoque metodológico empleado a partir de entrevistas en profundidad, análisis documental y evaluación de madurez con el modelo Test Maturity Model Integration (TMMi), junto con los hallazgos del diagnóstico y la línea base referencial que sirvió de punto de partida para el diseño de la solución. A partir de ese diagnóstico, las secciones 3.3 y 3.4 presentan el diseño del proceso formal de aseguramiento de calidad TO-BE: sus principios orientadores, el flujo de trabajo propuesto, las políticas y criterios del proceso, la estructura organizacional con matriz RACI y el plan de transición hacia el modelo propuesto. La sección 3.5 documenta la Fase B, que abarca los criterios de priorización de funcionalidades críticas, la aplicación de la matriz de riesgo que identificó el Módulo de Referencia 24/7 como candidato prioritario del piloto, la selección de técnicas de caja negra según el tipo de lógica a verificar y el diseño formal de los casos de prueba con trazabilidad completa. Finalmente, la sección 3.6 desarrolla la Fase C con la selección de herramientas del marco de automatización, la arquitectura de las cuatro capas de verificación implementadas con Pytest, Newman, Vitest y Playwright, la implementación piloto sobre el módulo priorizado y la documentación técnica generada como artefacto de transferencia institucional.

3.1. Marco metodológico del objetivo

3.1.1. Enfoque Metodológico

El desarrollo de este objetivo se enmarcó en un enfoque cualitativo y descriptivo dentro del paradigma Design Science Research (DSR), específicamente en la fase de Ïnvestigación del Problema"que caracteriza la primera etapa del ciclo de diseño. Este enfoque permitió comprender en profundidad el contexto organizacional, las prácticas actuales y las necesidades específicas de la EPSI AIC antes de proponer una solución de diseño.

La investigación cualitativa se justificó por la necesidad de capturar las perspectivas, experiencias y expectativas de múltiples actores organizacionales sobre la calidad del software y su impacto en los procesos misionales. Según (Yin, 2018), este tipo de enfoque es apropiado cuando se busca entender fenómenos complejos en su contexto natural, especialmente en organizaciones donde los procesos no están formalizados y el conocimiento reside principalmente en las personas.

Figura 3.1: Enfoque metodológico en la fase de investigación del problema bajo el paradigma Design Science Research (DSR). Elaboración propia

3.1.2. Estrategia de Investigación

Para avanzar en la estrategia de investigación, se definieron tres frentes complementarios que permitieron triangular la información recolectada y construir así un diagnóstico integral, en este sentido; los frentes fueron:

a. Entrevistas cualitativas en profundidad.

b. Análisis documental y de procesos.

c. Evaluación de madurez mediante TMMI.

Esta triangulación metodológica, recomendada por (Denzin and Lincoln, 2011), aumenta la validez de los hallazgos al contrastar información proveniente de diferentes fuentes y métodos.

3.1.2.1. Entrevistas Cualitativas en Profundidad

Las entrevistas fueron la fuente principal de información para comprender el estado actual desde múltiples perspectivas organizacionales; estas entrevistas en profundidad permitieron abordar aquellas experiencias y percepciones, es decir las entrevistas no solo permitieron capturar datos técnicos, sino que también permitieron entender cómo los diversos roles encontrados en la EPSI perciben la calidad del software y cual es su significado dentro del contexto laboral. En este sentido podemos definir entonces que las entrevistas fueron esenciales para:

- Capturar la perspectiva de actores con diferentes roles sobre la calidad del software.
- Identificar necesidades, expectativas y prioridades por nivel organizacional.
- Comprender el impacto de las fallas de sistemas en usuarios y procesos misionales.
- Descubrir conocimiento tácito no documentado sobre prácticas actuales.

Diseño del Instrumento Los instrumentos de recolección de la información fueron adaptados estratégicamente a los distintos niveles jerárquicos de la organización, manteniendo de esta forma una estructura flexible basada en preguntas abiertas que facilitaron la exploración de los conceptos y percepciones de los dinamizadores de la EPSI, cada entrevista estuvo estructurada por alrededor de 8 a 11 preguntas y se dividió en tres variantes las cuales fueron:

Variante	***Rol Organizacional***	***Enfoque Principal***
Nivel Estratégico	Representante Legal, Líder del Pilar Técnico	Visión estratégica, prioridades institucionales y compromiso organizacional frente al aseguramiento de la calidad
Nivel Táctico	Líder de Gestión TICs, Líder del Subproceso de Desarrollo	Procesos actuales de desarrollo, arquitectura del sistema, capacidades técnicas y desafíos operativos
Nivel Operativo	Desarrolladores, Líderes de procesos misionales	Experiencia práctica en el uso y desarrollo del sistema, impacto de fallas y necesidades operativas

Tabla 3.1: Variantes del instrumento de entrevista por nivel organizacional. Fuente: Elaboración propia.

Identificación y Mapeo de Actores De acuerdo al organigrama de la EPSI AIC y tomando en cuenta el principio de representatividad de roles se identificaron 7 actores clave para avanzar en las entrevistas:

Cabe precisar el alcance de los actores considerados en el diagnóstico. En la EPSI AIC, la función de seguridad de la información está orientada principalmente a la infraestructura tecnológica y no al subproceso de desarrollo de software, que constituyó el ámbito de este trabajo. Por esta razón, dicho rol no se incluyó entre los actores entrevistados de la Tabla 3.2, dado que sus responsabilidades no incidían directamente sobre las prácticas de pruebas ni sobre el ciclo de vida del desarrollo objeto del proyecto. No obstante, los requisitos de seguridad aplicables al software sí se incorporaron al

No	Rol	Nivel	Justificación
1	Representante Legal	Estratégico	Máxima autoridad, visión institucional
2	Líder del Pilar Técnico	Estratégico - Táctico	Coordinación de procesos técnicos
3	Líder de Gestión TICs	Táctico	Responsable directo de sistemas
4	Líder Subproceso de Desarrollo	Táctico - Operativo	Conocimiento detallado del proceso técnico
5	Desarrolladores	Operativo	Experiencia práctica del día a día
6	Líder Gestión Trámites Hospitalarios	Usuario - Negocio	Proceso altamente crítico (pacientes)
7	Líder Gestión Satisfacción del Comunero	Usuario - Negocio	Perspectiva del usuario final, proceso misional

Tabla 3.2: Actores clave identificados para entrevista. Fuente: Elaboración propia.

diagnóstico a través del marco normativo (Resolución 866 de 2021 (Ministerio de Salud y Protección Social, 2021) y normatividad de *Habeas Data*), analizado como fuente documental, lo que garantizó que las exigencias de confidencialidad, integridad y trazabilidad quedaran contempladas en el diseño del proceso.

Protocolo de entrevistas

Duración: 45-60 minutos por entrevista.

Modalidad: Virtual en las instalaciones de la EPSI AIC - Vía Meet.

Consentimiento informado: Autorizado por cada participante antes de iniciar.

Grabación: Audio con autorización explícita de los entrevistados.

Principios éticos: Confidencialidad, anonimización cuando fuera necesario, devolución de resultados.

Total de entrevistas realizadas: 7 de 7 planificadas inicialmente.

Tiempo total de grabación: ∼323 minutos (∼5.4 horas).

3.1.2.2. Análisis Documental

Durante la fase de diagnóstico se recopilo y analizo la siguiente documentación:

La ausencia de documentación formal de procesos de desarrollo y pruebas constituyó en sí misma un hallazgo significativo que confirmó la hipótesis inicial sobre la falta de procesos estructurados de aseguramiento de calidad.

No	Tipo de Documento	Descripción	Disponibilidad
1	Organigrama oficial	Estructura organizacional actualizada a agosto de 2025	Obtenido
2	Arquitectura tecnológica	Descripción de sistemas, servidores, infraestructura	Información parcial (no documentación formal)
3	Procesos de desarrollo	Procedimientos formales de SDLC	Información parcial (no documentación de plan de pruebas y procedimiento de paso a producción)
4	Casos de prueba	Documentación de pruebas	Inexistente
5	Métricas de calidad	Indicadores históricos	Inexistente
6	Registros de incidentes	Logs, tickets de producción	Inexistente
7	Normativas aplicables	Resolución 866 de 2021, otras	Obtenido

Tabla 3.3: Documentación recopilada durante la fase de diagnóstico. Fuente: Elaboración propia.

3.1.2.3. Evaluación de madurez del proceso de pruebas

Para complementar el diagnóstico cualitativo y el análisis documental, se aplicó una evaluación estructurada del nivel de madurez de los procesos de pruebas, tomando como referencia el artefacto de evaluación propuesto por Urbano Guevara y Valencia García (Urbano Guevara and Valencia García, 2023) en su trabajo de grado de la Pontificia Universidad Javeriana Cali, el cual adapta los criterios y áreas de proceso del modelo Test Maturity Model Integration (TMMi) a un instrumento de evaluación aplicable en organizaciones con baja madurez en procesos de testing. Dicho artefacto define 32 criterios de evaluación binarios (Sí/No) distribuidos en diez áreas de proceso correspondientes a los niveles 2 -Gestionado y 3 -Definido del modelo TMMi, con pesos ponderados basados en la encuesta de importancia realizada por la Fundación TMMi en 2020 a 202 organizaciones (Urbano Guevara and Valencia García, 2023). El detalle exhaustivo de esta metodología, incluyendo el esquema de codificación, las fórmulas de cálculo y la triangulación de la evidencia utilizada, se encuentra documentado en el Anexo D

Nivel de madurez global identificado Aplicando la metodología de cálculo detallada en el Anexo D y la escala de calificación ISO 33001:2015, la EPSI AIC obtuvo un porcentaje de cumplimiento del 3,3 % para el nivel de madurez 2 (Gestionado) y del 0,0 % para el nivel 3 (Definido). De acuerdo con las reglas de determinación del nivel de madurez, un resultado inferior al 50 % en el nivel 2 ubica a la organización en el **Nivel 1 (Inicial)**, caracterizado por procesos de *testing ad hoc*, reactivos y completamente dependientes del conocimiento y criterio individual de cada desarrollador. Este nivel implica que las pruebas no están planificadas, no existen políticas ni procedimientos formalizados y los resultados son impredecibles e irrepetibles.

Como se detalla en los resultados consolidados (véase la Tabla 4, Anexo D), el único criterio que obtuvo respuesta afirmativa en toda la evaluación fue el **C15**, correspondiente a la disponibilidad del ambiente de pruebas, dado que la organización cuenta con un ambiente de *staging* que replica el 99 % del esquema de producción. Este hallazgo aislado permitió que el área de proceso **2.5 (Ambientes de Prueba)** alcanzara un 33 %, siendo la única área con cumplimiento parcial en todo el diagnóstico; sin embargo, su peso relativo de 0,12 dentro del nivel 2 no fue suficiente para impactar significativamente el resultado global, que se mantuvo en 3,3 %.

Las respuestas abiertas de los cuestionarios reforzaron los resultados cuantitativos: el desarrollador 2 señaló que las pruebas son de calidad aleatoria por la ausencia de roles definidos y seguimiento centralizado; el desarrollador 1 identificó la implementación de CI/CD como la mejora más urgente; y el desarrollador 3 señaló la ausencia total de parámetros de validación de código. Desde el nivel táctico, el líder del subproceso identificó como prioridad inmediata la formalización de un proceso de pruebas estructurado con documentos y herramientas definidas, mientras que el líder del proceso destacó la necesidad de personal capacitado en calidad del software como condición habilitante para cualquier mejora.

La Figura 3.2 ilustra el perfil de madurez actual de la EPSI AIC mediante un diagrama radar que contrasta el estado actual; con nueve de diez áreas de proceso en 0 % y una sola área en 33 %, frente al perfil objetivo del proceso *TO-BE* propuesto.

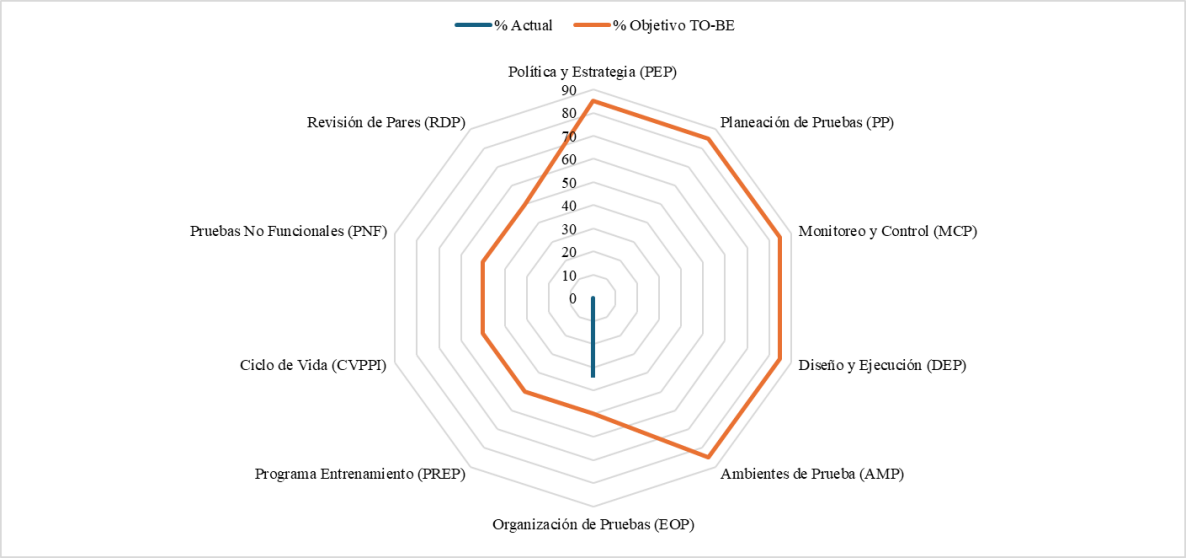


Figura 3.2: Radar de madurez: perfil actual vs. perfil objetivo TO-BE. Fuente: elaboración propia.

Esta evaluación, triangulada con los hallazgos de las entrevistas cualitativas y el análisis documental, proporcionó una caracterización cuantitativa y metodológicamente fundamentada del estado de madurez que sustentó las decisiones de diseño del proceso *TO-BE* presentadas en la sección 3.3.

La selección de este instrumento se fundamentó en tres criterios:

1. Su derivación rigurosa del modelo TMMi con pesos ponderados obtenidos de una encuesta aplicada por la Fundación TMMi a 202 organizaciones en 2020, lo que garantiza su validez empírica.

2. Su adaptación al contexto latinoamericano y su aplicabilidad práctica en organizaciones con baja madurez inicial en procesos de *testing*, característica relevante para el contexto de la EPSI AIC.

3. La disponibilidad del instrumento validado y documentado, que permitió su aplicación directa sin requerir un proceso de construcción y validación adicional.

El uso del instrumento adaptado no reemplaza el modelo TMMi original, sino que operacionaliza sus criterios en un formato aplicable al contexto del proyecto.

3.2. Diagnóstico del Estado Actual

Esta sección permite visualizar los resultados del diagnóstico elaborado a partir de la triangulación de los tres frentes metodológicos descritos en la sección anterior. Se abordan cuatro puntos complementarios que son:

- Caracterización Organizacional.
- Hallazgos de las entrevistas cualitativas.
- Resultados del análisis documental.
- Línea base cuantitativa del estado actual.

3.2.1. Caracterización de la Organización EPSI AIC

La caracterización de la organización constituye el punto de partida del diagnóstico, ya que comprender el contexto institucional, tecnológico y operativo de la EPSI AIC es condición necesaria para diseñar un proceso de aseguramiento de calidad que sea pertinente, viable y sostenible en su entorno específico. Esta sección no pretende ser una descripción exhaustiva de la organización, sino una aproximación estratégica a los elementos que inciden directamente en las prácticas de desarrollo de software y en las condiciones bajo las cuales se implementará el marco de QA propuesto. Se abordan cuatro dimensiones complementarias: el contexto organizacional y los roles relevantes para el proyecto, el marco normativo aplicable, la arquitectura tecnológica actual y el proceso de desarrollo vigente. El análisis organizacional se realizó con base en el organigrama institucional oficial actualizado en agosto de 2025, el cual correspondía a la versión más reciente aprobada por la organización y se encontraba vigente al momento del diagnóstico efectuado en enero y febrero de 2026. Durante el periodo de ejecución del proyecto no se reportaron cambios estructurales formales en la organización que modificaran la distribución de roles del proceso de Tecnologías de la Información.

3.2.1.1. Contexto Organizacional

Naturaleza y Misión de EPSI AIC La Asociación Indígena del Cauca (AIC) opera como Entidad Promotora de Salud Indígena (EPS-I), La Asociación Indígena del Cauca AIC-EPS-I es una entidad Pública de carácter especial que tiene como objeto fortalecer el Sistema Indígena de salud propio e intercultural-SISPI de la población afiliada, a través de la administración de los recursos y el aseguramiento del cuidado integral de salud; respetando la diversidad étnico y cultural de cada pueblo y comunidad orientada al buen vivir (Asociación Indígena del Cauca EPS-I, 2025)

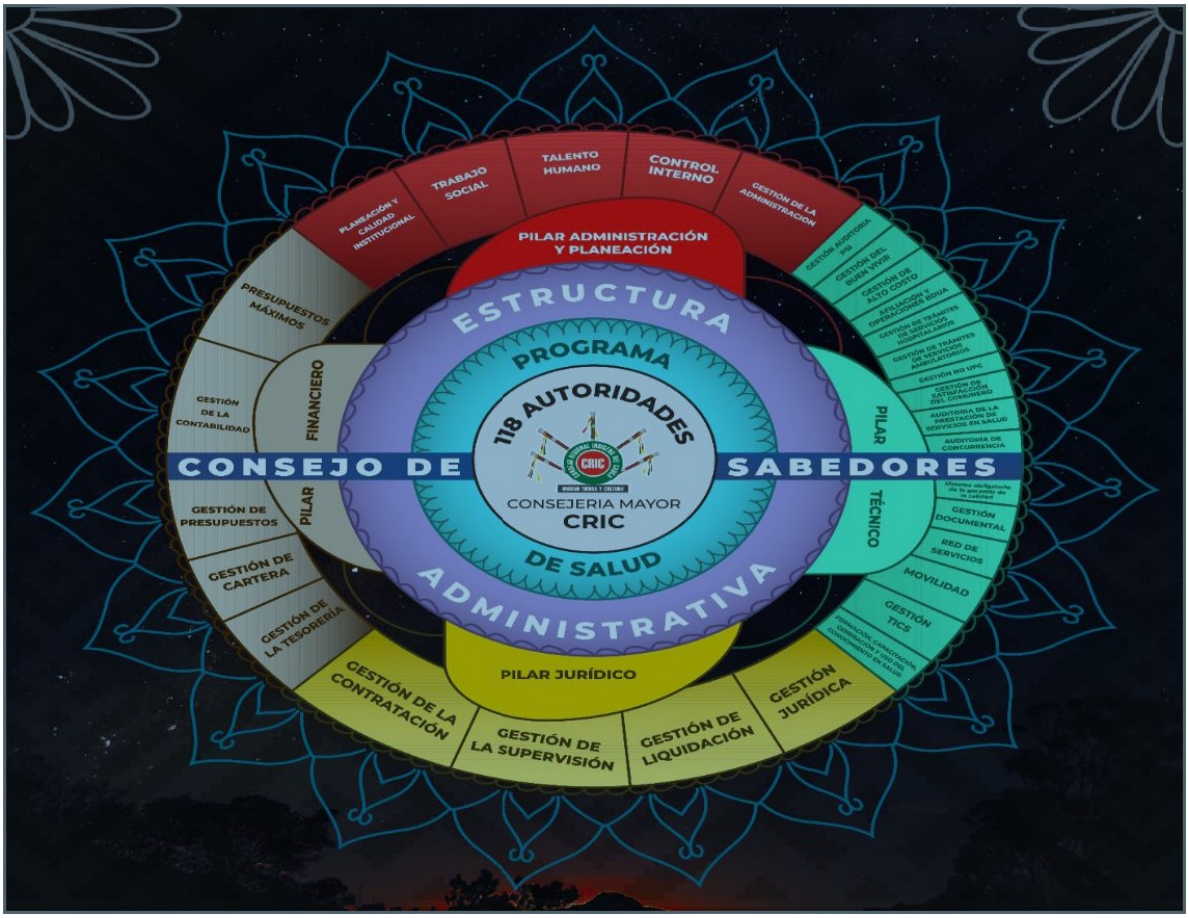


Figura 3.3: Estructura Organizacional EPSI AIC, Fuente: Manual de Armonia

La estructura organizacional de la Asociación Indígena del Cauca – EPSI AIC ha sido concebida como un sistema funcional orientado por procesos, en coherencia con el principio de gestión propia definido por las autoridades indígenas del Cauca. Este modelo parte de la necesidad de garantizar una operación institucional que responda con eficiencia, transparencia y pertinencia cultural a las dinámicas territoriales y a los mandatos emanados desde los espacios políticos de decisión colectiva (Asociación Indígena del Cauca EPS-I, 2025).

Roles relevantes para el proyecto

- **Representante Legal:** Máxima autoridad administrativa e institucional de la EPSI AIC. Su relevancia para el proyecto radica en el respaldo estratégico que puede brindar a la implementación del proceso de aseguramiento de calidad, dado que cualquier iniciativa de mejora tecnológica requiere aval desde este nivel.
- **Pilar Técnico:** Responsable de la coordinación y supervisión de los procesos técnicos de la organización. Es el puente entre la dirección estratégica y los procesos operativos de TI, por lo que su apoyo es determinante para la viabilidad y sostenibilidad del proceso de QA propuesto.
- **Líder de Gestión TICs:** Responsable directo de la operación continua y segura de los servicios tecnológicos de la organización. Para este proyecto representa el actor táctico principal, con capacidad de decisión sobre herramientas, recursos y prioridades del subproceso de desarrollo.
- **Líder del Subproceso de Desarrollo:** Responsable de planificar, coordinar y supervisar el ciclo completo de desarrollo de software. Es el actor más directamente involucrado en la implementación del marco de aseguramiento de calidad, dado que el proceso propuesto se integra en su subproceso.
- **Desarrollador:** Responsables de la implementación técnica de los sistemas de información. Son los ejecutores directos de las prácticas de desarrollo y quienes adoptarán e incorporarán las pruebas automatizadas en su flujo de trabajo cotidiano.
- **Líder de Gestión del Buen Vivir:** Responsable del proceso misional de seguimiento a rutas de promoción y mantenimiento de la salud. Representa la perspectiva del usuario crítico, dado que sus procesos dependen directamente de la confiabilidad de los sistemas de información para actividades de alto impacto como vigilancia epidemiológica y evaluación de contratos capitados.
- **Líder de Gestión de Satisfacción del Comunero:** Responsable de la evaluación del grado de satisfacción de los comuneros y la gestión de PQRSF. Aporta la perspectiva del usuario final sobre la calidad y confiabilidad percibida de los sistemas, siendo relevante para entender el impacto externo de las deficiencias tecnológicas.

Proceso de Gestión TICs Compuesto por aproximadamente **27 personas** (según entrevista con Líder de Gestión TICs), organizadas en los siguientes subprocesos: Administración de operaciones, Mesa de ayuda, Desarrollo de software, Gerencia de la información, Seguridad informática e Implementación de sistemas de información

Perfil	Cantidad	Observaciones
Aprendices SENA	2	Rotación cada 6 meses
Tecnólogos	3	Personal de apoyo técnico
Ingenieros de Sistemas	3	Algunos cuentan con especialización
Total	**8**	**Organizados en 2 células de desarrollo**

Tabla 3.4: Composición del equipo de desarrollo de software EPSI AIC.

Estructura del Subproceso de Desarrollo Distribución por células:

- **Célula 1:** Especializada en tecnologías python (Django y FastAPI) typescript, javascript.
- **Célula 2:** Enfoque en otras tecnologías (según necesidades).

3.2.1.2. Marco Normativo y Regulatorio Aplicable

La EPSI AIC opera bajo el marco regulatorio de seguridad y privacidad de la información del sector salud colombiano, del cual son especialmente relevantes para este proyecto la Resolución 866 de 2021 del Ministerio de Salud y Protección Social, que exige garantizar la confidencialidad, integridad, disponibilidad y trazabilidad de la información clínica, junto con políticas de control de accesos, gestión del riesgo y alineación con estándares internacionales de ciberseguridad, y la normatividad de *Habeas Data* (Ley 1581 de 2012 y Decreto 1377 de 2013), que regula el tratamiento de datos personales sensibles, particularmente crítico al tratarse de población indígena: consentimiento informado y derechos de acceso, rectificación y supresión. Este marco es directamente pertinente para el proyecto, pues la implementación de pruebas automatizadas en el ciclo de vida del *software* permite verificar de forma continua y sistemática la correcta gestión de permisos, el cifrado de datos y los controles de privacidad exigidos, contribuyendo al cumplimiento normativo en cada ciclo de desarrollo.

3.2.1.3. Arquitectura Tecnológica Actual

Infraestructura física La EPSI AIC cuenta con un datacenter propio con servidores, racks, switches, planta eléctrica de respaldo eléctrico (instalada en 2026) y equipos adquiridos hace varios años, actualmente en proceso de inventario por parte del subproceso de infraestructura para mitigar riesgos de obsolescencia.

Sistemas de Información Misionales Los sistemas misionales más críticos no son desarrollos propios, sino que dependen de un proveedor externo que gestiona la mayoría de módulos del ciclo de operación de la EPS en salud. Una limitación identificada es que Gestión TICs no tiene acceso directo a las bases de datos, aunque el proveedor expone algo de información mediante interfaces.

La organización también desarrolla sus propios sistemas expuestos a través de la página web de AIC o por medio de links de acceso, incluyendo el Sistema de gestión de PQRSD (Peticiones,

Quejas, Reclamos, Sugerencias y Denuncias) desarrollado en el año 2024, el Módulo de referencia 24/7, Los sistemas de gestión del buen vivir y más de 20 aplicativos *legacy* desarrollados hace más de 10 años con tecnología específica cuyo código fuente no está disponible, lo cual complica su soporte cuando presenta fallas. Estos sistemas de desarrollo propio constituyen el alcance de intervención del presente proyecto. A diferencia de los sistemas del proveedor externo, sobre los cuales no se tiene acceso al código fuente ni a las bases de datos de forma directa, los sistemas internos permiten la implementación completa del ciclo de aseguramiento de calidad: desde el diseño de casos de prueba hasta la automatización e integración en el *pipeline* de GitLab. Esta distinción es fundamental para comprender el alcance real y la viabilidad técnica de la solución implementada.

Capa	**Tecnologías Utilizadas**
Backend	Django, FastAPI, Java Spring Boot, Node Js
Frontend	Angular, React
Bases de Datos	Oracle
Control de Versiones	Git, Gitlab
Contenerización	Docker, Docker compose
Calidad de Código	Sin herramienta

Tabla 3.5: Stack tecnológico del subproceso de desarrollo. Fuente: Entrevistas con desarrolladores.

Ambientes de Trabajo La infraestructura actual de desarrollo se distribuye en tres niveles lógicos que permiten la transición del código desde la construcción local hasta la operación final, como se detalla en la Figura 3.4.

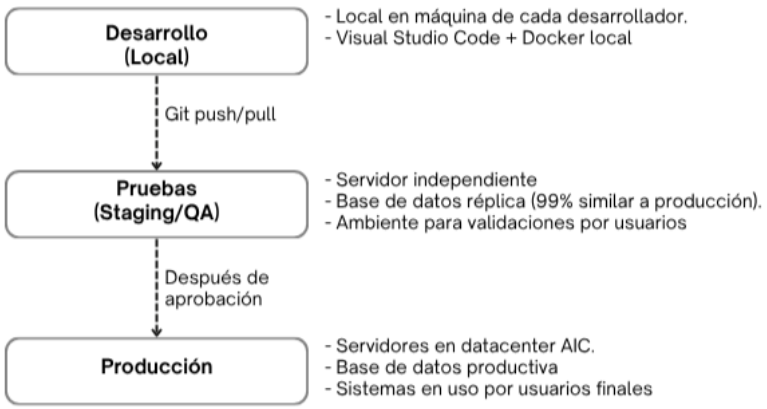


Figura 3.4: Arquitectura de ambientes. Elaboración propia

3.2.1.4. Procesos Actuales de Desarrollo y Operación (AS-IS)

El proceso actual de desarrollo de *software* en la EPSI AIC abarca desde la identificación de la necesidad hasta el mantenimiento y soporte de la solución en producción, involucrando distintos actores y etapas. Su caracterización se realizó mediante revisión documental, análisis de los procesos existentes y validación con el equipo técnico responsable, considerando las fases principales del ciclo de vida del *software* y de la operación tecnológica.

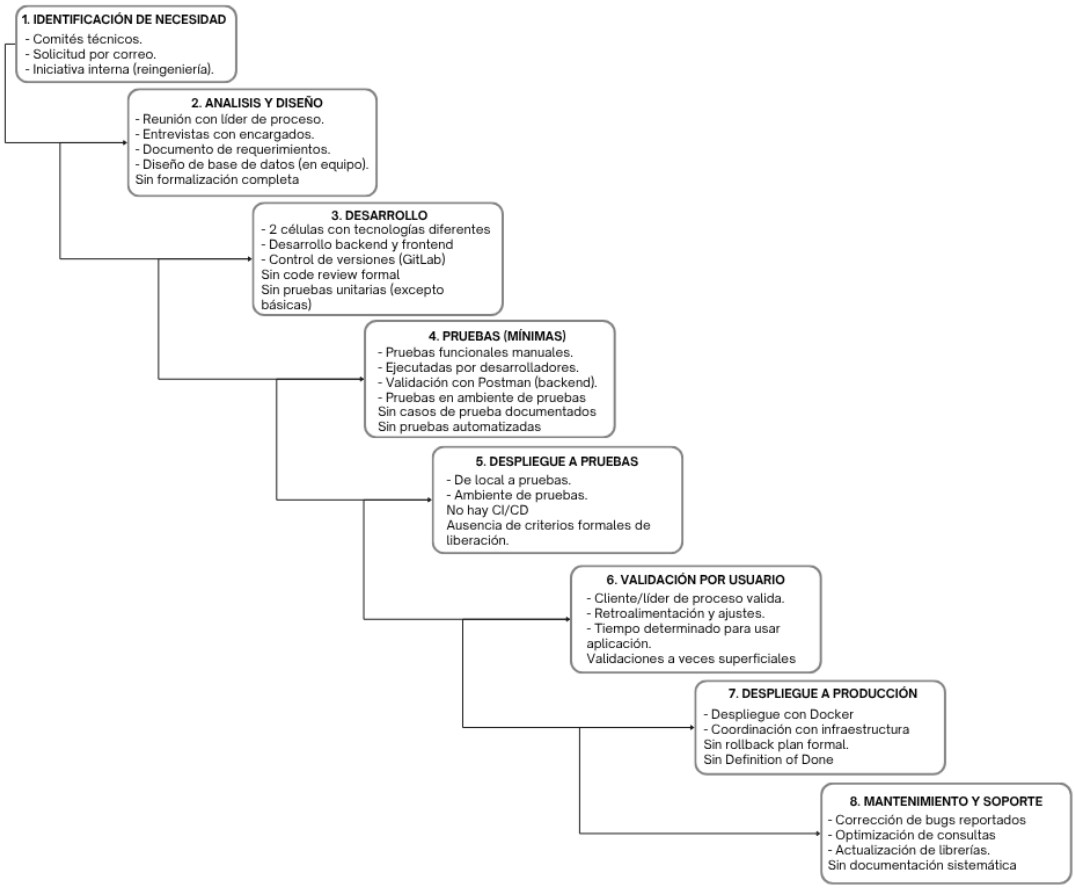


Figura 3.5: Proceso AS-IS de desarrollo de software en EPSI AIC

La organización adopta conceptualmente el marco *Scrum* para la gestión del desarrollo; sin embargo, el flujo operativo evidencia una ejecución estructurada por fases consecutivas y dependientes entre sí, que corresponde en la práctica a un modelo en cascada. No se identificaron iteraciones definidas, entregas incrementales ni ceremonias *Scrum* regulares, lo que confirma una brecha significativa entre la metodología declarada y la operación real del subproceso de desarrollo. La Tabla 3.6 sintetiza, fase por fase, el estado actual del proceso y las brechas identificadas en cada una.

Fase	Descripción del estado actual	Brecha identificada
1. Identificación de la necesidad	Surge de comités técnicos, correos o normatividad del sector salud; se realiza completa antes de construir.	Requerimientos informales y sin documentación estructurada; las fallas de análisis son la principal causa de incidentes en producción.
2. Análisis y diseño	Reuniones con líderes, análisis de normatividad, documento de requerimientos y diseño de la base de datos, previos al desarrollo.	*Scrum* parcial: sin *Scrum Master* certificado, sin *dailies* regulares ni *Definition of Done* formal; se avanza sin criterios de aceptación.
3. Desarrollo	Implementación con control de versiones en GitLab y pruebas funcionales mínimas, manuales, por los propios desarrolladores (Postman).	Sin revisión de código entre pares; deuda técnica estimada del 30 %–35 %.
4. Pruebas	Pruebas funcionales mínimas y manuales al finalizar el desarrollo, concentrando la validación al final del ciclo.	Fase de mayor riesgo: sin pruebas unitarias, de integración ni de regresión formales; verificación manual de 5–30 min por módulo, sin trazabilidad y dependiente de un único responsable.
5. Despliegue a pruebas	Publicación manual en el ambiente de pruebas para validación funcional.	Sin *pipeline* de integración continua; despliegues manuales propensos a error y sin criterios formales de liberación.
6. Validación con el usuario	El líder del proceso revisa la funcionalidad y retroalimenta mediante una matriz de observaciones; luego autoriza el despliegue.	Validación sin criterios técnicos de calidad ni documento formal de aceptación; genera ajustes frecuentes posteriores.
7. Despliegue a producción	Despliegue con apoyo de infraestructura usando Docker; sin ventanas formales de despliegue.	Monitoreo *post*-despliegue reactivo: sin alertas ni monitoreo automatizado; las fallas se conocen solo cuando el usuario las reporta.
8. Mantenimiento y soporte	Corrección de errores, optimización de consultas y actualización de componentes; concentra aprox. 70 % del tiempo del Líder del Subproceso.	Documentación técnica no sistemática que, sumada a la rotación cada dos años, genera pérdida de conocimiento institucional.

Tabla 3.6: Síntesis del proceso *AS-IS* de desarrollo y operación, con las brechas identificadas por fase. Fuente: elaboración propia con base en revisión documental y entrevistas.

A partir de este análisis se evidencia que, aunque la organización adopta lineamientos asociados a metodologías ágiles, su ejecución operativa mantiene un comportamiento predominantemente secuencial. Esta brecha entre el enfoque declarado y la práctica real de desarrollo y operación fun-

damenta la necesidad de definir un modelo objetivo (*TO-BE*) orientado a prácticas ágiles efectivas, automatización del ciclo de entrega y fortalecimiento de la operación tecnológica, presentado en la sección 3.3.

3.2.2. Hallazgos Cualitativos de las Entrevistas

En el marco del diagnóstico del estado actual de las prácticas de aseguramiento de calidad en la EPSI AIC, se realizaron siete entrevistas en profundidad a actores clave distribuidos en los niveles estratégico, táctico y operativo de la organización. Las entrevistas se llevaron a cabo entre el mes de febrero de 2026 y totalizaron aproximadamente 323 minutos de grabación. Los participantes correspondieron a siete roles organizacionales identificados previamente: Representante Legal, Pilar Técnico, Líder de Gestión TICs, Líder del Subproceso de Desarrollo, Líder de Gestión de Satisfacción al Comunero, Líder de Gestión del Buen Vivir y el rol de Desarrollador de Software, este último representado por dos profesionales del equipo de desarrollo.

El análisis de las transcripciones se realizó mediante codificación temática, identificando categorías emergentes a partir del contenido de las respuestas. A continuación se presentan los hallazgos organizados en seis dimensiones analíticas que articulan las perspectivas de los distintos actores.

3.2.2.1. Ausencia Total de un Proceso Formal de Pruebas

El hallazgo más consistente y transversal en todas las entrevistas fue la inexistencia de un proceso estructurado de pruebas de software. Desde el nivel estratégico hasta el operativo, los entrevistados reconocieron e identificaron que las validaciones actuales se limitan a pruebas funcionales manuales ejecutadas informalmente, sin criterios estandarizados, sin trazabilidad sistemática y sin responsable exclusivo. El **Líder de Gestión TICs**, señaló que aunque en teoría deberían existir pruebas técnicas y automatizadas, estas no se han evidenciado de manera sistemática en la organización. Mientras que el **Líder del Subproceso de Desarrollo**, calificó las pruebas actuales como: *"muy mínimas y se enfocan principalmente en aspectos funcionales"*.

El **desarrollador 1** confirmó que no se escriben pruebas unitarias y que la verificación de funcionalidades se realiza mediante pruebas manuales pensando en cómo interactuaría el usuario con el aplicativo. El **desarrollador 2** señaló que, si bien realiza algunas pruebas unitarias básicas para prevenir errores específicos de consultas a base de datos, estas son iniciativas personales no institucionalizadas, es decir dependen de cada desarrollador y de su conocimiento. Este hallazgo confirmó la línea base cuantitativa establecida en el planteamiento del problema: ausencia total de cobertura en pruebas automatizadas y 100 % de dependencia en validaciones manuales.

Tipo de Prueba	Estado Actual de la EPSI AIC
Pruebas unitarias	No implementadas de forma sistemática. Solo iniciativas individuales aisladas.
Pruebas de integración	Inexistentes. Sin mecanismo formal para verificar interacciones entre módulos.
Pruebas de regresión	Revisión manual del código como sustituto informal de pruebas de regresión
Pruebas automatizadas	Nunca implementadas, aunque han sido consideradas en el pasado.
Documentación de pruebas	Inexistente. Las actas de reunión son el único registro de referencia disponible.

Tabla 3.7: Síntesis del estado actual de las prácticas de pruebas en la EPSI AIC. Fuente: elaboración propia con base en entrevistas.

3.2.2.2. Dependencia Crítica de Conocimiento Individual y Alta Rotación de Personal

Un segundo hallazgo crítico, señalado de forma recurrente por actores de todos los niveles, es la dependencia operacional del conocimiento no documentado de personas específicas. Esta situación constituye un punto único de falla que puede paralizar los procesos de desarrollo y despliegue cuando las personas clave no se encuentran disponibles. El **líder del subproceso de desarrollo** identificó que la dependencia de validaciones manuales por parte de personas específicas puede afectar significativamente los tiempos de entrega. El **desarrollador 1** complementó este hallazgo señalando que la dependencia de una sola persona para ciertas validaciones o permisos puede convertirse en un factor que impacta los tiempos de entrega. Esta vulnerabilidad se agrava por la política organizacional de rotación de personal, documentada por **El líder del proceso de gestion TICs**: *"La rotación de personal debido a las políticas de la organización (cambios cada dos años) es una dificultad, ya que las personas dejan el proceso cuando han adquirido experiencia."*

El **líder del subproceso de desarrollo** manifestó que la documentación del conocimiento para el mantenimiento futuro no se está llevando a cabo de forma sistemática, confirmando la ausencia de mecanismos de transferencia de conocimiento técnico. La combinación de conocimiento tácito no documentado y alta rotación genera un ciclo de pérdida de capacidad institucional que impide la acumulación de buenas prácticas de aseguramiento de calidad.

3.2.2.3. Ausencia de Métricas de Calidad e Indicadores de Proceso

Las entrevistas permitieron evidenciar que la EPSI no cuenta con métricas formales para medir la calidad del software ni para identificar defectos de manera sistemática. La medición del desempeño del equipo de desarrollo se basa exclusivamente en el tiempo de entrega, comparando desarrollos anteriores con los actuales para estimar plazos, sin incluir indicadores de calidad del producto. El **líder de Gestion TICs** confirmó que no se utilizan métricas o indicadores para evaluar la calidad

del software que sale a producción. El **líder del subproceso** reconoció que los tiempos se evalúan de forma comparativa, sin reportes formales de calidad. La ausencia de un sistema de alertas o monitoreo activo fue reportada por los **desarrolladores de software**: *."Actualmente, no existen sistemas de alertas o monitoreo de servicios; solo somos avisados de fallas cuando el cliente las reporta."*

Esta situación configura un modelo de monitoreo completamente reactivo, en el que el tiempo entre la ocurrencia de un defecto y su detección depende de la visibilidad del impacto sobre los usuarios. Esto incrementa significativamente el costo de corrección de errores y el riesgo para la continuidad operativa de los sistemas misionales.

3.2.2.4. Impacto de las Fallas en los Procesos Misionales de Salud

Las perspectivas de los líderes de procesos misionales revelaron el impacto concreto que tienen las deficiencias en la calidad del software sobre la operación de servicios de salud para comunidades indígenas. La **líder de Gestión del Buen Vivir**, destacó que su proceso depende críticamente de los sistemas de información para actividades de alto impacto como la vigilancia epidemiológica, el seguimiento a gestantes, el control de enfermedades crónicas y la evaluación de contratos cápita. La líder identificó el módulo de contratos como una funcionalidad de interrupción crítica: *."El módulo de contratos es considerado crítico porque el proceso no puede continuar con la evaluación si no existe un contrato debidamente establecido e ingresado en el aplicativo."*

La líder señaló además que los sistemas de RIPS (Registro Individual de Prestación de Servicios de Salud) y Afiliaciones requieren mayor atención por su criticidad, dado que los errores en estos datos generan estimaciones con datos poco reales que pueden llevar a errores en la gestión y al malgasto de recursos en actividades de salud pública. El **líder de Satisfacción al Comunero**, por su parte, reportó que las dificultades operativas del sistema de PQRS se asocian principalmente a factores externos como fallas de internet y cortes eléctricos, aunque subrayó la necesidad de un proceso formal de evaluación y seguimiento preventivo de los sistemas para evitar fallas graves a futuro.

3.2.2.5. Brechas en la Metodología de Desarrollo y Gestión de Requerimientos

Los hallazgos evidencian que, aunque la organización declara la adopción formal de SCRUM, el proceso operativo mantiene características predominantemente secuenciales, con validaciones concentradas al final del desarrollo y limitada iteración estructurada. Esta situación configura un modelo híbrido emergente, donde coexisten artefactos ágiles aislados con prácticas tradicionales de gestión. Este contexto organizacional fundamenta el diseño del proceso TO-BE como metodológicamente agnóstico, orientado no a imponer un framework específico, sino a introducir prácticas de aseguramiento de calidad compatibles tanto con esquemas predictivos como con enfoques ágiles en evolución.

3.2.2.6. Reconocimiento de la Necesidad y Disposición Organizacional para el Cambio

Un hallazgo transversal de gran relevancia estratégica es el consenso entre todos los entrevistados sobre la necesidad urgente de implementar un proceso formal de aseguramiento de calidad. Las entrevistas revelaron una disposición organizacional favorable, acompañada de expectativas concretas en todos los niveles. El **líder del proceso de Gestion TICs** expresó que la propuesta de automatización de pruebas es vista como muy interesante para mitigar los riesgos actuales. El **líder del subproceso** estableció una métrica de éxito clara: *"Si el proceso de QA no funciona, significa que están saliendo bugs a producción. La aplicación no debe pasar a producción hasta que cumpla con todos los requerimientos."* A nivel operativo, el **desarrollador 1** señaló que la automatización no es una carga sino una forma viable de mejorar el desarrollo, mientras que el **desarrollador 2** la identificó como lo mejor que podría sucederle al equipo. Desde la perspectiva de usuarios, el **líder de satisfacción al comunero** valoró la propuesta como necesaria para asegurar que los sistemas no queden obsoletos, y la **lider de Gestion del Buen Vivir** enfocó sus expectativas en la confiabilidad de los sistemas para las evaluaciones cápita, que tienen impacto financiero directo. Cabe destacar que todos los actores técnicos señalaron la falta de formación formal en técnicas de diseño de pruebas y frameworks de QA como una brecha habilitante que debe atenderse de forma paralela a la implementación técnica.

3.2.3. Línea Base Referencial del Estado Actual

Dado que la EPSI AIC no contaba con registros históricos formales, la línea base se construyó a partir de estimaciones declaradas por actores clave, trianguladas entre sí para garantizar consistencia interna. Estos valores tienen carácter referencial y sirven como punto de comparación tendencial, no como métricas de precisión objetiva. Esta ausencia constituye en sí misma un hallazgo crítico del diagnóstico, plenamente coherente con el Nivel 1 de madurez identificado mediante la evaluación TMMi. En consecuencia, siguiendo la recomendación metodológica de Yin (2018) para contextos organizacionales donde los procesos no están formalizados y el conocimiento reside principalmente en las personas, la línea base se construyó mediante estimaciones declaradas por actores clave del nivel táctico y operativo, trianguladas entre sí para aumentar su consistencia interna.

Es importante señalar que los valores presentados tienen carácter referencial y no instrumental: no representan mediciones objetivas previas a la intervención, sino el estado percibido y estimado antes de la misma. Su propósito no es establecer una línea base de precisión métrica, sino documentar el punto de partida desde la perspectiva de quienes operan el sistema, permitiendo una comparación cualitativa y tendencial del cambio generado tras la implementación. Esta aproximación es coherente con los diseños cuasi-experimentales recomendados por Cook y Campbell (1979) para contextos donde no existen grupos de control independientes ni registros históricos, reconociendo explícitamente que la ausencia de datos objetivos previos constituye una limitación de la evaluación de impacto que debe interpretarse con la cautela correspondiente.

Indicador	Valor estimado	Fuente	Categoría
Cobertura de pruebas automatizadas	0 %	Todos los entrevistados técnicos	Confirmado
Dependencia en validaciones manuales	100 %	Todos los entrevistados técnicos	Confirmado
Tiempo de ejecución de regresión manual	5 min – 30 min por módulo	Desarrolladores	Declarativo
Tiempo de corrección de errores en producción	5 min – 4 horas (casos complejos hasta 1 semana)	Desarrolladores	Declarativo
Responsables exclusivos de pruebas funcionales	Inexistente	Líder Subproceso Desarrollo	Confirmado
Deuda técnica estimada	30 % – 35 %	Líder Subproceso Desarrollo	Declarativo
Métricas formales de calidad disponibles	0	Líder Gestión TICs y Desarrollo	Confirmado
Documentación formal de casos de prueba	Inexistente	Análisis documental	Confirmado
Sistema de monitoreo o alertas en producción	Inexistente	Desarrolladores	Confirmado
Detección de fallas en producción	Reactiva — solo por reporte del usuario	Desarrolladores	Confirmado

Tabla 3.8: Línea base cuantitativa del estado actual de las prácticas de aseguramiento de calidad en la EPSI AIC. Fuente: elaboración propia con base en entrevistas realizadas entre el mes de febrero de 2026.

Los datos presentados en la tabla anterior muestran un panorama de riesgo operativo significativo para una entidad que depende directamente de la estabilidad y funcionamiento de sus sistemas de información. Los indicadores de cero cobertura automatizada y un modelo completamente reactivo de detección de fallas; configura un escenario donde los errores solo se conocen cuando ya han impactado a los usuarios de los sistemas misionales. El tiempo de corrección de errores, que puede extenderse hasta una semana en casos de mayor complejidad, sumado a una deuda técnica estimada entre el 30 % y el 35 %, evidencia que el costo operativo de la ausencia de procesos formales de calidad es progresivo y acumulativo.

3.2.4. Síntesis de Brechas Identificadas

Los hallazgos de las entrevistas, analizados en conjunto con el análisis documental y el proceso AS-IS, permiten construir una caracterización integral del estado actual. La triangulación de perspectivas desde los niveles estratégico, táctico y operativo, así como desde el equipo técnico y los usuarios de los sistemas misionales, proporciona una visión consistente y multidimensional de las brechas existentes.

Brecha Identificada	**Evidencia / Hallazgo**
Ausencia de pruebas automatizadas	0 % de cobertura. Confirmado por todos los entrevistados técnicos.
Dependencia de validaciones manuales	100 % de pruebas son manuales y funcionales. Sin criterios estandarizados.
Punto único de falla operacional	Sin responsable exclusivo de pruebas. Alta rotación de personal cada 2 años.
Ausencia de métricas de calidad	Sin indicadores de proceso ni de producto. Sin trazabilidad de defectos.
Monitoreo reactivo de producción	Fallas conocidas solo cuando el usuario las reporta. Sin alertas automáticas.
Gestión informal de requerimientos	Requerimientos ambiguos. Ajustes frecuentes post entrega. Deuda técnica 30 %–35 %.
Ausencia de documentación técnica	Bitácoras no utilizadas. Conocimiento tácito no transferido.
Brecha de capacitación en *testing*	Equipo sin formación formal en técnicas de pruebas ni *frameworks* de QA.
Ausencia de *pipeline* de integración continua	Los despliegues al ambiente de pruebas se realizan de forma manual, sin criterios formales de liberación ni automatización CI/CD.

Tabla 3.9: Síntesis de brechas identificadas en el diagnóstico de la EPSI AIC. Fuente: elaboración propia.

El diagnóstico revela que la EPSI AIC se encuentra en un estado de madurez muy bajo en sus prácticas de *testing*, caracterizado por la ausencia total de procesos formalizados, la dependencia de conocimiento individual no documentado, la carencia de métricas de calidad y la gestión reactiva de incidentes en producción. Esta situación representa un riesgo significativo para la continuidad operativa de sistemas que soportan servicios críticos de salud para comunidades indígenas vulnerables.

Al mismo tiempo, el diagnóstico identificó condiciones favorables para el cambio: disposición organizacional positiva, capacidades técnicas existentes en el equipo de desarrollo, infraestructura tecnológica disponible y apoyo de la alta gerencia. Estos factores habilitantes son determinantes

para el éxito de la propuesta de aseguramiento de calidad desarrollada en las fases siguientes del proyecto.

3.3. Diseño del Proceso de Aseguramiento de Calidad (TO-BE)

Tomando como base el diagnóstico y la caracterización realizada, esta sección permite presentar el diseño del proceso de aseguramiento de calidad propuesto para la EPSI AIC. El proceso *TO-BE* responde de manera directa a cada brecha identificada. El proceso propuesto se fundamenta en el principio de *Shift Left Testing*, es decir; desplazar la actividad de pruebas hacia las etapas más tempranas del ciclo de desarrollo, donde el costo de corrección de errores es significativamente menor (Boehm, 1988). Esto implica que las pruebas dejan de ser una fase final y se integran como una responsabilidad continua a lo largo de todo el ciclo de vida del software.

3.3.1. Principios de diseño del proceso

El proceso *TO-BE* se diseñó bajo seis principios rectores que responden al contexto específico de la EPSI AIC y garantizan su viabilidad y sostenibilidad:

Tabla 3.10: Principios de diseño del proceso TO-BE. Fuente: elaboración propia.

Principio	**Descripción**	**Brecha Identificada**
Shift Left Testing	Las pruebas inician desde la fase de análisis, no desde la fase de desarrollo.	Ausencia total de pruebas en etapas tempranas del ciclo.
Automatización integrada en CI/CD	Las pruebas se ejecutan automáticamente ante cada cambio de código mediante GitLab, sin intervención manual.	100 % de dependencia en validaciones manuales por un único responsable.
Responsabilidad distribuida	Cada desarrollador es responsable de escribir y mantener las pruebas de su código, bajo la guía del Líder del Subproceso.	Punto único de falla operacional.
Pruebas como documentación	Los casos de prueba en código sirven como documentación viva del comportamiento esperado del sistema.	Pérdida de conocimiento institucional por alta rotación de personal.
Mejora incremental	El proceso inicia con metas de cobertura modestas y alcanzables (15 %), con escalamiento progresivo conforme el equipo gana madurez.	Baja madurez en procesos de *testing*; riesgo de abandono por metas inalcanzables.

Continúa en la siguiente página...

Tabla 3.10 – Continuación de la página anterior

Principio	Descripción	Brecha Identificada
Alineación normativa	El diseño del proceso incorpora controles que contribuyen al cumplimiento de la Resolución 866 de 2021 (confidencialidad, integridad y trazabilidad).	Riesgo de incumplimiento normativo identificado en el diagnóstico.

3.3.2. Descripción del proceso TO-BE.

El proceso TO-BE mantiene las ocho fases del ciclo de desarrollo actual, pero introduce actividades específicas de aseguramiento de calidad en cada una de ellas. La figura 6 ilustra el flujo rediseñado con los puntos de integración de QA resaltados.

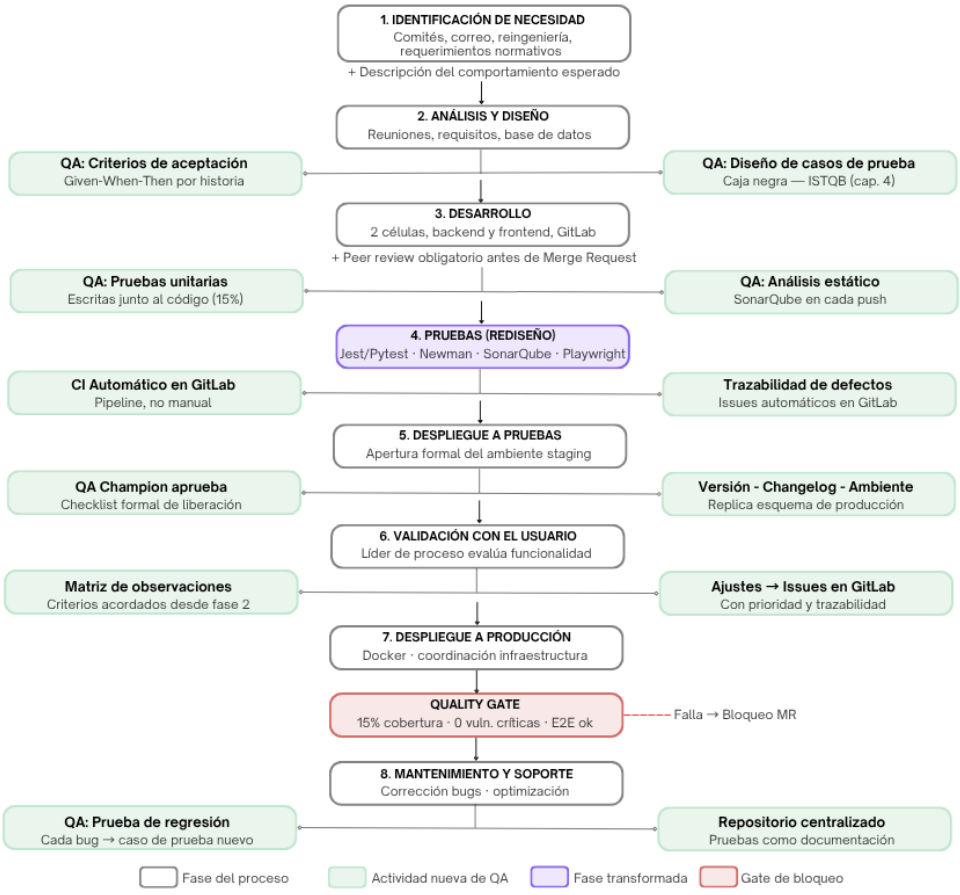


Figura 3.6: Proceso TO-BE de desarrollo de software en EPSI AIC. Fuente: Elaboración propia

Fase 1 - Identificación de la necesidad (con QA)

No se realizan cambios de estructura con respecto al AS-IS; sin embargo, en esta fase es necesario establecer que todas la solicitud de desarrollo deberá incluir, en su formalización inicial, por lo menos una descripción del comportamiento esperado que sirve como base para el diseño posterior de los casos de pruebas. Con esto se obliga a que los requerimientos sean suficientemente concretos desde su origen, de manera que se reducen posibles ambigüedades que causen incidentes en la fase de producción.

Fase 2 - Análisis y diseño (con QA)

Se incorpora la actividad de definición de criterios de aceptación para cada una de las historias de usuario o requerimiento, redactados de forma testable (Given - When - Then o equivalente, adaptado al nivel de formalización del requerimiento). En esta fase se diseñan también los casos de prueba de caja negra para las funcionalidades priorizadas, siguiendo las técnicas del ISTQB definidas en el capítulo 4. Este es el cambio de mayor impacto en el proceso: al tener casos de prueba antes de desarrollar, el equipo sabe exactamente qué debe validar.

Adición clave: el Líder del Subproceso de Desarrollo valida que cada historia de usuario tenga criterios de aceptación testeables antes de ingresar al sprint.

Fase 3 - Desarrollo (con QA)

Se introduce la práctica de pruebas unitarias como parte del trabajo de desarrollo, el desarrollador; al implementar una funcionalidad, escribe simultáneamente las pruebas unitarias asociadas. Se establece un mínimo inicial de cobertura del 15 % para el primer piloto, con incremento progresivo. Se integra SonarQube al pipeline de GitLab para análisis estático automático de calidad y seguridad del código con cada push.

Adición clave: se formaliza la revisión de código entre pares (peer review) como requisito antes de abrir un Merge Request, eliminando la práctica actual de código no revisado.

Fase 4 - Pruebas (rediseño)

Esta es la fase con mayor transformación respecto al AS-IS. Se definen cuatro tipos de pruebas formales con responsables y frecuencias claras:

Tipo de prueba	Tecnología aplicable	Herramienta	Frecuencia de ejecución	Responsable
Pruebas unitarias (Python)	Django, FastAPI	Pytest	En cada push a GitLab (CI automático)	Desarrollador
Pruebas unitarias (JavaScript/TypeScript)	Node.js, React, Angular	Jest	En cada push a GitLab (CI automático)	Desarrollador
Pruebas unitarias (Java)	Spring Boot	JUnit 5 + Mockito	En cada push a GitLab (CI automático)	Desarrollador
Análisis estático (SAST)	Todas las tecnologías	SonarQube	En cada push a GitLab (CI automático)	GitLab CI/CD
Pruebas de integración de APIs	Todas las tecnologías	Postman + Newman	En cada push a GitLab (CI automático)	GitLab CI/CD
Pruebas End-to-End (E2E) críticas	Frontend (Angular, React)	Playwright	Una vez al día (ejecución nocturna) y antes de cada despliegue a staging	GitLab CI/CD

Tabla 3.11: Tipos de prueba, herramientas, frecuencias y responsables en el proceso TO-BE. Fuente: elaboración propia.

Las pruebas de integración de servicios y APIs se formalizan mediante Newman, el cliente CLI de Postman, permitiendo ejecutar las colecciones existentes del equipo directamente desde el pipeline de GitLab sin intervención manual. Esto representa una transición natural para el equipo, dado que Postman ya era utilizado de forma manual en el AS-IS, reduciendo la curva de aprendizaje asociada a esta incorporación.

Los defectos encontrados se registran automáticamente como Issues en GitLab, vinculados al Merge Request o commit que los originó. Esto elimina la trazabilidad nula del estado actual y centraliza la gestión de defectos en la misma plataforma que ya usa el equipo.

La selección de herramientas de pruebas unitarias se realizó en función del ecosistema tecnológico de cada célula de desarrollo. La Célula 1, con enfoque en Django y FastAPI, utiliza pytest como framework principal. La Célula 2, orientada a tecnologías JavaScript y Java, utiliza Jest y JUnit 5 respectivamente. Esta diferenciación garantiza que cada herramienta sea nativa al ecosistema correspondiente, reduciendo la curva de aprendizaje y facilitando la integración con el pipeline de GitLab.

Fase 5 — Despliegue a pruebas (con QA)

Se formaliza el despliegue al ambiente de staging mediante un checklist obligatorio de liberación que incluye: verificación de versión del release, revisión del changelog y confirmación de que el ambiente de pruebas replica el esquema de producción. El QA Champion es responsable de ejecutar

y aprobar este checklist antes de que el ambiente quede disponible. A diferencia del AS-IS, el despliegue no ocurre de forma ad hoc: existe un momento formal de apertura del ambiente, trazable y registrado en GitLab.

Fase 6 – Validación con el usuario (con QA)

Una vez abierto el ambiente de staging, el líder del proceso valida que la funcionalidad desarrollada responda a la necesidad planteada. Para esta validación el equipo entrega una matriz de observaciones con los criterios de aceptación definidos desde la Fase 2, de modo que el usuario evalúa contra condiciones acordadas y no contra expectativas implícitas. Los ajustes identificados en esta fase se registran como Issues en GitLab con prioridad definida, evitando que ingresen a producción sin trazabilidad.

Fase 7 – Despliegue a producción (con QA)

Con la validación del usuario aprobada, se ejecuta el Quality Gate formal como condición obligatoria para autorizar el despliegue a producción. Este gate verifica automáticamente tres condiciones: cobertura mínima de pruebas unitarias alcanzada (15 % inicial), cero vulnerabilidades críticas en SonarQube, y pruebas de extremo a extremo (End-to-End, E2E) ejecutadas sin errores en staging. Si alguna condición no se cumple, el Merge Request queda bloqueado automáticamente y el desbloqueo requiere aprobación explícita del Líder del Subproceso con registro de la justificación.

Fase 8 – Mantenimiento y soporte (con QA)

Se establece que cualquier corrección de un bug en producción debe ir acompañada de un caso de prueba de regresión que reproduzca el error y verifique su corrección. Esto garantiza que el mismo bug no reaparezca en versiones futuras, transformando cada incidente en producción en un activo de pruebas.

Se implementa además un repositorio centralizado de casos de prueba en GitLab, documentado como "pruebas como documentación", accesible para todo el equipo y actualizable como parte del proceso rutinario de desarrollo.

3.3.3. Políticas y Criterios del Proceso

La siguiente tabla sintetiza las transformaciones principales introducidas por el proceso TO-BE en respuesta a las brechas identificadas en el diagnóstico:

Política 1: Definition of Done (DoD):

Una historia de usuario o requerimiento se considera terminado (Done) únicamente cuando se cumplen todas las condiciones siguientes:

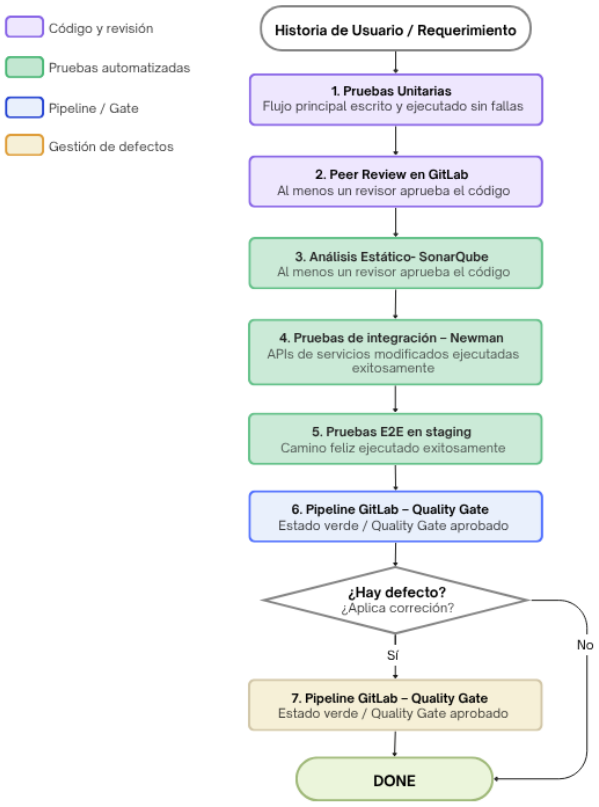


Figura 3.7: Flujo de calidad – Definition of Done. Fuente: Elaboración propia

1. Peer review en GitLab: El código ha sido revisado por al menos un par (peer review aprobado en GitLab).

2. Pruebas de integración con Newman: Las pruebas de integración de APIs asociadas a los servicios modificados han sido ejecutadas exitosamente mediante Newman en el pipeline de GitLab.

3. Pruebas unitarias: Las pruebas unitarias asociadas al flujo principal han sido escritas y ejecutadas sin fallas.

4. Análisis estático SonarQube: El análisis estático de SonarQube no reporta vulnerabilidades de seguridad críticas.

5. Pruebas E2E en staging: Las pruebas E2E del camino feliz han sido ejecutadas exitosamente en el entorno de staging.

6. Pipeline GitLab – Quality Gate: El pipeline de GitLab reporta estado verde (Quality Gate aprobado).

7. Registro y cierre de defecto (si aplica): El defecto, si aplica, ha sido registrado y cerrado en GitLab Issues con trazabilidad al commit de corrección.

Política 2: Gestión de fallas del pipeline:
Cuando una prueba automatizada falla en el pipeline de GitLab, se aplica el siguiente protocolo:

Severidad de la Falla	**Acción Inmediata**	**Responsable**
Falla en prueba unitaria o análisis estático	Bloqueo automático del Merge Request. El desarrollador corrige antes de continuar.	Desarrollador
Falla en prueba E2E en rama de desarrollo	Notificación al QA Champion. El equipo analiza si es una falla del código o del entorno de pruebas.	QA Champion
Falla en prueba E2E antes de despliegue a producción	Bloqueo del despliegue. Requiere aprobación explícita del Líder del Subproceso con registro de justificación para proceder.	Líder del subproceso
Falla en prueba de integración de APIs (Newman)	Notificación al QA Champion. Se analiza si la falla es del servicio o de la colección de Postman.	QA Champion

Tabla 3.12: Protocolo de gestión de fallas en el pipeline de CI/CD. Fuente: elaboración propia

Política 3: Registro y trazabilidad de defectos:
Todo defecto identificado, ya sea por las pruebas automatizadas, por revisión de código o por el usuario, debe registrarse como un Issue en GitLab con los siguientes campos mínimos:

1. Descripción del comportamiento observado.

2. Comportamiento esperado.

3. Pasos para reproducir, módulo o funcionalidad afectada.

4. Prioridad (crítica / alta / media / baja).

5. Estado (abierto / en progreso / cerrado).

El Issue debe vincularse al Merge Request o commit que lo originó para garantizar la trazabilidad completa.

Política 4: Integración del QA en el ciclo de desarrollo

Las actividades de aseguramiento de calidad se integran en cada ciclo de desarrollo de software, independientemente del marco metodológico vigente en el equipo. Dado que el diagnóstico evidenció que la organización opera en la práctica bajo un modelo predominantemente secuencial, el proceso TO-BE se diseña para adaptarse al flujo real de trabajo, sin requerir la adopción formal de metodologías ágiles como Scrum.

No obstante, el proceso incorpora prácticas comúnmente asociadas a enfoques ágiles, como la definición de criterios de aceptación, ciclos iterativos de entrega y criterios de finalización (Definition of Done), las cuales se implementan como mecanismos de control de calidad y no como parte de un marco metodológico específico.

En este sentido, el TO-BE es metodológicamente agnóstico y prioriza la incorporación progresiva de prácticas de aseguramiento de calidad sobre la adopción formal de un framework ágil, permitiendo que la organización evolucione hacia modelos más maduros de desarrollo de software de manera gradual.

3.3.4. Síntesis del Proceso TO-BE y Comparación con el AS-IS

La siguiente tabla sintetiza las transformaciones principales introducidas por el proceso TO-BE en respuesta a las brechas identificadas en el diagnóstico:

Tabla 3.13: Comparación del estado AS-IS vs. TO-BE en el proceso de aseguramiento de calidad. Fuente: elaboración propia

Dimensión	**Estado AS-IS**	**Estado TO-BE propuesto**
Inicio de las pruebas	Fase 4 (después del desarrollo)	Fase 2 (desde el análisis, criterios de aceptación testeables)
Tipo de pruebas	Solo funcionales manuales	Unitarias + análisis estático + E2E automatizadas
Responsable de pruebas	Único responsable informal	Equipo completo, supervisado por QA Champion
Ejecución de pruebas	Manual, bajo demanda	Automática en cada push y antes de cada despliegue
Bloqueo de despliegue	Ninguno (decisión informal)	Quality Gate automático con tres condiciones verificables
Registro de defectos	Inexistente	GitLab Issues con trazabilidad al código fuente
Cobertura automatizada	0 %	Mínimo 15 % inicial con escalamiento progresivo

Continúa en la siguiente página...

Tabla 3.13 – Continuación de la página anterior

Dimensión	Estado AS-IS	Estado TO-BE propuesto
Gestión del conocimiento	Conocimiento tácito individual	Repositorio de pruebas + Wiki técnica + onboarding documentado
Integración con Scrum	Pruebas fuera del ciclo; modelo operativo en cascada pese a declarar Scrum	QA integrado al ciclo de entrega real, sin dependencia de Scrum; sienta las bases para una adopción ágil futura

El proceso TO-BE no aborda la reducción de la deuda técnica acumulada, dado que ese objetivo excede el alcance del presente proyecto. Sin embargo, la exigencia de peer review y análisis estático con SonarQube desde la Fase 3 contribuirá a evitar su incremento en el código nuevo.

El proceso TO-BE propuesto constituye una transformación viable y gradual para la EPSI AIC. Su viabilidad se sustenta en tres factores habilitantes identificados en el diagnóstico: la disposición organizacional positiva de todos los niveles, la infraestructura tecnológica existente (GitLab, Docker, ambientes de prueba) y el apoyo explícito de la alta gerencia. La gradualidad del proceso, que inicia con una cobertura mínima del 15 % y escala progresivamente, garantiza que el equipo pueda adoptarlo sin comprometer su velocidad de respuesta a los requerimientos misionales de la EPSI.

Es importante señalar que el proceso TO-BE no condiciona su implementación a la adopción de Scrum, dado que el diagnóstico evidenció que la organización opera en la práctica bajo un modelo secuencial, con una brecha significativa entre la metodología declarada y la operación real. En consecuencia, el QA se integra al ciclo de entrega existente como punto de partida, sentando las bases organizacionales y técnicas que en el futuro podrían facilitar una transición real hacia prácticas ágiles efectivas.

3.4. Estructura Organizacional y Plan de Transición

3.4.1. Estructuras Organizacionales Evaluadas

La definición de la estructura organizacional para el proceso de aseguramiento de calidad de la EPSI AIC requirió una evaluación fundamentada en la literatura sobre diseño organizacional, disciplina presentada en la sección 2.1.1.11. Mintzberg (Mintzberg, 1979) identificó varios tipos de estructuras organizacionales: funcional, divisional, matricial y adhocracia, entre otras. Cada una con características específicas de coordinación, especialización y distribución del conocimiento. En el contexto de equipos pequeños de desarrollo de software con recursos limitados y baja madurez en procesos de *testing*, la literatura reduce las alternativas relevantes a dos configuraciones aplicables: la estructura funcional con *QA* centralizado y la estructura distribuida con *QA* integrado al equipo de desarrollo (Leite et al., 2021). Ambas fueron evaluadas frente a las condiciones objetivas identificadas en el diagnóstico de la EPSI AIC: un equipo de ocho personas distribuidas en dos células,

alta rotación de personal cada dos años, ausencia de un rol exclusivo de *QA* y nivel de madurez TMMi Nivel 1 confirmado.

Alternativa A — Estructura funcional con QA centralizado

En este modelo, el aseguramiento de calidad se concentra en un rol exclusivo y dedicado; un *QA Manager*, que lidera un equipo de analistas de pruebas independiente del equipo de desarrollo. Este enfoque replica el modelo funcional descrito por Mintzberg (Mintzberg, 1979), donde cada función especializada opera como un departamento independiente con autoridad y criterio propio. Pressman y Maxim (Pressman and Maxim, 2020) describen este modelo como el enfoque tradicional de control de calidad en organizaciones de software, donde la separación entre quien desarrolla y quien prueba garantiza objetividad en la detección de defectos. Sus ventajas principales son la especialización técnica profunda, el criterio objetivo e independiente sobre la calidad del producto y la visibilidad directa de las métricas de calidad ante la dirección. Sin embargo, en el contexto de la EPSI AIC este modelo presenta desventajas determinantes: exige la contratación de al menos un rol exclusivo de *QA*, inversión que la organización no contempla en su estructura presupuestal actual. Más críticamente, concentra el conocimiento institucional sobre las pruebas en un actor específico, replicando exactamente el punto único de falla operacional identificado como la brecha más grave del diagnóstico *AS-IS*. La política de rotación de personal cada dos años agrava este riesgo de manera estructural, dado que cada ciclo implicaría una pérdida total de la capacidad de *QA* acumulada.

Alternativa B — Estructura distribuida con QA integrado al equipo

En este modelo, la responsabilidad del aseguramiento de calidad se distribuye entre todos los miembros del equipo de desarrollo, con un rol de coordinación operativa; el *QA Champion*, asumido por un desarrollador *senior* de cada célula. Este enfoque es coherente con la configuración de equipo multifuncional identificada por Leite et al. (Leite et al., 2021) como la más compatible con la integración sostenible de prácticas de calidad en equipos de desarrollo, y con las recomendaciones de Dustin et al. (Dustin et al., 1999) para organizaciones que inician la adopción de *QA* sin disponibilidad de roles exclusivos. Sus ventajas principales son la viabilidad sin presupuesto adicional, la distribución del conocimiento institucional entre varios actores, reduciendo la vulnerabilidad ante la rotación de personal y la mayor adopción cultural, dado que las pruebas son responsabilidad de quien construye el código. Como desventaja, la menor objetividad en la revisión se mitiga mediante el *peer review* obligatorio y el *Quality Gate* automatizado en GitLab, que actúan como controles independientes del criterio individual.

Decisión y justificación

La Tabla 3.14 sintetiza la evaluación comparativa de ambas alternativas frente a los criterios derivados del diagnóstico organizacional:

Viabilidad presupuestal	Baja — requiere rol adicional	Alta — usa capacidades existentes
Reducción del punto único de falla	Baja — concentra en *QA Manager*	Alta — distribuye en el equipo
Compatibilidad con madurez TMMi 1	Baja — exige capacidades no existentes	Alta — parte del estado actual
Sostenibilidad ante rotación	Baja — conocimiento en actor único	Alta — documentación viva y *onboarding*
Velocidad de adopción	Baja — cambio cultural profundo	Media-alta — integración gradual
Independencia de revisión	Alta	Media — mitigada por *Quality Gate*

Tabla 3.14: Comparación de alternativas de estructura organizacional para el proceso de aseguramiento de calidad. Fuente: Elaboración propia.

La evaluación evidencia que la Alternativa B responde de manera más adecuada a las condiciones organizacionales de la EPSI AIC en su estado actual. La selección no descarta la Alternativa A como modelo de evolución futura: en contextos de mayor madurez o con disponibilidad presupuestal para un rol exclusivo de *QA*, el modelo centralizado representaría una evolución natural recomendable, constituyendo una línea de trabajo futuro para la organización.

3.4.2. Roles y Responsabilidades

La estructura organizacional propuesta para el proceso de aseguramiento de calidad de la EPSI AIC responde a las restricciones identificadas en el diagnóstico: un equipo de desarrollo de ocho personas distribuidas en dos células, alta rotación de personal cada dos años y ausencia de un rol exclusivo de QA. En consecuencia, se adoptó un modelo de calidad distribuida, en el que la responsabilidad del aseguramiento de calidad recae sobre todo el equipo de desarrollo bajo la coordinación de un rol transversal denominado QA Champion. Este modelo evita la dependencia de un único responsable y garantiza que las prácticas de QA se institucionalicen como parte del trabajo cotidiano de desarrollo, y no como una actividad aislada o posterior al mismo.

Figura 3.8: Estructura organizacional propuesta para el proceso de QA. Fuente: Elaboración propia

La estructura se conformó por cuatro roles, articulados con la organización existente de la EPSI AIC: el Líder de Gestión TICs (rol táctico, encargado de la sostenibilidad estratégica del proceso), el Líder del Subproceso de Desarrollo (líder técnico, responsable último del QA en el nivel técnico), el QA Champion (coordinador operativo que guía al equipo en la escritura y mantenimiento de las pruebas) y los Desarrolladores (responsables de escribir y ejecutar las pruebas junto con el código). El modelo se complementa con dos mecanismos de gestión del conocimiento ante la rotación de personal: el repositorio de pruebas como documentación viva y una guía de onboarding técnico en GitLab.

El detalle de las responsabilidades específicas de cada rol, el perfil recomendado para el QA Champion y los mecanismos de transferencia de conocimiento se presentan en el Anexo C

3.4.3. Matriz RACI

La matriz RACI presentada a continuación definió de manera explícita el nivel de participación de cada rol en las actividades clave del proceso de aseguramiento de calidad propuesto. Las categorías utilizadas son:

- **R (Responsable):** Ejecuta la actividad.
- **A (Aprobador):** Rinde cuentas por el resultado.
- **C (Consultado):** Aporta criterio antes de ejecutar.
- **I (Informado):** Recibe notificación del resultado.

Actividad del proceso QA	Líder TICs	Líder Subproceso DS	QA Champion	Desarrolladores
Definición de criterios de aceptación (Given-When-Then)	I	A	C	R
Escritura de pruebas unitarias	I	A	C	R
Revisión de código entre pares (peer review)	I	A	C	R
Análisis estático con SonarQube	I	A	R	R
Ejecución de pruebas de integración (Newman)	I	A	R	R
Ejecución de pruebas E2E (Playwright)	I	A	R	C
Aprobación del checklist de liberación a staging	I	C	R	I
Aprobación del Quality Gate para producción	C	A	R	I
Aprobación de excepciones al Quality Gate	A	R	C	I
Registro de defectos en GitLab Issues	I	A	C	R
Reporte periódico de indicadores de calidad	A	R	C	I
Mantenimiento del repositorio de casos de prueba	I	C	R	C
Actualización de la wiki técnica y onboarding	I	C	R	C
Gestión del conocimiento ante rotación de personal	A	C	R	C

Figura 3.9: Matriz RACI del proceso de aseguramiento de calidad propuesto para la EPSI AIC. Fuente: Elaboración propia

3.4.4. Plan de Transición al Modelo Propuesto

El plan de transición describió el proceso mediante el cual la EPSI AIC evolucionó desde el estado actual de madurez, caracterizado por validaciones manuales informales y ausencia total de automatización, hacia el modelo de aseguramiento de calidad propuesto. El plan se estructuró en cuatro fases secuenciales alineadas a los tres ejes de la metodología DSR que enmarcó el proyecto, con inicio en enero de 2026 y cierre en mayo de 2026.

Es importante señalar que, dado el carácter aplicado y piloto del proyecto, el plan de transición no buscó alcanzar una madurez plena en todos los procesos de QA durante este periodo, sino estableció las bases organizacionales, técnicas y culturales que permitieran el escalamiento sostenible del proceso en fases futuras.

Tabla 3.15: Plan de transición al modelo de aseguramiento de calidad propuesto. Fuente: elaboración propia

Fase	**Período**	**Actividades principales**
Fase A: Diagnóstico y diseño del proceso de QA	Enero - Febrero 2026	Entrevistas cualitativas, análisis documental, evaluación del proceso de pruebas, diseño del proceso TO-BE, definición de estructura organizacional y matriz RACI
Continúa en la siguiente página...		

Tabla 3.15 – Continuación de la página anterior

Fase	Período	Actividades principales
Fase B: Priorización y diseño de casos de prueba	Febrero - Marzo 2026	Definición de criterios de priorización basados en riesgo y criticidad, aplicación de la matriz de priorización, selección de técnicas de caja negra ISTQB, diseño formal de casos de prueba con trazabilidad completa
Fase C: Implementación de la automatización	Marzo - Abril 2026	Selección del stack tecnológico, diseño de la arquitectura del marco de automatización, implementación piloto en módulos misionales priorizados, configuración del pipeline de CI/CD en GitLab, documentación técnica del marco
Fase D: Evaluación de impacto	Abril - Mayo 2026	Medición de indicadores de proceso y producto, análisis comparativo antes/después, evaluación cualitativa del equipo, generación de recomendaciones para escalamiento futuro

3.4.4.1. Condiciones habilitantes para la transición

El diagnóstico identificó tres factores que hicieron viable la transición en el plazo establecido. En primer lugar, la disposición organizacional positiva de todos los niveles jerárquicos entrevistados, quienes reconocieron la urgencia del cambio y expresaron apertura explícita hacia la adopción del proceso propuesto. En segundo lugar, la infraestructura tecnológica existente: GitLab, Docker y ambientes de prueba diferenciados, que permitió implementar el marco de automatización sin inversión adicional en herramientas. En tercer lugar, el apoyo explícito de la alta gerencia, que garantizó la provisión de tiempo y recursos necesarios para que el equipo de desarrollo pudiera adoptar las nuevas prácticas sin comprometer la atención a los requerimientos misionales en curso.

3.4.4.2. Riesgos y medidas de mitigación

La transición enfrentó tres riesgos principales identificados durante el diagnóstico. El primero fue la resistencia al cambio por parte del equipo ante la incorporación de nuevas prácticas en un contexto de alta carga operativa; este riesgo se mitigó mediante la adopción gradual del proceso, iniciando con metas de cobertura modestas del 15 % que se escalaron progresivamente conforme el equipo ganó confianza en las nuevas prácticas.

El segundo riesgo fue la pérdida de continuidad por rotación de personal durante el periodo de implementación; se mitigó mediante el repositorio centralizado de pruebas como documentación viva y la guía de onboarding técnico en GitLab, ambos mecanismos diseñados para garantizar la transferencia de conocimiento institucional independientemente de los cambios en el equipo.

El tercero fue la dependencia de la disponibilidad del ambiente de pruebas para la ejecución

del piloto; se mitigó mediante la coordinación formal con el subproceso de infraestructura para garantizar la estabilidad del ambiente de staging durante las fases C y D del proyecto.

El presente capítulo documentó el proceso completo correspondiente al primer objetivo específico del proyecto: el diagnóstico del estado actual de las prácticas de aseguramiento de calidad en la EPSI AIC y el diseño del proceso formal que respondió a las brechas identificadas. A partir de la triangulación metodológica de entrevistas cualitativas, análisis documental y evaluación de madurez del proceso de pruebas, se construyó un diagnóstico integral que evidenció una madurez muy baja en los procesos de testing, caracterizada por la ausencia total de automatización, la dependencia de validaciones manuales por un único responsable y la carencia de métricas formales de calidad.

En respuesta a este diagnóstico, se diseñó un proceso TO-BE fundamentado en el principio de Shift Left Testing, con actividades de QA integradas en cada fase del ciclo de desarrollo, una estructura organizacional de calidad distribuida supervisada por el rol de QA Champion, una matriz RACI que formalizó las responsabilidades de cada actor y un plan de transición que orientó la evolución progresiva hacia el modelo propuesto. Los elementos diseñados en este capítulo constituyeron la base sobre la cual se desarrollaron los capítulos siguientes, que abordaron la priorización de funcionalidades críticas, el diseño formal de casos de prueba y la implementación del marco de automatización.

3.5. Priorización y Diseño de Casos de Prueba

La implementación de un marco de pruebas automatizadas no puede abordarse de manera indiscriminada sobre la totalidad de los sistemas de la organización; especialmente en contextos en los cuales los recursos técnicos son limitados y se presenta baja madurez en los procesos de testing tal como se identificó en la EPSI AIC. Iniciar procesos de automatización sin establecer criterios claros puede llevar a invertir esfuerzo en funcionalidades de bajo impacto mientras que los módulos más críticos que intervienen en la continuidad del servicio permanecen sin cobertura. Debido a esto, antes de escribir pruebas automatizadas es necesario establecer el orden de priorización de las pruebas y de qué manera se deben diseñar estas pruebas.

En cuanto al desarrollo del paradigma *Design Science Research* (DSR), esta fase corresponde a la etapa de **Diseño del Tratamiento**, y se desarrolló a partir de los insumos disponibles del diagnóstico: los hallazgos cualitativos de las entrevistas realizadas, la línea base referencial del estado actual presentada en la Tabla 3.8 y la caracterización de los sistemas de información misionales establecidos en la sección 3.2.1.

Sobre esa base se estructuró el trabajo en dos frentes secuenciales:

1. **Construcción de una matriz de priorización:** Evaluación de los sistemas de desarrollo propio de la EPSI AIC según criterios de riesgo, criticidad normativa y frecuencia de uso.

2. **Diseño formal de casos de prueba:** Aplicación de técnicas de caja negra del ISTQB sobre las funcionalidades priorizadas, seleccionadas según el tipo de lógica que valida cada funcionalidad.

El resultado de esta fase constituye el insumo directo de la Fase C, sin funcionalidades priorizadas y sin casos de prueba diseñados, la implementación de la automatización carecería de dirección y sustento metodológico.

3.5.1. Criterios de Priorización

La selección de los criterios de priorización se fundamenta en el modelo bidimensional de pruebas basadas en riesgo establecido por el ISTQB (International Software Testing Qualifications Board, 2023), el cual determina el nivel de riesgo de un componente de software a partir de la probabilidad de ocurrencia de una falla y el impacto que esta tendría sobre el negocio. En coherencia con ese marco teórico y con los hallazgos del diagnóstico organizacional presentados en la sección 3.2, se definieron tres criterios que operacionalizan ambas dimensiones en el contexto específico de la EPSI AIC.

Criterio 1 — Riesgo para la continuidad del servicio (peso: 40 %): Corresponde a la dimensión de *impacto operativo*. Este criterio evalúa en qué medida una falla en el sistema comprometería la prestación de servicios de salud a las comunidades indígenas atendidas por la EPSI AIC. Las entrevistas realizadas en febrero de 2026 revelaron de forma consistente en todos los niveles organizacionales que la continuidad operativa constituye la preocupación dominante, dado que los incidentes en producción generan interrupciones directas en procesos misionales como la gestión de afiliaciones, la facturación y el seguimiento a rutas de salud. Por esta razón, este criterio recibe el mayor peso relativo.

Criterio 2 — Criticidad normativa según la Resolución 866 de 2021 (peso: 35 %): Corresponde a la dimensión de *impacto legal y regulatorio*. Evalúa las consecuencias que tendría una falla del sistema frente al cumplimiento de obligaciones normativas del Ministerio de Salud y Protección Social, particularmente en materia de confidencialidad, integridad, disponibilidad y trazabilidad. Como señalan (Knight and Wika, 1995), los sistemas médicos son intrínsecamente críticos para la seguridad, lo que exige considerar las consecuencias normativas como una dimensión independiente del impacto operativo.

Criterio 3 — Frecuencia de uso (peso: 25 %): Corresponde a la dimensión de *probabilidad de ocurrencia*. Evalúa el volumen de transacciones y la regularidad de interacción del usuario; a mayor frecuencia, mayor exposición del sistema y mayor probabilidad de que un defecto latente se manifieste. Recibe el menor peso debido a que, en el sector salud, una falla de baja frecuencia puede tener consecuencias igual de críticas que una frecuente, limitando su capacidad de discriminación por sí solo.

La ponderación asignada a cada criterio no se estableció de manera apriorística, sino que emergió de los hallazgos del diagnóstico organizacional. Siguiendo a Yin (2018), en contextos donde el conocimiento reside principalmente en las personas, la evidencia cualitativa obtenida mediante entrevistas a actores clave constituye una fuente válida para informar estas decisiones de diseño.

3.5.2. Aplicación de la Matriz

Con los criterios y pesos definidos en la sección 3.5.1, se procedió a evaluar los tres sistemas de desarrollo propio de la EPSI AIC que constituyen el alcance de intervención del presente proyecto: el Sistema de Gestión de PQRSD, el Módulo de Referencia 24/7 y el Sistema de Gestión del Buen Vivir.

La evaluación se realizó a partir de los hallazgos cualitativos de las entrevistas realizadas en febrero de 2026, la caracterización de los sistemas presentada en la sección 3.2.1.3 y el conocimiento operacional validado con los actores clave del nivel táctico y operativo de la organización.

3.5.2.1. Escala de valoración

Para garantizar la objetividad y comparabilidad de la evaluación, cada criterio se valoró mediante una escala ordinal de tres niveles:

Valor	Nivel	Descripción
3	Alto	El sistema presenta alta exposición al criterio evaluado
2	Medio	El sistema presenta exposición moderada al criterio evaluado
1	Bajo	El sistema presenta baja exposición al criterio evaluado

Tabla 3.16: Escala de valoración. Fuente: elaboración propia.

3.5.2.2. Fórmula de cálculo

El puntaje final de cada sistema se calculó mediante la siguiente expresión ponderada, coherente con la fórmula de agregación utilizada en la evaluación TMMi presentada en la sección 3.1.2.3:

$$Puntaje\ Final = (C1 \times 0{,}40) + (C2 \times 0{,}35) + (C3 \times 0{,}25) \tag{3.1}$$

Donde:

- C1: Riesgo de continuidad del servicio
- C2: Criticidad normativa
- C3: Frecuencia de uso

3.5.2.3. Resultados de la matriz

Sistema	C1 (40 %)	C2 (35 %)	C3 (25 %)	Puntaje Final
PQRSD	$3 \times 0{,}40 = 1{,}20$	$3 \times 0{,}35 = 1{,}05$	$3 \times 0{,}25 = 0{,}75$	3.00
Referencia 24/7	$3 \times 0{,}40 = 1{,}20$	$3 \times 0{,}35 = 1{,}05$	$3 \times 0{,}25 = 0{,}75$	3.00
Gestión del Buen Vivir	$2 \times 0{,}40 = 0{,}80$	$2 \times 0{,}35 = 0{,}70$	$2 \times 0{,}25 = 0{,}50$	2.00

Tabla 3.17: Resultados de la matriz de priorización. Fuente: elaboración propia.

El Sistema de Gestión de PQRSD y el Módulo de Referencia 24/7 obtuvieron puntajes idénticos de 3.00, lo que requirió la aplicación de un criterio de desempate. Dado que la escala ordinal utilizada no permite discriminar entre sistemas con igual puntuación en los tres criterios, se incorporó como criterio de desempate la naturaleza operacional del sistema y su impacto directo sobre la seguridad del paciente.

El Módulo de Referencia 24/7 gestiona el traslado oportuno de pacientes entre niveles de atención hospitalaria. A diferencia del Sistema de PQRSD, cuya falla genera consecuencias críticas pero con un margen de gestión posterior, una falla o demora en el Módulo de Referencia 24/7 puede comprometer directamente la seguridad y la vida del paciente, dado que las decisiones de traslado son *time-sensitive* y no admiten interrupciones sin consecuencias clínicas inmediatas.

Este atributo lo clasifica como un sistema de seguridad crítica en el sentido definido por Knight y Wika (Knight and Wika, 1995), quienes señalan que en sistemas médicos donde la oportunidad de la respuesta es determinante, los riesgos de falla trascienden lo operativo y adquieren una dimensión clínica que debe ser prioritaria en cualquier estrategia de aseguramiento de calidad.

En consecuencia, el Módulo de Referencia 24/7 fue seleccionado como el sistema sobre el cual se desarrolló el piloto de implementación de la automatización en la Fase C del proyecto.

3.5.2.4. Orden final de priorización

Posición	Sistema	Puntaje Final	Justificación
1	Referencia 24/7	3.00	Empate resuelto por seguridad crítica del paciente
2	PQRSD	3.00	Alto impacto operativo y normativo
3	Gestión del Buen Vivir	2.00	Impacto moderado en los tres criterios

Tabla 3.18: Orden de priorización. Fuente: elaboración propia.

Este orden de priorización constituyó el insumo directo para el diseño formal de casos de prueba presentado en la sección 3.5.3, donde las funcionalidades del Módulo de Referencia 24/7 fueron las primeras en ser abordadas mediante las técnicas de caja negra del ISTQB seleccionadas para el proyecto.

3.5.3. Selección de técnicas de caja negra

La selección de las técnicas de diseño de casos de prueba se realizó siguiendo los lineamientos del ISTQB Advanced Level Test Analyst Syllabus v4.0 (Hamburg and Roman, 2025), descrito en la sección 2.1 del presente documento, el cual establece que la elección de la técnica más adecuada depende del tipo de lógica que valida cada funcionalidad y no de una aplicación uniforme sobre todo el sistema.

En coherencia con este principio, se analizaron en detalle las cinco funcionalidades del Módulo de Referencia 24/7, identificando para cada una sus subfuncionalidades y el tipo de lógica subyacente.

Funcionalidad	**Subfuncionalidad**	**Técnica**	**Justificación**
Búsqueda y validación de afiliados	Búsqueda por documento de identidad, validación de estado, restricción de caracteres, codificación MS/AS	Particiones de Equivalencia	Presenta clases discretas claramente diferenciables: afiliado encontrado/no encontrado, activo/inactivo, formato válido/inválido y codificaciones especiales para menores y adultos sin identificación.
Registro de urgencias	Datos del prestador, datos complementarios, soporte	Particiones de Equivalencia	Los campos presentan dominios de entrada definidos donde los valores válidos e inválidos pueden agruparse en clases equivalentes que producen el mismo comportamiento del sistema.
Clasificación TRIAGE	Condición y destino, motivo	Tablas de Decisión	La combinación de condiciones clínicas determina diferentes resultados y rutas del sistema, requiriendo verificación sistemática de todas las combinaciones relevantes.
Gestión de referencias	Búsqueda de prestador, código de aceptación, fecha y hora de disponibilidad de cupo	Tablas de Decisión	Múltiples condiciones combinadas determinan si la referencia puede concretarse, requiriendo verificación de todas las combinaciones posibles entre prestador disponible, código válido y fecha vigente.
Seguimiento y bitácora	Filtros (Estado, Fecha inicio, Fecha fin), visualización de reportes	Arreglos Ortogonales	La combinación de tres filtros independientes genera un espacio de variaciones que los arreglos ortogonales permiten cubrir eficientemente con un subconjunto representativo.
Acciones operativas	Gestionar traslado: tipo de transporte (terrestre, fluvial, aéreo, multimodal) y modalidad (básico, medicalizado)	Tablas de Decisión	La combinación de tipos de transporte y modalidades determina diferentes configuraciones del traslado que deben verificarse sistemáticamente.
	Ingreso a destino: fecha y hora de ingreso, comentarios	Particiones de Equivalencia	Los campos presentan dominios de entrada con valores válidos e inválidos agrupables en clases equivalentes.

Continúa de la página anterior			
Funcionalidad	**Subfuncionalidad**	**Técnica**	**Justificación**
	Cancelar remisión: 13 motivos de cancelación	Tablas de Decisión	Los motivos de cancelación generan consecuencias clínicas y operativas diferenciadas que deben verificarse de manera sistemática.

Tabla 3.19: Selección de técnicas de caja negra por funcionalidad y subfuncionalidad. Fuente: elaboración propia.

Esta asignación garantiza que cada técnica se aplique donde genera mayor valor en términos de cobertura y eficiencia. Las particiones de equivalencia reducen el número de casos necesarios para cubrir dominios de entrada al agrupar valores con comportamiento equivalente. Las tablas de decisión aseguran que todas las combinaciones de condiciones clínicas y operativas relevantes sean verificadas sistemáticamente. Los arreglos ortogonales permiten una cobertura eficiente de combinaciones de parámetros de filtrado sin necesidad de ejecutar la totalidad de las variaciones posibles, lo cual sería inabordable en términos de esfuerzo de prueba.

3.5.4. Diseño de Casos de Prueba

El diseño formal de los casos de prueba se realizó aplicando las técnicas de caja negra seleccionadas en la sección 3.5.3 para cada funcionalidad del Módulo de Referencia 24/7, siguiendo la estructura documental establecida por el estándar ISO/IEC/IEEE 29119-3 (ISO/IEC/IEEE, 2013), el cual define los campos mínimos que debe contener un caso de prueba para garantizar su trazabilidad, reproducibilidad y utilidad en contextos de automatización.

Para cada caso de prueba se documentaron los siguientes campos: identificador único, nombre descriptivo, funcionalidad y subfuncionalidad asociada, técnica de diseño aplicada, precondiciones del entorno, datos de entrada, pasos de ejecución, resultado esperado y poscondición. La separación explícita entre resultado esperado y poscondición responde a una distinción fundamental establecida en el estándar ISO/IEC/IEEE 29119-3 (ISO/IEC/IEEE, 2013, p. 4.6): el resultado esperado describe la respuesta inmediata y visible del sistema ante los pasos ejecutados, mientras que la poscondición describe el estado persistente en que debe quedar el entorno o la base de datos tras la ejecución de la prueba. Esta distinción es especialmente relevante en sistemas con operaciones críticas como el Módulo de Referencia 24/7, donde el sistema puede mostrar un mensaje de confirmación correcto pero fallar al registrar el evento en la base de datos, situación que solo es detectable si la poscondición está explícitamente definida y verificada.

Adicionalmente, para cada técnica de diseño se elaboró una tabla de diseño que documenta el proceso de derivación de los casos de prueba a partir de la técnica aplicada, siguiendo los lineamientos del ISTQB Advanced Level Test Analyst Syllabus v4.0 (Hamburg and Roman, 2025). Para las particiones de equivalencia, la tabla de diseño incluye las columnas: variable o campo, clase de equivalencia, tipo de clase (válida o no válida), representante de la clase y referencia al caso de prueba que la cubre. Para las tablas de decisión, la tabla de diseño presenta los casos en columnas,

con una sección de condiciones en la parte superior y una sección de acciones en la parte inferior, siguiendo el formato formal establecido por el ISTQB. Para los arreglos ortogonales, la tabla de diseño documenta los factores, sus niveles y la matriz de combinaciones seleccionadas.

En total se diseñaron 48 casos de prueba distribuidos en ocho funcionalidades del módulo, como se resume en la Tabla 3.20:

Funcionalidad	**Técnica**	**Total Casos**
Búsqueda y validación de afiliados	Particiones de Equivalencia	5
Registro de urgencias	Particiones de Equivalencia	5
Registro de urgencias - TRIAGE	Tablas de Decisión	6
Gestión de referencias	Tablas de Decisión	6
Seguimiento y bitácora	Arreglos Ortogonales	9
Acciones operativas - Traslado	Tablas de Decisión	8
Acciones operativas - Ingreso Destino	Particiones de Equivalencia	4
Acciones operativas - Cancelar Remisión	Tablas de Decisión	5

Tabla 3.20: Resumen de casos de prueba diseñados por funcionalidad y técnica. Fuente: elaboración propia.

Es importante distinguir entre el diseño de casos de prueba y su automatización, dado que corresponden a dos actividades con alcances diferentes dentro del proyecto. El diseño cubre la totalidad de las funcionalidades del módulo con 48 casos, garantizando trazabilidad completa y cobertura formal de todos los flujos críticos. La automatización, en cambio, se circunscribió al piloto de implementación definido en la Fase C, para el cual se seleccionaron las funcionalidades de búsqueda y validación de afiliados y registro de urgencias correspondientes a los casos CP-REF-01 al CP-REF-05 y del CP-REF-07 al CP-REF-11.

La selección de estas funcionalidades para el piloto responde a su condición de flujo crítico de entrada del módulo: sin una búsqueda exitosa del afiliado no es posible registrar una urgencia, y sin una urgencia registrada con su clasificación TRIAGE correspondiente ninguna de las demás funcionalidades puede ejecutarse. Esta dependencia secuencial las convierte en el punto de mayor riesgo operativo del sistema, dado que concentran la mayor probabilidad de falla con el mayor impacto en cascada sobre todo el flujo clínico posterior. La automatización de las funcionalidades restantes constituye una línea de trabajo futuro que podrá desarrollarse de manera incremental sobre el marco implementado en este proyecto, siguiendo el principio de mejora incremental definido en la sección 3.3.1.

El diseño completo de los 48 casos de prueba, incluyendo las tablas de diseño por técnica y el detalle de cada caso con sus campos ISO 29119-3, se presenta en el Anexo A del presente documento.

3.6. Automatización

La implementación del marco de automatización de pruebas constituyó el tercer eje del proyecto dentro del paradigma DSR, correspondiente a la etapa de Diseño del Tratamiento en su dimensión técnica. Esta fase tomó como insumo directo los casos de prueba diseñados en la Fase B y los transformó en pruebas ejecutables integradas en el ciclo de desarrollo de la EPSI AIC. El trabajo se estructuró en cinco actividades secuenciales: selección del stack tecnológico, diseño de la arquitectura del marco, desarrollo del sistema de reportería, implementación piloto y generación de documentación técnica.

3.6.1. Marco Metodológico

El desarrollo de este objetivo se enmarcó en el eje de Diseño del Tratamiento del paradigma DSR, específicamente en la construcción del artefacto técnico central del proyecto: el marco de automatización de pruebas. A diferencia de la Fase A, cuyo enfoque fue predominantemente cualitativo y diagnóstico, esta fase adoptó un enfoque técnico-constructivo orientado a la producción de componentes de software funcionales, verificables y mantenibles.

La estrategia metodológica siguió un proceso de toma de decisiones basado en evidencia, donde cada decisión de diseño se fundamentó en los hallazgos del diagnóstico organizacional. Las restricciones identificadas en la Fase A: ausencia de presupuesto para licencias, equipo sin formación formal en QA, infraestructura basada en GitLab y Docker, y stack tecnológico heterogéneo con Python, JavaScript y Java; actuaron como criterios de selección que delimitaron el espacio de soluciones técnicas viables. Este principio de diseño orientado al contexto es coherente con las recomendaciones de (Dustin et al., 1999), quienes señalan que los marcos de automatización sostenibles son aquellos que se adaptan a las capacidades y restricciones reales de la organización, y no aquellos que imponen estándares externos desvinculados del contexto operativo.

El proceso de selección de herramientas se realizó mediante una matriz de decisión multicriterio que evaluó las alternativas disponibles para cada capa del stack tecnológico, considerando los cinco criterios establecidos en la metodología del proyecto: compatibilidad con la infraestructura existente, costo de adquisición y mantenimiento, curva de aprendizaje para un equipo sin experiencia previa en automatización, soporte de la comunidad y capacidad de integración con el pipeline de GitLab CI/CD. La implementación siguió un diseño modular basado en el patrón *Page Object Model* (POM), que separa la lógica de negocio de la interfaz de usuario, facilitando el mantenimiento y la escalabilidad del marco ante cambios en el sistema.

3.6.2. Selección de Herramientas

Las herramientas que conforman el stack tecnológico del marco de automatización fueron definidas en el proceso TO-BE presentado en la sección 3.3.2. La presente sección documenta la justificación formal de esa selección mediante una matriz de decisión multicriterio que evaluó cada herramienta frente a sus principales alternativas, considerando las restricciones organizacionales identificadas en el diagnóstico: ausencia de presupuesto para licencias, equipo sin formación previa

en automatización, infraestructura basada en GitLab y stack tecnológico heterogéneo con Python, JavaScript y Java.

Criterios de evaluación

Cada herramienta candidata fue evaluada en una escala de 1 a 5 en los siguientes criterios:

- **Compatibilidad:** grado de integración nativa con el stack tecnológico de la EPSI AIC.
- **Costo:** disponibilidad como herramienta de código abierto, dado que el diagnóstico confirmó la ausencia de presupuesto para licencias.
- **Curva de aprendizaje:** facilidad de adopción para un equipo sin formación previa en automatización.
- **Integración CI/CD:** capacidad de ejecución nativa dentro del pipeline de GitLab.
- **Soporte y comunidad:** actividad de la comunidad, frecuencia de actualizaciones y disponibilidad de documentación.

Pruebas Unitarias Backend (Pytest vs. Unittest)

Criterio	Pytest	Unittest
Compatibilidad con Django/FastAPI	5	4
Costo	5	5
Curva de aprendizaje	5	3
Integración GitLab CI/CD	5	4
Soporte y comunidad	5	4
Total	**25**	**20**

Tabla 3.21: Matriz de decisión - Pruebas Unitarias Backend, selección de herramientas del marco de automatización. Fuente: elaboración propia.

Pytest obtuvo la puntuación máxima en todos los criterios. Su integración nativa con Django y FastAPI, su sintaxis basada en funciones sin necesidad de clases heredadas y su amplia adopción en el ecosistema Python lo convierten en el estándar de facto para pruebas unitarias en este stack (Okken, 2022). A diferencia de *unittest*, que requiere estructuras de clases más verbosas, Pytest permite escribir pruebas con menor fricción, lo cual es determinante para un equipo que adopta estas prácticas por primera vez.

Pruebas de Integración de APIs (Newman vs. Karate)

Criterio	Newman	Karate
Compatibilidad con stack existente	5	3
Costo	5	5
Curva de aprendizaje	5	3
Integración GitLab CI/CD	5	4
Soporte y comunidad	5	3
Total	**25**	**18**

Tabla 3.22: Matriz de decisión - Pruebas de Integración de APIs, selección de herramientas del marco de automatización. Fuente: elaboración propia.

Newman fue seleccionado porque el equipo ya utilizaba Postman de manera manual para validar APIs durante el proceso AS-IS, como se documentó en la sección 3.2.1.4. Esta condición preexistente elimina la necesidad de rediseñar las colecciones de prueba existentes y reduce la curva de aprendizaje a prácticamente cero, dado que Newman es el cliente de línea de comandos oficial de Postman que permite ejecutar esas mismas colecciones directamente desde el pipeline de GitLab. Karate, aunque robusto, requiere el aprendizaje de un lenguaje o DSL nuevo, lo que representa una barrera significativa para el contexto organizacional.

Pruebas Unitarias Frontend (Vitest vs. Jest)

Criterio	Vitest	Jest
Compatibilidad con React-Vite	5	3
Costo	5	5
Curva de aprendizaje	5	4
Integración GitLab CI/CD	5	5
Soporte y comunidad	5	5
Total	**25**	**22**

Tabla 3.23: Matriz de decisión - Pruebas Unitarias Frontend, selección de herramientas del marco de automatización. Fuente: elaboración propia.

Vitest fue seleccionado por su compatibilidad nativa con React Vite, el *bundler* utilizado en el frontend de la EPSI AIC. A diferencia de Jest, que requiere configuración adicional para funcionar

con proyectos Vite, Vitest reutiliza directamente la configuración existente del proyecto, eliminando incompatibilidades entre el entorno de pruebas y el entorno de construcción (Vitest Team, 2024). Su API es intencionalmente compatible con Jest, lo que facilita una migración futura si fuera necesario.

Pruebas E2E (Playwright vs. Selenium vs. Cypress)

Criterio	Playwright	Selenium	Cypress
Compatibilidad con React/Angular	5	4	4
Costo	5	5	4
Curva de aprendizaje	5	2	4
Integración GitLab CI/CD	5	4	4
Soporte y comunidad	5	5	4
Total	**25**	**20**	**20**

Tabla 3.24: Matriz de decisión - Pruebas E2E, selección de herramientas del marco de automatización. Fuente: elaboración propia.

Playwright fue seleccionado por su soporte nativo para múltiples navegadores, su ejecución optimizada en modo *headless* para pipelines CI/CD y su capacidad para generar capturas de pantalla y trazas de ejecución integradas (Microsoft, 2024). Frente a Selenium, presenta una curva de aprendizaje significativamente menor y una API moderna basada en *async/await*. Frente a Cypress, no tiene restricciones de *same-origin* que pudieran afectar flujos con múltiples dominios o *iframes*, y su versión de código abierto no limita funcionalidades clave.

Análisis Estático (SonarQube vs. ESLint + Pylint)

Criterio	SonarQube	ESLint + Pylint
Compatibilidad con stack heterogéneo	5	3
Costo	4	5
Curva de aprendizaje	4	4
Integración GitLab CI/CD	5	4
Soporte y comunidad	5	4
Total	**23**	**20**

Tabla 3.25: Matriz de decisión - Análisis Estático, selección de herramientas del marco de automatización. Fuente: elaboración propia.

SonarQube fue seleccionado por ser la única herramienta del conjunto evaluado capaz de analizar simultáneamente código Python, JavaScript y Java en un único reporte consolidado, lo cual es determinante para un equipo con dos células tecnológicas diferenciadas. ESLint y Pylint son excelentes analizadores para sus respectivos lenguajes pero requieren configuraciones separadas y no ofrecen una vista unificada de la calidad del código. SonarQube *Community Edition* es gratuita y se integra nativamente con GitLab mediante su plugin oficial.

3.6.3. Arquitectura del Marco

El marco de automatización se diseñó siguiendo dos principios arquitectónicos derivados de las restricciones organizacionales identificadas en el diagnóstico: modularidad por dominio funcional y centralización de reportería. El primero responde a la estructura de aplicaciones Django del proyecto, donde cada módulo del sistema: afiliados, urgencias y referencias constituye una unidad funcional independiente con su propia lógica de negocio. El segundo responde a la necesidad de generar evidencia consolidada y trazable del proceso de aseguramiento de calidad, requisito derivado del cumplimiento de la Resolución 866 de 2021.

Patrón arquitectónico - Pruebas co-localizadas por dominio

Las pruebas unitarias del backend adoptaron el patrón de pruebas co-localizadas por dominio, estándar recomendado en proyectos Django (Okken, 2022). En este patrón cada aplicación del sistema contiene su propio archivo `tests.py` y sus datos de prueba en formato JSON, ubicados dentro de la misma carpeta de la aplicación. Esta decisión garantiza que las pruebas estén físicamente próximas al código que validan, facilitando su mantenimiento cuando el código de negocio evoluciona y reduciendo el riesgo de que las pruebas queden desactualizadas tras cambios en el sistema.

Estructura del marco - Componente Backend

El componente backend se organiza en tres niveles dentro del repositorio:

El primer nivel corresponde a la configuración central, compuesta por el archivo `pytest.ini` que define los parámetros globales de ejecución, el archivo `requirements-qa.txt` que declara las dependencias del marco de pruebas de manera separada a las dependencias del sistema, el archivo `conftest.py` que centraliza los fixtures compartidos entre aplicaciones, el archivo `sonar-project.properties` que define los parámetros de análisis estático para SonarQube, el archivo `package.json` que centraliza el comando de ejecución de Newman, permitiendo que el pipeline invoque las pruebas de integración mediante un comando unificado independientemente del entorno de ejecución, y el archivo `.gitlab-ci.yml` que orquesta la ejecución automatizada en el pipeline.

El segundo nivel corresponde a las pruebas por dominio funcional. Cada aplicación Django del módulo de Referencia 24/7 contiene un archivo `tests.py` con los casos de prueba automatizados y un archivo JSON con los datos sintéticos utilizados como *fixtures*. Para el piloto se implementaron

pruebas en dos aplicaciones: `appafiliados`, que cubre los casos CP-REF-01 al CP-REF-05 correspondientes a la funcionalidad de búsqueda y validación de afiliados, y `urg_aturgencia`, que cubre los casos CP-REF-07 al CP-REF-11 correspondientes al registro de urgencias.

El tercer nivel corresponde a la carpeta centralizada `reportes_qa/`, que consolida todos los artefactos generados por la ejecución de las pruebas. Esta carpeta contiene cuatro tipos de reporte generados automáticamente por el pipeline: el reporte HTML interactivo de ejecución de pruebas (`reporte_unitarias.html`), el reporte de cobertura en formato HTML navegable (`htmlcov/`), el reporte de cobertura en formato XML compatible con GitLab (`coverage.xml`) y el reporte JUnit XML para integración nativa con la interfaz de GitLab (`junit_report.xml`).

La figura siguiente ilustra la estructura completa del marco en el componente backend:

```
marco-qa-epsi-aic/
│
├── backend/                          ← IMPLEMENTADO EN PILOTO
│   ├── pytest.ini                    ← Configuración global de Pytest
│   ├── requirements-qa.txt           ← Dependencias del marco QA
│   ├── conftest.py                   ← Fixtures compartidos
│   ├── sonar-project.properties      ← Análisis estático backend
│   ├── .gitlab-ci.yml                ← Pipeline CI/CD backend (3 etapas)
│   │                                    etapa 1: test_unit (Pytest)
│   │                                    etapa 2: quality (SonarQube + Quality Gate)
│   │                                    etapa 3: test_integration (Newman)
│   ├── package.json                  ← Comando npm run test:integration:qa
│   │
│   ├── appafiliados/                 ← Dominio: búsqueda de afiliados
│   │   ├── tests.py                  ← CP-REF-01 al CP-REF-05
│   │   └── afiliados_mock.json       ← Datos sintéticos de prueba
│   │
│   ├── urg_aturgencia/               ← Dominio: urgencias y TRIAGE
│   │   ├── tests.py                  ← CP-REF-07 al CP-REF-11
│   │   └── urgencias_mock.json       ← Datos sintéticos de prueba
│   │
│   ├── postman/                      ← Pruebas de integración de APIs
│   │   ├── collections/
│   │   │   └── API_Referencia_24_7.postman_collection.json
│   │   ├── env/
│   │   │   ├── local.json            ← Entorno local desarrollador
│   │   │   └── qa.json               ← Entorno staging pipeline
│   │   └── reportes_qa/
│   │       ├── report.html           ← Reporte HTML interactivo Newman
│   │       └── newman-report.xml     ← Reporte JUnit XML GitLab
│   │
│   └── reportes_qa/                  ← Reportería centralizada Pytest
│       ├── reporte_unitarias.html
│       ├── junit_report.xml
│       ├── coverage.xml
│       └── htmlcov/
```

Figura 3.10: Estructura arquitectónica del marco de automatización - Componente Backend. Fuente: elaboración propia.

Estructura del marco - Componente Frontend

El componente frontend adoptó una arquitectura de pruebas en dos niveles con patrones diferenciados según el tipo de validación. Para las pruebas unitarias se aplicó el patrón de co-locación, donde cada componente transaccional aloja su archivo `.test.tsx` en el mismo directorio de su implementación, garantizando una rápida identificación y mantenimiento ante cambios en la interfaz. Para las pruebas E2E se aplicó el patrón *Page Object Model*, centralizando los selectores del DOM y las interacciones de usuario en clases reutilizables dentro de la carpeta `pages/`, separando así la lógica de navegación de la lógica de validación.

La configuración del entorno de pruebas unitarias integró el motor de simulación `jsdom` para proveer abstracciones sintéticas del DOM — `window`, `document`, `navigator` — permitiendo ejecutar pruebas de componentes React en un entorno Node.js sin necesidad de un navegador real. El motor de cobertura `v8` se configuró con el parámetro `all: true`, que obliga a Vitest a escanear en frío la totalidad de los archivos del código fuente incluyendo aquellos sin pruebas adjuntas, garantizando una métrica matemáticamente real de cobertura integral. El frontend alcanzó una cobertura del 42.74 %, superando ampliamente el umbral mínimo del 15 % definido en el *Quality Gate* del proceso TO-BE.

Las pruebas E2E con Playwright se configuraron para ejecutarse en tres navegadores simultáneamente: Chromium, Firefox y WebKit, garantizando compatibilidad transversal del sistema. Para optimizar la trazabilidad diagnóstica sin degradar el rendimiento del *pipeline*, se definieron políticas estrictas de recolección de artefactos de falla: capturas de pantalla solo ante fallos, grabación de video conservada únicamente en casos fallidos y trazas del navegador almacenadas en el primer reintento. Esta configuración garantiza que cada falla en el *pipeline* genere evidencia visual suficiente para su diagnóstico sin incrementar innecesariamente el almacenamiento del servidor.

Una decisión de diseño relevante fue la sincronización matemática entre Vitest y SonarQube mediante espejo de exclusiones. Dado que SonarQube Community evalúa el proyecto de forma monolítica, cualquier archivo analizado por Vitest pero ignorado por SonarQube, o viceversa, genera inconsistencias en las métricas de cobertura. Para garantizar la simetría absoluta, las exclusiones declaradas en `vite.config.ts` se replicaron exactamente en `sonar-project.properties`, eliminando advertencias de desincronización y asegurando que el porcentaje reportado en SonarQube corresponda exactamente al calculado por Vitest. Adicionalmente, se implementó una optimización del *pipeline* que redujo el tiempo de instalación de dependencias de siete minutos a menos de 45 segundos mediante las banderas `--no-audit`, `--no-fund` y `--prefer-offline` combinadas con redirección de caché local.

La figura siguiente ilustra la estructura completa del marco en el componente frontend:

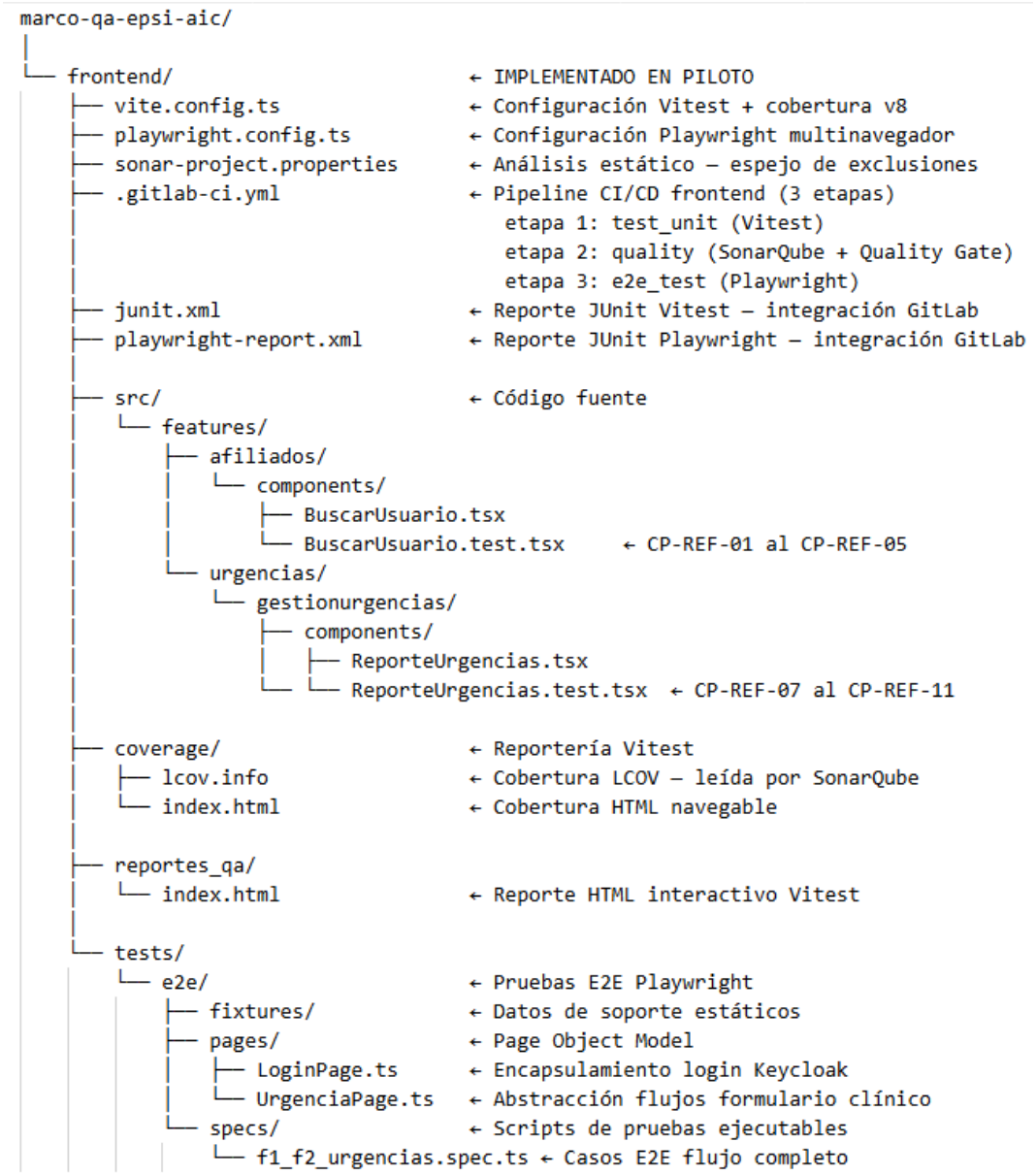


Figura 3.11: Estructura arquitectónica del marco de automatización - Componente Frontend. Fuente: elaboración propia.

Pipeline CI/CD - Etapas secuenciales con Quality Gate

El *pipeline* de GitLab orquesta la ejecución del marco en tres etapas secuenciales con dependencias explícitas entre ellas, implementando el *Quality Gate* definido en la sección 3.3.3 como barrera de control entre etapas:

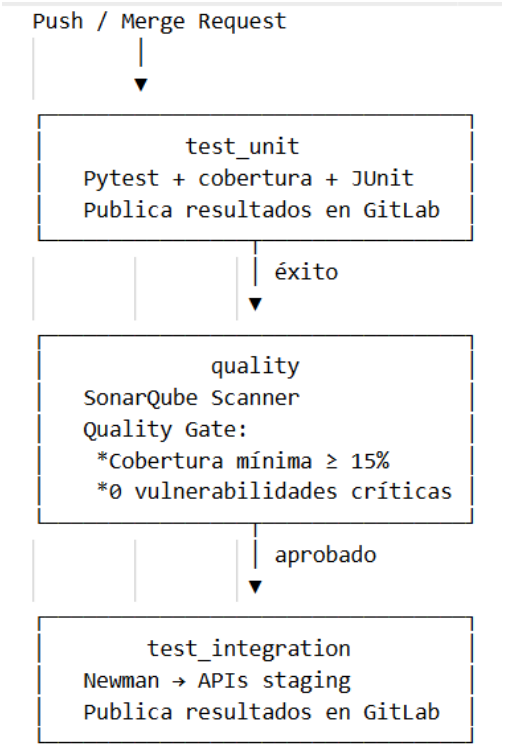


Figura 3.12: Pipeline CI/CD: Etapas secuenciales con Quality Gate. Fuente: elaboración propia.

La primera etapa `test_unit` ejecuta Pytest contra las aplicaciones del piloto usando imágenes Docker del registro interno de la organización, genera los cuatro artefactos de reportería y publica los resultados directamente en la interfaz de GitLab mediante integración nativa con los formatos JUnit y Cobertura. La segunda etapa `quality` ejecuta SonarQube Scanner para análisis estático del código — si esta etapa detecta vulnerabilidades críticas el *pipeline* se detiene y bloquea el avance a la siguiente etapa, implementando así el *Quality Gate* definido en el proceso TO-BE. La tercera etapa `test_integration` ejecuta las colecciones de Postman mediante Newman contra el ambiente de *staging*, tomando como prerrequisito la finalización exitosa de las dos etapas anteriores. Una característica relevante del *pipeline* es que opera completamente sobre infraestructura local de la EPSI AIC, utilizando imágenes Docker alojadas en el registro interno de la organización. Esto garantiza que el marco de automatización no depende de servicios externos ni de conectividad a internet para su ejecución, condición necesaria dado que los sistemas del módulo de Referencia 24/7 operan en una red interna sin exposición pública.

Capa de integración de APIs — Newman

Las pruebas de integración de APIs se organizaron en una carpeta `postman/` dentro del mismo repositorio del backend, estructurada en tres subcarpetas. La carpeta `collections/` contiene la

colección `API_Referencia_24_7.postman_collection.json` con los casos de integración del módulo. La carpeta `env/` contiene dos archivos de entorno separados: `local.json` para ejecución desde la máquina del desarrollador y `qa.json` para ejecución contra el ambiente de *staging* desde el *pipeline*, permitiendo que la misma colección se ejecute en ambos contextos sin modificaciones. La carpeta `reportes_qa/` centraliza los artefactos generados por Newman: el reporte HTML interactivo `report.html` y el reporte JUnit XML `newman-report.xml` para integración nativa con GitLab. La ejecución de Newman desde el *pipeline* se centraliza mediante el comando `npm run test:integration:qa` declarado en el `package.json` del backend, garantizando que tanto el desarrollador en local como el *pipeline* de GitLab usen exactamente el mismo comando de ejecución. Esta separación de entornos es una decisión de diseño relevante para el contexto de la EPSI AIC, donde el ambiente de *staging* replica el 99 % del esquema de producción según se documentó en la sección 3.2.1.3. Contar con un archivo de entorno específico para cada contexto garantiza que las pruebas de integración validen el comportamiento real del sistema en condiciones equivalentes a producción, sin depender de configuraciones locales que podrían introducir falsos positivos.

3.6.4. Implementación Piloto

La implementación piloto constituyó la materialización concreta del marco de automatización diseñado en las secciones anteriores. El piloto se desarrolló sobre el Módulo de Referencia 24/7, sistema seleccionado como prioritario en la Fase B por su condición de seguridad crítica para el paciente, e integró dos componentes: el backend desarrollado en Django y el frontend construido con React y Vite. En el backend se automatizaron 10 casos de prueba en los dominios de búsqueda y validación de afiliados (CP-REF-01 al CP-REF-05) y registro de urgencias (CP-REF-07 al CP-REF-11); en el frontend, 13 pruebas unitarias con Vitest y 3 pruebas End-to-End con Playwright que cubren el flujo integrado de ambas funcionalidades. La ejecución se integró en un pipeline de CI/CD de tres etapas secuenciales en GitLab, operando sobre infraestructura local de la EPSI AIC, de forma que cada ciclo de desarrollo activa la verificación del Quality Gate antes de habilitar la promoción del artefacto. El objetivo del piloto no fue alcanzar cobertura total, sino demostrar la viabilidad técnica y organizacional del marco, estableciendo una base funcional y documentada sobre la cual escalar la automatización de manera incremental.

Las evidencias detalladas de la ejecución del piloto (logs de consola, integración de resultados en GitLab, cobertura por módulo, análisis estático y reportes de integración y E2E) se recopilan en el Anexo B

3.6.4.1. Componente backend

La implementación del backend automatizó 10 casos de prueba en dos dominios funcionales del Módulo de Referencia 24/7: búsqueda y validación de afiliados (CP-REF-01 al CP-REF-05) y registro de urgencias (CP-REF-07 al CP-REF-11). La ejecución se integró en un pipeline de tres etapas secuenciales sobre infraestructura local de la EPSI AIC, validando en cada ciclo que el código cumple los criterios de calidad del proceso TO-BE antes de avanzar.

Ejecución del pipeline El *pipeline* #1364, ejecutado sobre la rama `desarrollo` tras la integración de `feature/api-integration-newman`, completó exitosamente sus tres etapas (`test_unit`, `quality` y `test_integration`) de forma secuencial en 3 minutos y 41 segundos, evidenciando el flujo de control del proceso *TO-BE* en el que cada etapa actúa como prerrequisito de la siguiente.

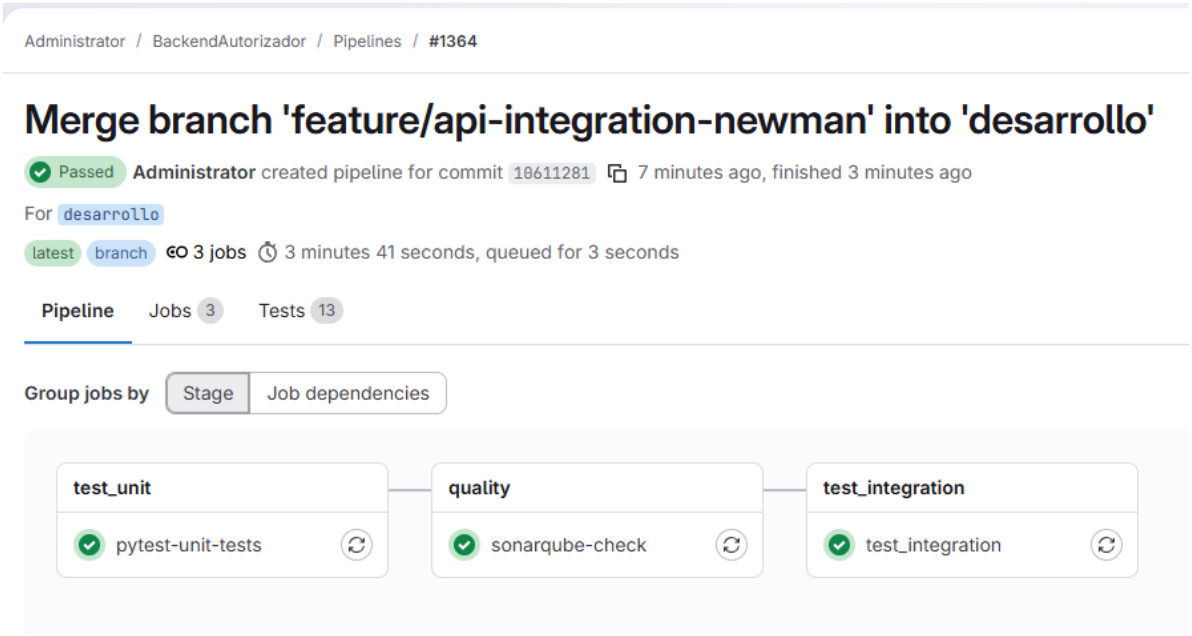


Figura 3.13: Vista general del pipeline CI/CD en GitLab — tres etapas completadas exitosamente. Fuente: elaboración propia.

Resultados de pruebas unitarias La suite unitaria ejecutó los 10 casos CP-REF-01 a CP-REF-11 con resultado *Passed* y 100 % de éxito, cubriendo las aplicaciones `appafiliados` (`TestBusquedaAfiliado`, CP-REF-01 a 05) y `urg_aturgencia` (`TestRegistroUrgencia`, CP-REF-07 a 11), con generación automática de los reportes `junit_report.xml`, `coverage.xml` y `reporte_unitarias.html`. El *log* de consola de la ejecución y la integración nativa de los resultados JUnit en la pestaña *Tests* de GitLab se presentan en el Anexo B (Figuras B.1 y B.2).

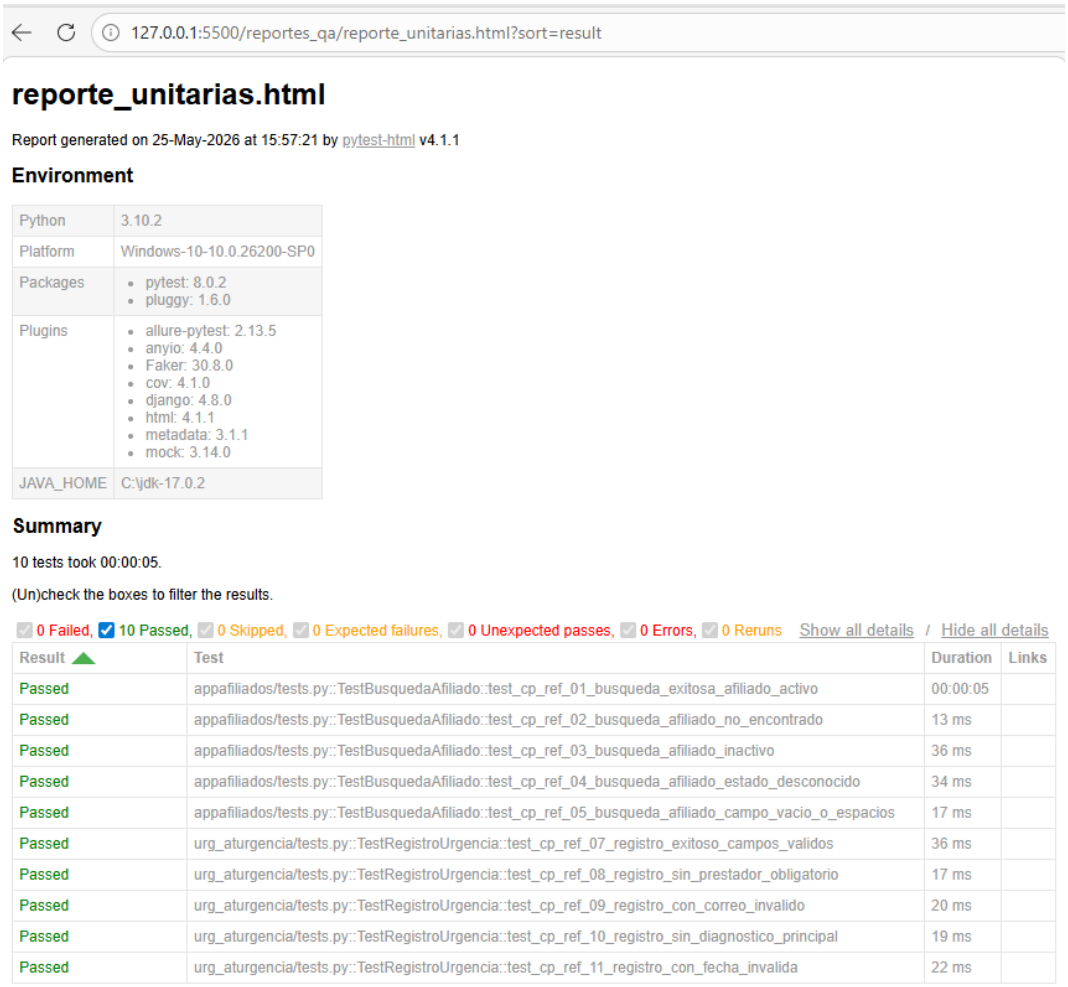
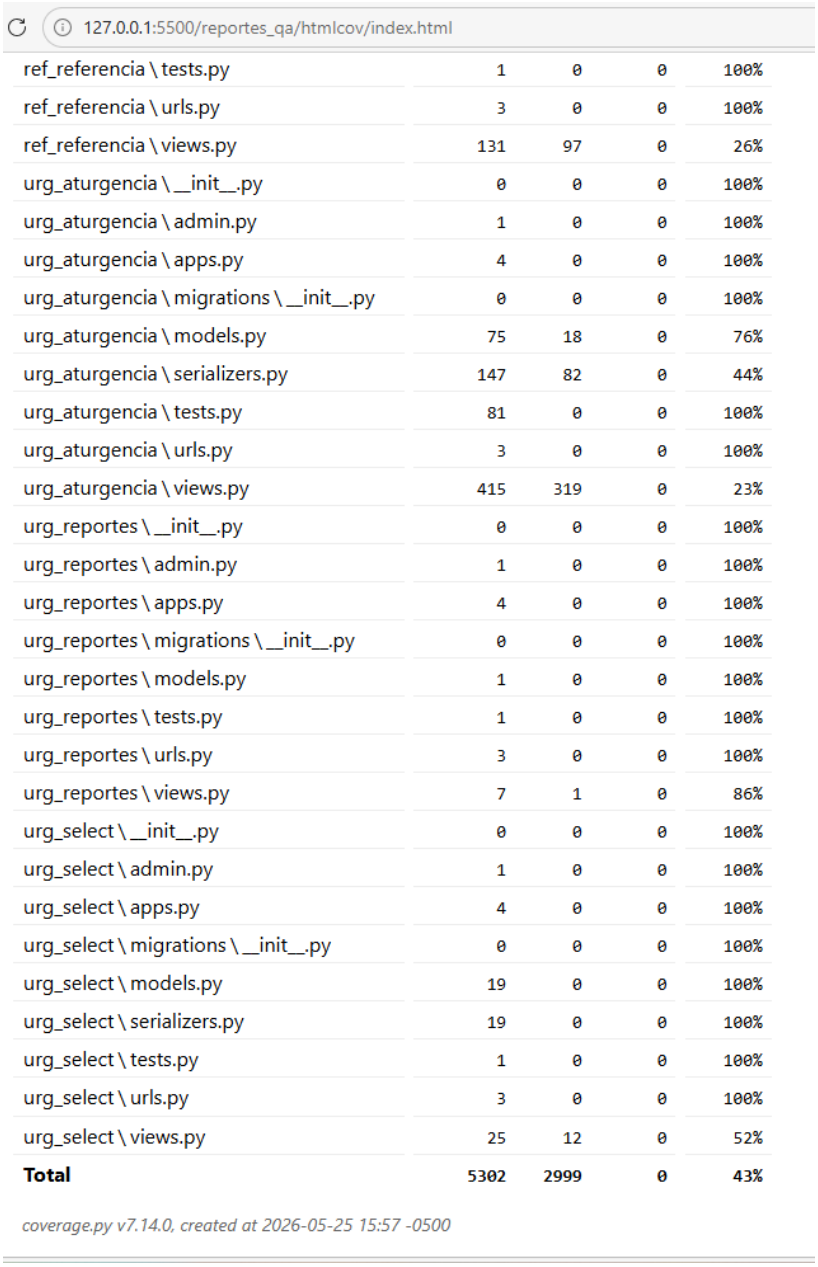


Figura 3.14: Reporte HTML de Pytest — listado completo de casos CP-REF-01 al CP-REF-11 con resultado `Passed`. Fuente: elaboración propia.

Cobertura de código — Reporte HTML de Pytest El reporte generado por `coverage.py` reportó un 43 % de cobertura global sobre 5.302 *statements*, superando el umbral mínimo del 15 % del *Quality Gate*; este valor refleja la cobertura integral del proyecto; incluidos módulos sin pruebas en el piloto, de forma coherente con el principio de mejora incremental de la sección 3.3.1. El detalle de cobertura por módulo de `urg_aturgencia` y `appafiliados` se presenta en el Anexo B (Figuras B.3 y B.4).



127.0.0.1:5500/reportes_qa/htmlcov/index.html

ref_referencia \ tests.py	1	0	0	100%
ref_referencia \ urls.py	3	0	0	100%
ref_referencia \ views.py	131	97	0	26%
urg_aturgencia \ __init__.py	0	0	0	100%
urg_aturgencia \ admin.py	1	0	0	100%
urg_aturgencia \ apps.py	4	0	0	100%
urg_aturgencia \ migrations \ __init__.py	0	0	0	100%
urg_aturgencia \ models.py	75	18	0	76%
urg_aturgencia \ serializers.py	147	82	0	44%
urg_aturgencia \ tests.py	81	0	0	100%
urg_aturgencia \ urls.py	3	0	0	100%
urg_aturgencia \ views.py	415	319	0	23%
urg_reportes \ __init__.py	0	0	0	100%
urg_reportes \ admin.py	1	0	0	100%
urg_reportes \ apps.py	4	0	0	100%
urg_reportes \ migrations \ __init__.py	0	0	0	100%
urg_reportes \ models.py	1	0	0	100%
urg_reportes \ tests.py	1	0	0	100%
urg_reportes \ urls.py	3	0	0	100%
urg_reportes \ views.py	7	1	0	86%
urg_select \ __init__.py	0	0	0	100%
urg_select \ admin.py	1	0	0	100%
urg_select \ apps.py	4	0	0	100%
urg_select \ migrations \ __init__.py	0	0	0	100%
urg_select \ models.py	19	0	0	100%
urg_select \ serializers.py	19	0	0	100%
urg_select \ tests.py	1	0	0	100%
urg_select \ urls.py	3	0	0	100%
urg_select \ views.py	25	12	0	52%
Total	**5302**	**2999**	**0**	**43%**

coverage.py v7.14.0, created at 2026-05-25 15:57 -0500

Figura 3.15: Reporte de cobertura HTML — cobertura global del 43 % sobre el proyecto completo. Fuente: elaboración propia.

Análisis estático — SonarQube El proyecto Autorizador - Backend reportó el *Quality Gate* como *Passed* sobre 7.6k líneas analizadas, con *Security* D (3 *issues*), *Reliability* C (11 *issues*), *Maintainability* A (103 *issues*), cobertura del 41.1 % y 2.8 % de duplicación. El desglose de *issues* por severidad, sin hallazgos *Blocker* ni *Info*, y la vista de cobertura y resultados de pruebas se

documentan en el Anexo B (Figuras B.5 y B.6).

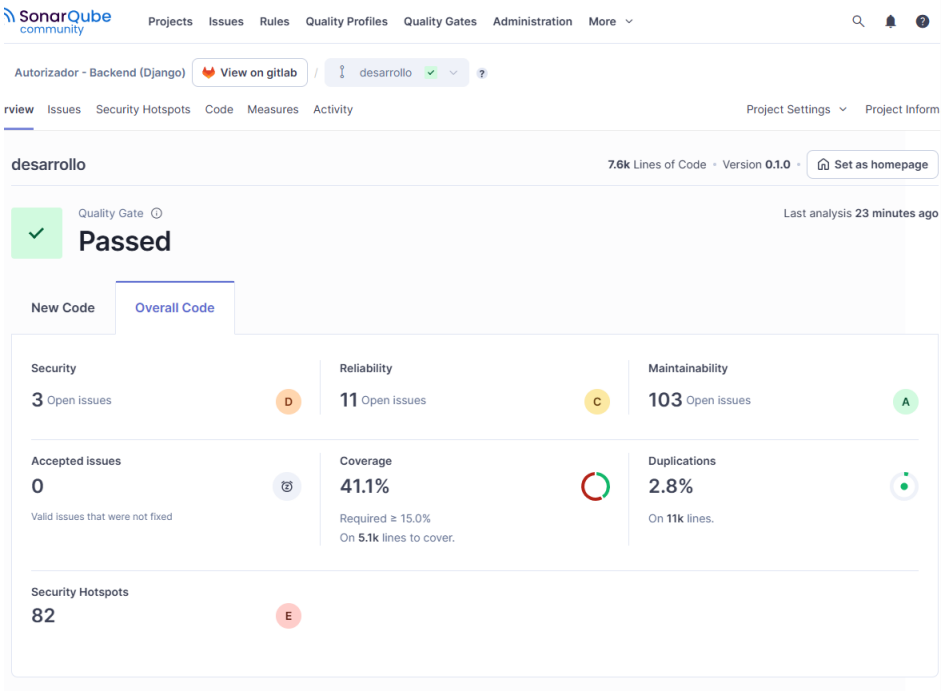


Figura 3.16: Dashboard principal del proyecto en SonarQube — Quality Gate aprobado para el backend Autorizador. Fuente: elaboración propia.

Pruebas de integración — Newman Las pruebas de integración de APIs se ejecutaron con Newman sobre la colección `API_Referencia_24_7` (entorno `REF_AIC_Test`): 3 *requests* y 3 *assertions* ejecutados sin fallas, con la validación funcional "Urgencia creada exitosamente" e integración nativa de los resultados en la pestaña *Tests* del *pipeline* (100 % de éxito en 3.63 s). La evidencia detallada reporte HTML, resumen de consola e integración en GitLab, se presenta en el Anexo B (Figuras B.7 a B.9).

3.6.4.2. Componente Frontend

Ejecución del pipeline El *pipeline* #1459 del repositorio `frontautorizador` ejecutó secuencialmente sus tres etapas (`test_unit`, `quality` y `e2e_test`) con resultado satisfactorio sobre 16 pruebas, replicando en el *frontend* el flujo de control del proceso *TO-BE*. La vista general del *pipeline* y el *merge request* !137, que evidencia que la promoción del artefacto solo se habilita tras superar las validaciones se presentan en el Anexo B (Figuras B.10 y B.11).

Resultados de pruebas unitarias — Vitest La suite unitaria del *frontend*, ejecutada con Vitest, reportó 13 pruebas aprobadas sobre 13 (100 % de éxito), distribuidas en tres archivos: el módulo

de afiliados (`BuscarUsuario.test.tsx`, CP-REF-01 a 05), el de urgencias (`ReporteUrgencias.test.tsx`, CP-REF-07 a 11 más la validación del *checkbox* "Sin Validaciones") y los componentes compartidos (`BotonTest.test.tsx`). El listado completo por módulo confirma la trazabilidad de cada caso frente a su criterio funcional.

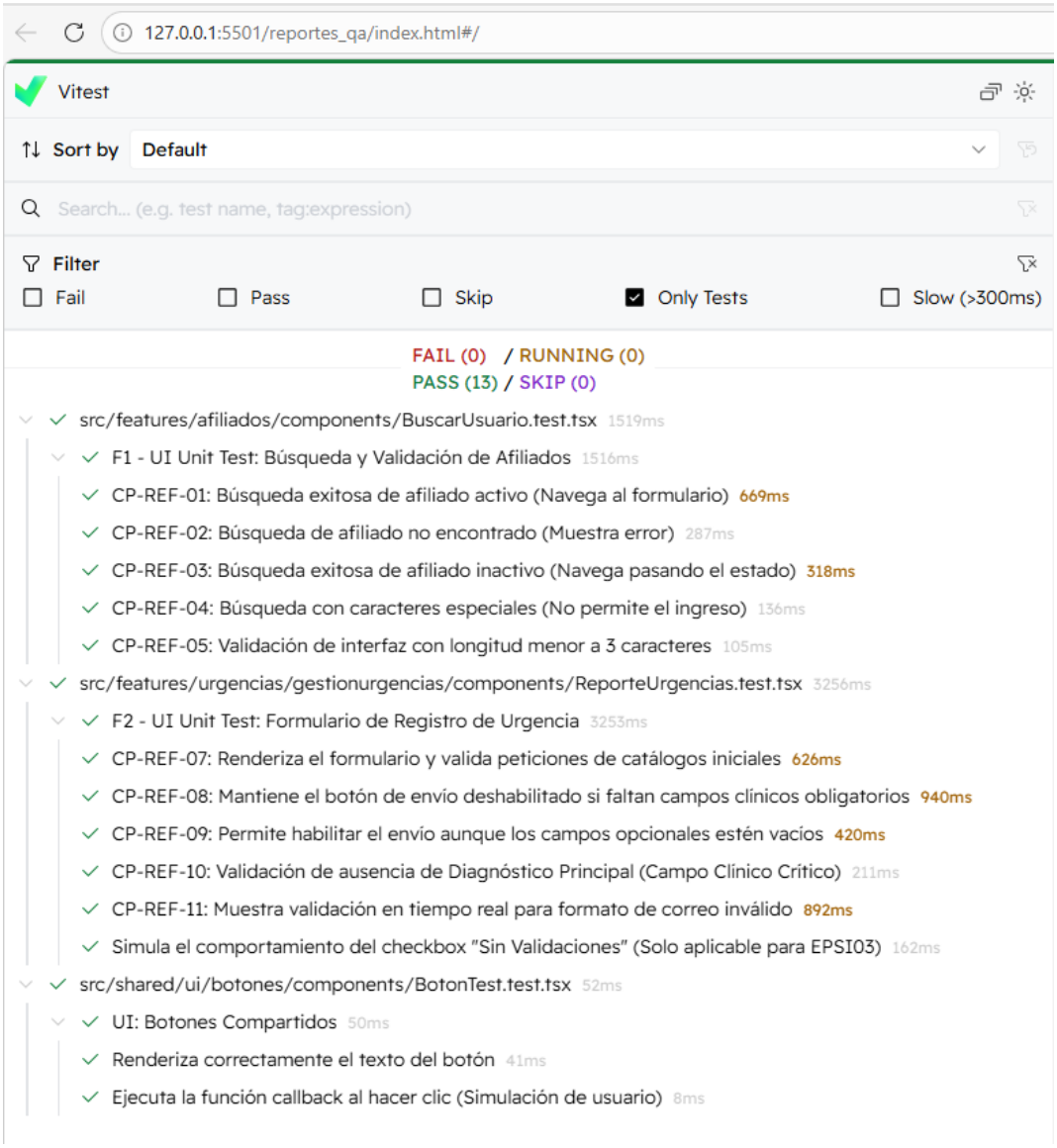


Figura 3.17: Reporte HTML de Vitest del frontend — listado completo de casos por módulo con resultado satisfactorio. Fuente: elaboración propia.

El detalle de la etapa `test_unit` y la integración de los resultados en la pestaña *Tests* de GitLab se documentan en el Anexo B (Figuras B.12 y B.13).

Cobertura de Código y Análisis Estático La cobertura del *frontend* (41.61 % de *statements* para `All files`, con 78.78 % en `features/afiliados/components` y 59.84 % en `features/urgencias/gestionur` áreas priorizadas por el piloto), su detalle por módulo, la sincronización de exclusiones entre Vitest y SonarQube, y el *dashboard* de SonarQube con *Quality Gate Passed* (*Security* A, *Reliability* C, *Maintainability* A, sin *issues Blocker*) se presentan en el Anexo B (Figuras B.14 a B.19).

Pruebas E2E — Playwright Las pruebas E2E se ejecutaron con Playwright sobre el flujo integrado de búsqueda de afiliado y registro de urgencia (`specs/f1_f2_urgencias.spec.ts`, caso "*Happy Path*" con intercepción de red), con 9 pruebas aprobadas sobre 9 ejecutadas en los tres motores de navegador (Chromium, Firefox y WebKit), confirmando el comportamiento consistente del flujo crítico.

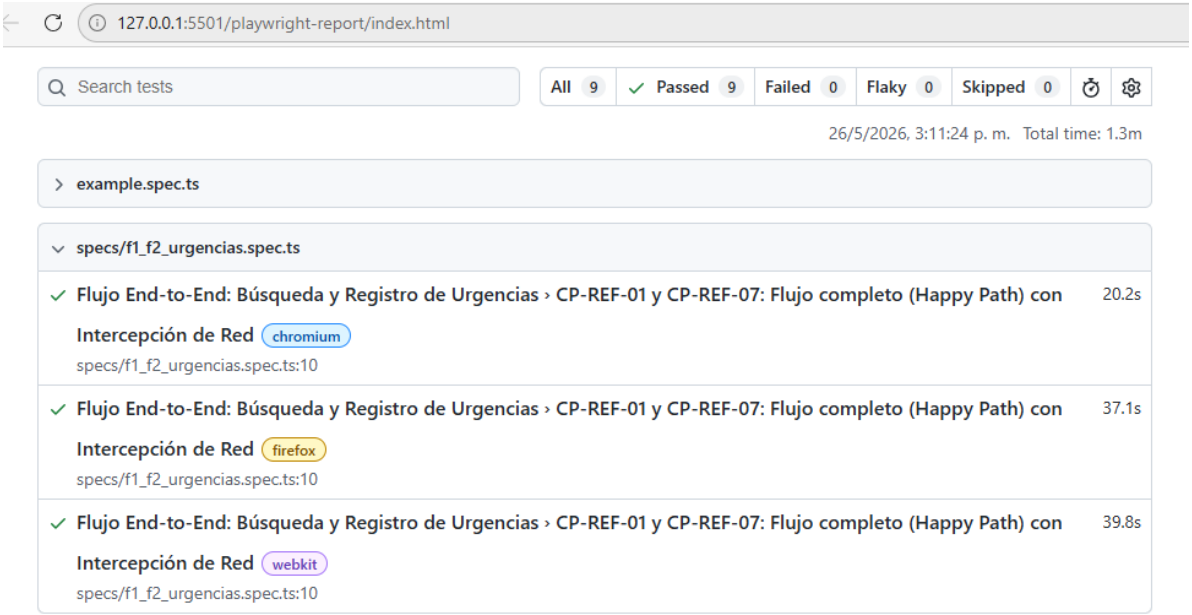


Figura 3.18: Reporte HTML de Playwright del frontend — ejecución del flujo *End-to-End* de búsqueda y registro de urgencias en Chromium, Firefox y WebKit. Fuente: elaboración propia.

La etapa `e2e_test` del *pipeline* y la integración de los resultados en la pestaña *Tests* de GitLab se incluyen en el Anexo B (Figuras B.20 y B.21).

3.6.5. Documentación Técnica

La sostenibilidad de un marco de automatización depende no solo de su implementación técnica sino de la calidad de su documentación, especialmente en organizaciones con alta rotación de personal como la EPSI AIC, donde el conocimiento institucional debe estar disponible independientemente de quién ocupe cada rol. Este principio, identificado como brecha crítica en el diagnóstico

AS-IS y abordado como mecanismo de transferencia en la sección 3.4.1.5, se materializó en la generación de ocho documentos técnicos que cubren la totalidad del marco de automatización implementado en el piloto, distribuidos entre el componente *backend* y el componente *frontend*.

La documentación se organizó en dos niveles. El primer nivel corresponde al documento general del repositorio, el archivo `README.md` principal que describe el propósito del marco, el estado de cada componente, los requisitos de instalación, los comandos de ejecución rápida para ambos componentes y la referencia al proyecto de grado del cual forma parte. Este documento actúa como puerta de entrada para cualquier integrante del equipo que acceda al repositorio por primera vez, orientándolo hacia la documentación específica de cada capa sin necesidad de explorar el código.

El segundo nivel corresponde a los documentos técnicos específicos de cada capa, organizados por componente. Cada documento sigue una estructura consistente que incluye descripción del módulo, casos de prueba cubiertos, arquitectura de la solución, requisitos, instrucciones de instalación y ejecución, reportes generados, relación con el *pipeline* y guía de resolución de problemas frecuentes.

Los cuatro documentos del componente *backend* son los siguientes:

- El `README.md` del *backend* describe el marco completo del componente, articulando las tres capas de verificación automatizada —pruebas unitarias con Pytest, pruebas de integración con Newman y análisis estático con SonarQube— y orientando al lector hacia cada guía específica mediante referencias directas.
- El `README_UNIT_TESTS.md` documenta en detalle la implementación de las pruebas unitarias con Pytest, incluyendo la arquitectura de optimización para entornos CI/CD basada en SQLite *in-memory*, el *bypass* de migraciones y la aceleración criptográfica que permiten ejecutar los 17 casos de prueba en milisegundos dentro del *pipeline* de GitLab.
- El `README_SONARQUBE.md` describe la configuración e interpretación del análisis estático, explicando los umbrales del *Quality Gate*, cómo acceder al *dashboard*, cómo interpretar cada sección de resultados y cómo se relaciona esta etapa con el resto del *pipeline*.
- El `README.md` del directorio `postman` documenta las pruebas de integración con Newman, incluyendo la técnica de *Request Chaining* que permite encadenar automáticamente los resultados de una petición como entrada de la siguiente, simulando así el flujo real del usuario en el sistema sin datos *hardcodeados* entre peticiones.

Los cuatro documentos del componente *frontend* son los siguientes:

- El `README.md` del *frontend* describe el marco completo del componente, articulando las tres capas de verificación —pruebas unitarias con Vitest, análisis estático con SonarQube y pruebas *End-to-End* con Playwright— detalla la estructura del repositorio, los comandos de ejecución por capa, el diagrama del *pipeline* y los reportes generados por cada herramienta.
- El `README_UNITARIAS.md` documenta en detalle las dos *suites* de pruebas unitarias implementadas con Vitest: la *suite* F1 sobre el componente `BuscarUsuario` (CP-REF-01 al CP-REF-05)

y la *suite* F2 sobre el componente `ReporteUrgencias` (CP-REF-07 al CP-REF-11 y casos de componentes compartidos), describiendo la arquitectura de aislamiento mediante *mocks* de servicios API, contextos de autenticación y la función auxiliar `renderWithProviders` que replica el árbol de contexto real de la aplicación.

- El `README_E2E.md` documenta las pruebas *End-to-End* con Playwright, describiendo la implementación del patrón *Page Object Model* en las clases `LoginPage` y `UrgenciaPage`, el flujo paso a paso del caso misional CP-REF-01 y CP-REF-07, la configuración de variables de entorno para las credenciales de Keycloak y el manejo de variabilidad por tipo de habilitación organizacional implementado mediante bloques `try/catch` en los *Page Objects*.
- El `README_SONARQUBE.md` del *frontend* describe la configuración del análisis estático para el componente React + Vite, explicando la lógica de exclusiones de módulos no priorizados en el piloto, la regla de sincronización entre `sonar.exclusions` y `sonar.coverage.exclusions` que garantiza el cálculo correcto del porcentaje de cobertura, y el procedimiento para incorporar nuevos módulos al análisis de forma incremental sin generar bloqueos inesperados del *pipeline*.

Toda la documentación está disponible en el repositorio público del marco de automatización en la siguiente dirección: https://gitlab.com/andersonlopez.dev/marco-qa-epsi-aic. Este repositorio contiene adicionalmente los archivos de configuración anonimizados, los casos de prueba con datos sintéticos y la carpeta de evidencias del piloto, constituyendo un artefacto completo y verificable del trabajo realizado.

Capítulo 4

Evaluación

4.1. Marco Metodológico de la Evaluación

La evaluación del impacto del marco de aseguramiento de calidad implementado corresponde al cuarto eje del paradigma *Design Science Research* (DSR) adoptado en este proyecto: la Validación del Tratamiento. Este eje tiene como propósito central determinar en qué medida el artefacto diseñado e implementado resolvió el problema organizacional identificado en la Fase A, generando evidencia verificable de su utilidad y contribución a los objetivos del proyecto.

La estrategia de evaluación siguió un diseño cuasi-experimental con medición pre/post implementación, recomendado por (Cook and Campbell, 1979) para contextos organizacionales donde no existen grupos de control independientes ni registros históricos formales. En este diseño, el mismo equipo actúa como grupo de control (estado AS-IS, medido mediante la línea base referencial construida en la sección 3.2.3) y como grupo experimental (estado TO-BE, medido al finalizar el piloto). La triangulación de cuatro fuentes de evidencia — mediciones cronometradas de validación manual, métricas objetivas del *pipeline*, análisis estático de SonarQube y evaluación cualitativa del equipo — garantiza la validez interna de los hallazgos y permite identificar factores contextuales que influyen en los resultados.

La evaluación se estructuró en cinco actividades secuenciales correspondientes a las actividades D.1 a D.5 definidas en la sección 1.5: medición de indicadores de proceso, medición de indicadores de producto, evaluación de continuidad operativa, análisis cualitativo del equipo y análisis comparativo con recomendaciones de escalamiento.

Dado el carácter piloto del proyecto y el tamaño reducido del equipo, los resultados tienen validez contextual específica para la EPSI AIC y no son generalizables estadísticamente. Esta limitación es coherente con el paradigma DSR, que prioriza la relevancia contextual y la utilidad práctica del artefacto sobre la generalización estadística.

4.2. Instrumento de Evaluación Cualitativa

Para recolectar la percepción del equipo sobre el impacto del marco implementado, se diseñó y aplicó un instrumento de evaluación post-implementación estructurado en cinco secciones, alineadas con las dimensiones de evaluación del objetivo específico 5: confianza en despliegues, detección y diagnóstico de errores, adopción del proceso, sostenibilidad y escalamiento, y una pregunta abierta final. El instrumento combinó escalas Likert de cinco puntos con preguntas de selección múltiple

y respuestas abiertas, siguiendo el principio de triangulación metodológica recomendado por (Yin, 2018).

El instrumento fue aplicado durante el mes de mayo de 2026 a cuatro actores clave del proceso de aseguramiento de calidad, seleccionados por representar los niveles estratégico, táctico y operativo de la organización. La Tabla 4.1 describe el perfil de cada participante y su nivel de involucramiento en el piloto.

Rol	Nivel	Participación en el piloto
Líder del Subproceso de Desarrollo	Táctico	Sí, de forma parcial
Desarrollador de Software 1	Operativo	Sí, de forma parcial
Desarrollador de Software 2	Operativo	Sí, de forma parcial
Líder de Gestión TICs	Táctico-Estratégico	No, sólo informado de los resultados

Tabla 4.1: Participantes en la evaluación post-implementación. Fuente: elaboración propia.

La distinción entre participación directa e indirecta en el piloto es metodológicamente relevante, dado que permite contrastar las percepciones de quienes interactuaron directamente con el marco con la perspectiva del actor que lo evaluó desde los resultados reportados. Esta diferencia de perspectiva se refleja en algunos de los hallazgos cualitativos presentados en la sección 4.5.

4.3. Indicadores de Proceso (D.1)

4.3.1. Cobertura de Pruebas Automatizadas

La cobertura de pruebas automatizadas constituye el indicador de proceso más directo para evaluar el impacto del marco implementado, dado que la ausencia total de cobertura automatizada fue el hallazgo más crítico del diagnóstico AS-IS y el punto de partida de todo el proceso de diseño.

El piloto alcanzó los siguientes resultados de cobertura al finalizar la Fase C:

Componente	Cobertura AS-IS	Cobertura TO-BE	Variación	Umbral Quality Gate
Backend (Django)	0.0 %	41.1 %	+41.1 pp	≥15 %
Frontend (React + Vite)	0.0 %	21.9 %	+21.9 pp	≥15 %

Tabla 4.2: Cobertura de pruebas automatizadas AS-IS vs. TO-BE. Fuente: elaboración propia con base en reportes de SonarQube y Vitest.

Ambos componentes superaron el umbral mínimo del 15 % definido en el *Quality Gate* del proceso TO-BE (sección 3.3.3), validando la viabilidad técnica del marco en el contexto tecnológico de la EPSI AIC. La cobertura del backend alcanzó el 41.1 % sobre 5.1k líneas a cubrir, mientras que el frontend alcanzó el 21.9 % sobre 1.8k líneas. La diferencia entre ambos componentes se explica por el mayor número de módulos no priorizados en el piloto frontend respecto al backend, y no por una limitación técnica del marco.

Es importante contextualizar estos valores dentro del principio de mejora incremental definido en la sección 3.3.1: el piloto no tenía como objetivo cubrir la totalidad del sistema, sino demostrar la viabilidad del marco y establecer una base desde la cual el equipo pueda escalar la cobertura progresivamente. Los porcentajes obtenidos representan cobertura real sobre el proyecto completo incluyendo módulos no priorizados en el piloto; los flujos críticos específicamente cubiertos alcanzaron coberturas individuales entre el 76 % y el 100 %.

4.3.2. Productividad — Reducción del Tiempo de Validación de Regresión

Con el fin de obtener evidencia empírica objetiva del tiempo de validación en el estado AS-IS, se realizó una sesión de medición controlada en la que el Desarrollador de Software 1 ejecutó el proceso de validación manual de los dos flujos cubiertos por el piloto siguiendo exactamente el procedimiento utilizado antes de la implementación del marco: apertura de Postman, ejecución de los *endpoints*, verificación del comportamiento en el sistema y confirmación de resultado. Para el flujo de registro de urgencia se realizaron dos mediciones independientes y se tomó el valor promedio como referencia. Los resultados se presentan en la Tabla 4.3.

Flujo	**Componente**	**Tiempo AS-IS**
Búsqueda y validación de afiliados	Backend	8:58 min
Registro de referencia o urgencia	Backend	6:41 min
Total ciclo completo Backend	**-**	**15:39 min**
Búsqueda y validación de afiliados	Frontend	3:41 min
Registro de referencia o urgencia	Frontend	9:22 min
Total ciclo completo Frontend	**-**	**13:03 min**

Tabla 4.3: Tiempos de validación manual medidos AS-IS por flujo y componente. Fuente: elaboración propia.

Con el marco implementado, el *pipeline* de CI/CD ejecuta la totalidad de las pruebas automatizadas del componente backend: 10 casos de prueba unitarias, análisis estático SonarQube y pruebas de integración Newman; en 3 minutos y 41 segundos de forma completamente autónoma, sin intervención manual. Frente a los 15:39 minutos de validación manual medida, esto representa una reducción de 11:58 minutos por ciclo, equivalente al 76.5 % del tiempo de validación manual.

Componente	Validación manual	Pipeline TO-BE	Reducción abs.	Reducción rel.
Backend (2 flujos)	15:39 min	3:41 min	11:58 min	76.5 %
Frontend (2 flujos, 1 nav.)	13:03 min	15:24 min	-	-
Frontend (2 flujos, 3 nav. equivalente)	39:09 min (estimado)	15:24 min	23:45 min	60.7 %

Tabla 4.4: Comparativo de tiempos de validación AS-IS vs. TO-BE. Fuente: elaboración propia.

Respecto al componente frontend, la comparación directa de tiempos requiere una distinción metodológica importante. La validación manual medida (13:03 min) corresponde a la ejecución de los dos flujos en un único navegador, que era el procedimiento habitual del equipo antes del piloto. El *pipeline* frontend (15:24 min), en cambio, ejecuta 13 pruebas unitarias con Vitest, análisis estático completo con SonarQube y 3 pruebas E2E en tres navegadores simultáneamente: Chromium, Firefox y WebKit, cubriendo una profundidad de validación que el proceso manual no contemplaba. Si se estimara el tiempo de una validación manual equivalente en los tres navegadores (13:03 min $\times$ 3 = 39:09 min), el *pipeline* representaría una reducción del 60.7 % respecto a ese escenario. El valor diferencial del *pipeline* frontend no reside por tanto en la velocidad frente al procedimiento previo, sino en la profundidad de cobertura que habilita: realiza en 15 minutos de forma autónoma lo que manualmente requeriría 39 minutos y una disciplina de ejecución multi-navegador que el equipo no practicaba.

Ahorro acumulado durante el piloto

Durante el periodo del piloto (mayo 2026) el repositorio backend registró 32 ejecuciones del *pipeline*, con una tasa de éxito del 78 % y una tasa de falla del 19 %. El repositorio frontend registró 68 ejecuciones, con una tasa de éxito del 81 % y una tasa de falla del 16 %, para un total de 100 ejecuciones en ambos componentes.

Considerando la reducción de 11:58 min por ciclo en el backend, el marco generó un ahorro de 383 minutos (6.4 horas) de esfuerzo manual solo en ese componente durante el periodo de evaluación. Para el frontend, tomando como referencia la validación equivalente en tres navegadores, el ahorro acumulado en las 68 ejecuciones asciende a 1.615 minutos (26.9 horas). En conjunto, el marco evitó aproximadamente 33.3 horas de esfuerzo manual de validación durante el piloto, equivalentes a 4.2 jornadas laborales de un desarrollador. A esto se suma que durante la ejecución del *pipeline* el desarrollador queda completamente liberado para actividades productivas, dimensión especialmente relevante en un equipo pequeño donde cada unidad de capacidad técnica tiene alto costo de oportunidad.

4.4. Indicadores de Producto (D.2)

4.4.1. Correctitud

La correctitud se evaluó conforme a la definición del ISTQB, como la proporción de casos de prueba (CP) que se ejecutaron de forma satisfactoria respecto al total de casos de prueba ejecutados durante el piloto:

$$Correctitud = \frac{CP\ exitosos}{Total\ CP\ ejecutados} \tag{4.1}$$

Durante el piloto se ejecutaron de forma automatizada 29 casos de prueba distribuidos en los cuatro niveles del marco: pruebas unitarias del *backend* (CP-REF-01 a CP-REF-11, con Pytest), pruebas unitarias del *frontend* (Vitest), pruebas de integración de APIs (Newman) y pruebas extremo a extremo (Playwright). La totalidad de los casos se ejecutó con resultado satisfactorio, según los reportes de la sección 3.6.4 y el Anexo B.

Nivel de prueba	**Herramienta**	**CP ejecutados**	**CP exitosos**
Unitario — *backend*	Pytest	10	10
Unitario — *frontend*	Vitest	13	13
Integración de APIs	Newman	3	3
Extremo a extremo (E2E)	Playwright	3	3
Total	**-**	**29**	**29**

Tabla 4.5: Casos de prueba ejecutados y exitosos por nivel durante el piloto. Fuente: elaboración propia.

$$Correctitud = \frac{29}{29} = 1{,}00 \rightarrow 100\,\% \tag{4.2}$$

Este resultado indica que, al cierre del piloto, la totalidad de los casos de prueba automatizados del Módulo de Referencia 24/7 se ejecutaba de forma satisfactoria, estableciendo una *suite* de regresión en verde como línea base para la evolución del sistema.

Como evidencia complementaria del aseguramiento de la correctitud a nivel de código, el análisis estático de SonarQube mostró una reducción sustancial de *issues* entre el estado *AS-IS* y el *TO-BE*, especialmente en el *frontend* (*Reliability* de 78 a 20 *issues*, $-74{,}4\,\%$; *Maintainability* de 550 a 233, $-57{,}6\,\%$), producto de la integración del *Quality Gate* como condición previa al despliegue. El comparativo detallado *AS-IS* vs. *TO-BE* se presenta en la tabla de indicadores de SonarQube (Tabla 4.7).

4.4.2. Confiabilidad

La confiabilidad se evaluó conforme a la definición del ISTQB, como el complemento de la proporción de no conformidades detectadas respecto al total de casos de prueba ejecutados:

$$Confiabilidad = 1 - \left(\frac{No\ Conformidades}{Total\ CP\ ejecutados} \right) \tag{4.3}$$

Para esta medición se contabilizó como No Conformidad cada situación en la que el *Quality Gate* del *pipeline* bloqueó la promoción de un artefacto por incumplimiento real de los criterios de calidad establecidos; es decir, cada defecto o condición no conforme detectada y contenida por el marco antes de alcanzar producción. Durante el piloto se documentaron tres no conformidades de este tipo —una en el repositorio *backend*, asociada al *merge request* de la funcionalidad de búsqueda de afiliado, y dos en el repositorio *frontend*— confirmadas mediante los *logs* de GitLab (mensaje `QUALITY GATE STATUS: FAILED`), lo que descartó que se tratara de falsos positivos.

$$Confiabilidad = 1 - \left(\frac{3}{29} \right) = 0{,}897 \rightarrow 89{,}7\,\% \tag{4.4}$$

Este resultado indica que el 89.7 % de los casos de prueba ejecutados transcurrió sin que el marco detectara una no conformidad que comprometiera la promoción del artefacto, y que el 10.3 % restante correspondió a situaciones en las que el *pipeline* actuó como mecanismo de contención, bloqueando el avance de código no conforme antes de que afectara la operación.

Este comportamiento es consistente con la evaluación de disponibilidad del sistema: el Líder de Gestión TICs confirmó que durante el periodo del piloto no se registraron incidentes en producción en el Módulo de Referencia 24/7 atribuibles a defectos en los flujos cubiertos por las pruebas automatizadas. Aunque la duración limitada del piloto impide establecer causalidad directa, la ausencia de incidentes es coherente con la contención de no conformidades verificada en el *pipeline*: el *Quality Gate* transformó la detección de errores de un modelo reactivo —en el que los fallos solo se conocían cuando el usuario los reportaba— a uno proactivo que interviene antes del despliegue.

4.5. Evaluación de Continuidad Operativa (D.3)

La evaluación de la continuidad operativa analizó tres dimensiones: el funcionamiento del *Quality Gate* como compuerta de control, el impacto en el tiempo de detección de errores y el cumplimiento del criterio de cobertura mínima antes de cada despliegue.

Respecto al *Quality Gate*, los logs de GitLab y los testimonios del equipo confirmaron que el *pipeline* bloqueó despliegues durante el piloto en al menos dos ocasiones documentadas: una en el repositorio backend, asociada al *merge request* de la funcionalidad de búsqueda de afiliado, y al menos dos en el repositorio frontend, asociadas a la configuración de SonarQube. Aunque algunos participantes clasificaron inicialmente estos bloqueos como falsos positivos, el análisis de los logs confirma que todos correspondieron a situaciones reales de incumplimiento del *Quality Gate*; ya sea por cobertura insuficiente o por configuración incorrecta del análisis estático. La percepción

de falso positivo refleja la curva de aprendizaje esperada en la adopción de un nuevo proceso de calidad (Dustin et al., 1999), donde los primeros bloqueos son frecuentemente interpretados como errores del sistema antes de que el equipo comprenda los criterios de aceptación establecidos.

El criterio de cobertura mínima del 15 % se cumplió en ambos componentes al cierre del piloto: 41.1 % en el backend y 21.9 % en el frontend, superando el umbral con márgenes de +26.1 y +6.9 puntos porcentuales respectivamente.

Respecto al tiempo de detección de errores, tres de los cuatro participantes confirmaron que el marco permitió identificar errores en los flujos cubiertos en mucho menos tiempo que antes del piloto (P2.5 del instrumento), pasando de un modelo completamente reactivo, según confirmaron los cuatro participantes en P2.1 - a un modelo proactivo en el que el *pipeline* detectó y reportó problemas antes de que el artefacto avanzara hacia etapas superiores..

4.6. Análisis Cualitativo del Equipo (D.4)

El análisis cualitativo se basó en las respuestas del instrumento de evaluación post-implementación aplicado a los cuatro participantes identificados en la sección 4.2. Los resultados se presentan por dimensión analítica.

4.6.1. Confianza en Despliegues

La confianza percibida al momento de aprobar o ejecutar despliegues a producción mostró un incremento positivo tras la implementación del marco. El promedio grupal de la escala Likert pasó de 2.75 (entre “Inseguro” y “Neutral”) antes del piloto a 3.25 (entre “Neutral” y “Seguro”) después de él, con un delta de +0.5 puntos.

Rol	**Confianza AS-IS**	**Confianza TO-BE**	**Delta**
Líder del Subproceso de Desarrollo	2 - Inseguro	4 - Seguro	+2
Desarrollador de Software 1	2 - Inseguro	3 - Neutral	+1
Desarrollador de Software 2	4 - Seguro	3 - Neutral	-1
Líder de Gestión TICs	3 - Neutral	3 - Neutral	0
Promedio	**2.75**	**3.25**	**+0.5**

Tabla 4.6: Comparativo de confianza en despliegues AS-IS vs. TO-BE. Fuente: elaboración propia.

El incremento más significativo corresponde al Líder del Subproceso de Desarrollo (+2 puntos), quien identificó como cambio concreto la mejora en la calidad, seguridad y control del código al aplicar buenas prácticas en el despliegue. El Desarrollador de Software 1 señaló que el proceso brinda mayor confianza al existir una verificación previa de la calidad y funcionamiento de la aplicación antes de cada despliegue. El Desarrollador de Software 2, que partía de una percepción de seguridad superior al promedio del grupo (4 - Seguro antes del piloto), experimentó una reducción de un punto

que puede interpretarse como una mayor conciencia de los riesgos que permanecen sin cobertura automatizada — efecto documentado en la literatura como uno de los resultados esperados de la implementación de procesos formales de QA (Dustin et al., 1999). El Líder de Gestión TICs mantuvo una percepción neutral estable, coherente con su distancia operativa del proceso.

4.6.2. Detección y Diagnóstico de Errores

La utilidad de los reportes generados por el *pipeline* para identificar problemas cuando algo falla obtuvo la puntuación más alta de todo el instrumento, con un promedio de 4.75 sobre 5 (tres participantes con 5 - "Muy claros y útiles" y uno con 4 - "Ayudan bastante"). Este resultado es especialmente relevante porque los reportes (reporte HTML interactivo de Pytest/Vitest, *dashboard* de SonarQube y reporte de Newman) reemplazan el diagnóstico manual de errores que en el estado AS-IS dependía completamente de la inspección individual del código o del reporte directo del usuario.

Las respuestas abiertas sobre la naturaleza de los problemas detectados aportaron evidencia directa adicional: el Desarrollador de Software 1 identificó la detección de insuficiencia de cobertura durante la etapa de análisis de calidad como el caso más concreto; el Desarrollador de Software 2 señaló que el *pipeline* detectó errores en pruebas del módulo Orquestador al momento de un *merge*, situación que habría pasado inadvertida bajo el proceso AS-IS. Estos testimonios, corroborados por los logs de GitLab, confirman que el marco detectó situaciones reales que sin el proceso TO-BE habrían avanzado a producción sin control.

4.6.3. Adopción del Proceso

La percepción sobre si escribir pruebas unitarias junto con el código constituye una carga adicional o parte natural del trabajo obtuvo un promedio de 4.0 sobre 5, con dos participantes en el nivel máximo (5 - "Mejora el trabajo") y dos en el nivel neutro (3). El Desarrollador de Software 1 señaló que, aunque la escritura de pruebas requiere un cambio en la forma de desarrollar y tiempo adicional para diseñar y mantener los casos, con la práctica se evidenció que contribuyen a mejorar la calidad del código y a detectar errores de manera temprana. Esta evolución en la percepción, de carga inicial a utilidad percibida, esto es consistente con los hallazgos de (Dustin et al., 1999) sobre la curva de adopción de prácticas de automatización en equipos sin experiencia previa.

La dificultad de adopción más recurrente entre los participantes directos fue el *Quality Gate*. El Líder del Subproceso de Desarrollo identificó el cambio de mentalidad respecto a la manera de trabajar como la principal barrera. El Desarrollador de Software 2 señaló específicamente que, como desarrollador, no realizaba pruebas antes de integrar cambios ni consideraba las pruebas de regresión, lo que en algunos momentos le generó problemas con los *endpoints* al ser bloqueado por el *pipeline*. El Desarrollador de Software 1 apuntó a la escritura de pruebas unitarias como el aspecto más exigente por requerir tiempo adicional. Los tres participantes directos calificaron la fricción del proceso TO-BE como "manejable", mientras que el Líder de Gestión TICs la catalogó como adaptable con ajustes menores.

4.6.4. Sostenibilidad Percibida

La capacidad percibida del equipo para mantener y actualizar las pruebas automatizadas existentes sin acompañamiento externo obtuvo un promedio de 2.75 sobre 5, mientras que la capacidad para escribir nuevas pruebas para funcionalidades no cubiertas obtuvo 3.25. Estos valores reflejan un nivel de autonomía emergente pero aún frágil, coherente con un equipo que ha completado un primer piloto y reconoce tanto las capacidades ganadas como las brechas que persisten.

Las condiciones necesarias para el escalamiento identificadas por los participantes convergen en tres elementos: la formalización de un rol de QA Champion con dedicación parcial definida (mencionada por tres de cuatro participantes), documentación más detallada del marco implementado (mencionada por tres participantes) y capacitación adicional en técnicas de pruebas (mencionada por dos participantes). Este consenso es directamente coherente con las recomendaciones del proceso TO-BE presentadas en la sección 3.4.2.

La satisfacción general con el proceso implementado obtuvo un promedio de 4.5 sobre 5, con tres participantes en el nivel máximo (5 - Muy satisfecho) y uno en 4 - Satisfecho. Este resultado refleja una percepción consistentemente positiva del marco a través de todos los niveles organizacionales evaluados, tanto entre quienes participaron directamente en el piloto como en el nivel táctico-estratégico. Los cuatro participantes recomendaron implementar el proceso en los demás sistemas de la EPSI AIC, tres de forma definitiva y uno con ajustes previos.

4.7. Análisis Comparativo y Recomendaciones (D.5)

4.7.1. Síntesis Comparativa AS-IS vs. TO-BE

La Tabla 4.7 consolida los indicadores evaluados en las secciones anteriores, presentando el comparativo entre el estado inicial y el estado alcanzado al finalizar el piloto:

Tabla 4.7: Síntesis comparativa AS-IS vs. TO-BE por dimensión de evaluación. Fuente: elaboración propia.

Dimensión	Indicador	Estado AS-IS	Estado TO-BE	Resultado
Proceso	Cobertura automatizada Backend	0 %	41.1 %	Supera umbral 15 %
Proceso	Cobertura automatizada Frontend	0 %	21.9 %	Supera umbral 15 %
Proceso	Tiempo validación manual - Backend	15:39 min (medido)	3:41 min (autónomo)	Reducción 76.5 %
Proceso	Tiempo validación manual - Frontend	13:03 min (medido, 1 nav.)	15:24 min (3 nav.)	Mayor profundidad

Continúa en la siguiente página...

Tabla 4.7 – Continuación de la página anterior

Dimensión	Indicador	Estado AS-IS	Estado TO-BE	Resultado
Proceso	Ahorro acumulado - piloto completo	0 horas	∼33.3 horas	Cuantificado
Proceso	Total ejecuciones *pipeline*	0	100 (32 back + 68 front)	Marco operativo
Producto	Issues Reliability (Front)	78 issues	20 issues	Reducción 74.4 %
Producto	Issues Maintainability (Front)	550 issues	233 issues	Reducción 57.6 %
Producto	Duplicaciones - Frontend	5.8 %	3.1 %	Reducción 46.6 %
Producto	Detección de errores	Reactiva	Proactiva	Transformación
Continuidad	Quality Gate operativo	Inexistente	Activo - bloqueos	Verificado
Continuidad	Incidentes en producción	Sin registro	0 durante piloto	Sin incidentes
Cualitativo	Confianza en despliegues	2.75/5	3.25/5	Incremento +0.5
Cualitativo	Utilidad de reportes	N/A	4.75/5	Alta utilidad
Cualitativo	Pruebas como trabajo	N/A	4.0/5	Positiva
Cualitativo	Sostenibilidad autónoma	N/A	2.75/5	Req. ayuda
Cualitativo	Satisfacción general	N/A	4.5/5	Alta
Cualitativo	Recomendaría el proceso	N/A	4/4 participantes	Consenso

Los resultados consolidados evidencian que el marco cumplió su objetivo de demostrar la viabilidad técnica y organizacional de la automatización de pruebas en la EPSI AIC. Los indicadores de proceso muestran transformaciones medibles con datos objetivos; los indicadores de producto revelan reducciones significativas en *issues* de calidad del código; y los indicadores cualitativos señalan una adopción inicial positiva con brechas claras que orientan el escalamiento.

4.7.2. Factores que Influyeron en los Resultados

El análisis de los datos recolectados permite identificar tres factores que influyeron positivamente en los resultados del piloto:

- **Reutilización de herramientas existentes:** La selección de Newman como herramienta de

pruebas de integración aprovechó el uso previo de Postman por parte del equipo, reduciendo a cero la curva de aprendizaje en esa capa del marco. Este factor valida la decisión de diseño de la sección 3.6.2 de priorizar compatibilidad con herramientas conocidas por encima de sofisticación técnica.

- **Respaldo de infraestructura existente:** La disponibilidad de GitLab, Docker y un ambiente de *staging* que replica el 99 % del esquema de producción eliminó la necesidad de inversión adicional. El *pipeline* operó completamente sobre infraestructura local de la EPSI AIC, garantizando independencia de servicios externos y eliminando uno de los riesgos de implementación más frecuentes en organizaciones con recursos limitados.
- **Disposición organizacional positiva:** Los cuatro participantes recomendaron implementar el proceso en los demás sistemas, y los testimonios abiertos muestran reconocimiento genuino del valor del marco incluso entre quienes experimentaron mayor fricción durante la adopción.

El factor que generó mayor tensión fue la curva de aprendizaje asociada al *Quality Gate*, que se manifestó en los bloqueos documentados durante el piloto y en las respuestas cualitativas que señalaron este mecanismo como el elemento de mayor fricción inicial. Este factor fue identificado como riesgo en el plan de transición de la sección 3.4.4, por lo que su materialización era esperada y manejable.

4.7.3. Recomendaciones para el Escalamiento

A partir de los hallazgos de la evaluación se formulan las siguientes recomendaciones para el escalamiento sostenible del marco en la EPSI AIC:

- **Formalizar el rol de QA Champion:** El consenso de tres de los cuatro participantes sobre la necesidad de un rol formal con dedicación parcial confirma que la sostenibilidad del marco depende de contar con una persona con tiempo asignado para guiar al equipo, mantener el repositorio de pruebas y actualizar la documentación. Se recomienda formalizar este rol con una dedicación mínima del 20 % de la carga semanal, tal como fue previsto en la sección 3.4.2.
- **Implementar un programa de capacitación progresivo:** La brecha de conocimiento en técnicas de diseño de pruebas fue identificada como condición habilitante por todos los actores técnicos. Se recomienda diseñar un plan de formación escalonado que inicie con taller de pruebas unitarias básicas, avance hacia diseño de casos ISTQB y culmine con pruebas E2E, distribuido en *sprints* consecutivos para minimizar el impacto en la capacidad productiva del equipo.
- **Calibrar los umbrales del Quality Gate con el equipo:** Los bloqueos documentados durante el piloto generaron fricción por la falta de comprensión inicial de los criterios de aceptación. Se recomienda realizar una sesión de calibración participativa al iniciar cada nuevo sistema, en la que el equipo defina y comprenda los umbrales antes de activar el *Quality Gate*, reduciendo así la fricción en los primeros ciclos.

- **Escalar la automatización al siguiente sistema priorizado:** Según el orden de priorización de la Tabla 18, el siguiente sistema a abordar es el de Gestión de PQRSD. Se recomienda aplicar la misma metodología del piloto aprovechando la arquitectura del marco ya instalada y la experiencia acumulada por el equipo.
- **Fortalecer los mecanismos de visibilidad hacia el nivel estratégico:** Aunque la satisfacción general fue alta (4.5/5), la consolidación del proceso a largo plazo requiere que los indicadores de calidad sean visibles de forma sistemática hacia el nivel táctico-estratégico sin depender de comunicaciones informales. Se recomienda implementar un reporte periódico automático de indicadores — cobertura, *issues*, estado del *Quality Gate* — generado desde SonarQube o GitLab y dirigido al nivel táctico, que institucionalice la visibilidad del valor del proceso conforme se escale a nuevos sistemas.

4.8. Resumen del Capítulo

Este capítulo documentó la evaluación del impacto del marco de aseguramiento de calidad implementado en la EPSI AIC, correspondiente al eje de Validación del Tratamiento del paradigma DSR. La evaluación trianguló cuatro fuentes de evidencia: mediciones cronometradas de validación manual, métricas objetivas del *pipeline* de CI/CD, análisis comparativo de *dashboards* de SonarQube y evaluación cualitativa del equipo mediante instrumento post-implementación aplicado a cuatro actores clave.

Los resultados confirmaron que el piloto cumplió su objetivo de demostrar la viabilidad técnica y organizacional del marco. En los indicadores de proceso, la cobertura automatizada pasó del 0 % al 41.1 % en el backend y al 21.9 % en el frontend, superando el umbral del *Quality Gate* en ambos componentes; la validación de regresión del backend se redujo un 76.5 % respecto a la medición manual (de 15:39 min a 3:41 min), y el marco generó un ahorro acumulado estimado de 33.3 horas de esfuerzo manual en 100 ejecuciones de *pipeline* durante el periodo del piloto. En los indicadores de producto, el análisis estático reveló reducciones del 74.4 % en *issues* de *Reliability* y del 57.6 % en *issues* de *Maintainability* en el frontend, y los logs de GitLab confirmaron que el *Quality Gate* bloqueó despliegues por situaciones reales de incumplimiento de criterios de calidad. En los indicadores cualitativos, el equipo percibió mejoras en la confianza al desplegar (promedio +0.5 puntos), alta utilidad en los reportes del *pipeline* (4.75/5) y una satisfacción general de 4.5/5 con el proceso implementado, reconociendo al mismo tiempo que la sostenibilidad autónoma requiere acompañamiento mediante capacitación y la formalización del rol de QA Champion.

Las cinco recomendaciones de escalamiento formuladas responden directamente a las brechas identificadas y proporcionan una hoja de ruta concreta para que la EPSI AIC continúe evolucionando hacia niveles superiores de madurez en sus procesos de aseguramiento de calidad.

Capítulo 5

Conclusiones

5.1. Conclusiones

Este proyecto diseñó e implementó un marco de aseguramiento de calidad basado en pruebas automatizadas para los sistemas de información misionales de la EPSI AIC, respondiendo a una situación de partida caracterizada por 0 % de cobertura automatizada, dependencia total en validaciones manuales ejecutadas por un único responsable y ausencia completa de métricas de calidad. Los resultados obtenidos permiten formular las siguientes conclusiones, organizadas en correspondencia con los cinco objetivos específicos del proyecto.

Sobre el diseño del proceso de aseguramiento de calidad (O.E. 1). El diagnóstico de la Fase A evidenció que la EPSI AIC se encontraba en el nivel TMMi 1 (inicial), con prácticas de pruebas completamente informales, sin documentación, sin roles definidos y sin métricas de ningún tipo. A partir de ese estado, se diseñó un proceso formal de aseguramiento de calidad *TO-BE* adaptado al contexto organizacional y a las restricciones tecnológicas de la entidad. El proceso incorporó una estructura de roles con matriz RACI, políticas explícitas de calidad, un flujo de trabajo metodológicamente agnóstico, es decir, diseñado para adaptarse al modelo de desarrollo real de la organización sin condicionar su adopción a ningún *framework* específico, y un mecanismo de *Quality Gate* que actúa como compuerta de control obligatoria antes de cada despliegue. La experiencia del piloto sugiere que, en organizaciones con madurez TMMi 1, un proceso gradual y contextualizado, diseñado a partir de un diagnóstico participativo, favorece la adopción frente a la imposición de estándares genéricos. Esta observación, consistente con los resultados obtenidos, se deriva de un piloto acotado y requiere validación en implementaciones posteriores.

Sobre la priorización de funcionalidades (O.E. 2). La aplicación de una matriz de priorización multicriterio, basada en el riesgo para la continuidad del servicio, la criticidad normativa según la Resolución 866 de 2021 (Ministerio de Salud y Protección Social, 2021) y la frecuencia de uso, permitió identificar el Módulo de Referencia 24/7 como el componente de mayor riesgo y, por tanto, el candidato prioritario para el piloto. Este resultado fue validado por el equipo de la EPSI AIC y resultó coherente con la percepción operativa de los desarrolladores. En el contexto del piloto, la priorización sistemática permitió concentrar el esfuerzo en el componente de mayor riesgo y obtener resultados perceptibles en el corto plazo, lo que sugiere que, en organizaciones con recursos limitados, constituye una condición habilitante para la viabilidad del aseguramiento de calidad más que un formalismo académico.

Sobre el diseño de casos de prueba con técnicas de caja negra (O.E. 3). El diseño formal de casos de prueba aplicando técnicas ISTQB, es decir, particiones de equivalencia, tablas de decisión y arreglos ortogonales según la naturaleza de cada funcionalidad, produjo un conjunto de

casos con trazabilidad completa desde los requisitos hasta la evidencia de ejecución. Esta trazabilidad, ausente en el estado *AS-IS*, permite rastrear de manera sistemática la relación entre un caso de prueba, el requisito que verifica y el resultado observado. La selección de la técnica adecuada según el tipo de lógica a verificar fue determinante para la eficiencia del diseño: las tablas de decisión resultaron especialmente adecuadas para los flujos de autorización con múltiples condiciones, mientras que las particiones de equivalencia fueron suficientes para las validaciones de formato y rangos de entrada.

Sobre la implementación del marco de automatización (O.E. 4). El marco implementado integró cuatro capas de verificación: pruebas unitarias de *backend* con Pytest, pruebas de integración de APIs con Newman, pruebas unitarias de *frontend* con Vitest y pruebas *end-to-end* con Playwright, articuladas mediante un *pipeline* de CI/CD en GitLab con análisis estático de SonarQube como *Quality Gate*. La selección de herramientas que ya formaban parte del ecosistema tecnológico de la organización, en particular el uso de Newman como extensión natural de Postman, herramienta ya conocida por el equipo, redujo la curva de aprendizaje en esa capa a prácticamente cero y aceleró la adopción. La arquitectura resultante es modular, opera completamente sobre infraestructura local de la EPSI AIC sin dependencia de servicios externos, y está documentada en un repositorio público con guías de instalación, configuración y mantenimiento.

Sobre la evaluación del impacto (O.E. 5). La evaluación cuasi-experimental con diseño pre/post implementación demostró que el marco cumplió su objetivo de evidenciar la viabilidad técnica y organizacional de la automatización de pruebas en la EPSI AIC. Los principales hallazgos cuantitativos fueron:

- Los indicadores de producto, medidos conforme a las definiciones del ISTQB, arrojaron una correctitud del 100 % (29 de 29 casos de prueba ejecutados de forma exitosa) y una confiabilidad del 89.7 % (complemento de 3 no conformidades detectadas sobre el total de 29 casos ejecutados).
- La cobertura automatizada pasó del 0 % al 41.1 % en el *backend* y al 21.9 % en el *frontend*, superando en ambos casos el umbral mínimo del 15 % definido en el *Quality Gate*.
- El tiempo de validación de regresión del *backend* se redujo un 76.5 %, de 15:39 minutos en el proceso manual a 3:41 minutos en el *pipeline* automatizado.
- El marco generó un ahorro acumulado de aproximadamente 33.3 horas de esfuerzo manual de validación durante el periodo del piloto, equivalentes a 4.2 jornadas laborales.
- El análisis estático reveló reducciones del 74.4 % en *issues* de *Reliability* y del 57.6 % en *issues* de *Maintainability* en el componente *frontend*.
- El *Quality Gate* bloqueó la promoción de artefactos en tres ocasiones documentadas por incumplimiento real de criterios de calidad, transformando la detección de errores de un modelo reactivo a uno proactivo.

En el plano cualitativo, la satisfacción general del equipo con el proceso alcanzó un promedio de 4.5 sobre 5 ($n = 4$ participantes), y la totalidad de ellos recomendó su implementación en los demás sistemas de la entidad. Estos resultados deben interpretarse a la luz de las limitaciones del estudio, es decir, un piloto acotado a un módulo, con duración limitada y una evaluación de disponibilidad de carácter declarativo, por lo que constituyen evidencia de la viabilidad del marco más que de un impacto generalizable.

Este proyecto aportó un caso documentado de implementación de prácticas formales de aseguramiento de calidad en una organización de salud con madurez TMMi inicial, contexto escasamente representado en la literatura existente, que tiende a concentrarse en organizaciones con equipos de QA establecidos y recursos suficientes para adoptar estándares completos. Los hallazgos confirman que la combinación de *Design Science Research* como paradigma metodológico con un enfoque de implementación piloto incremental resulta adecuada para generar conocimiento verificable en organizaciones con estas características. El artefacto generado, es decir, el marco de automatización con su arquitectura, documentación técnica y repositorio público, constituye un punto de partida transferible para iniciativas similares en organizaciones del sector salud con restricciones comparables.

5.2. Trabajos futuros

A partir de los resultados del piloto se identifican cinco líneas de trabajo prioritarias para consolidar y escalar el proceso:

- **Escalamiento al siguiente sistema priorizado:** El siguiente sistema a abordar, según la matriz de priorización de la Fase B, es el de Gestión de PQRSD. La metodología, la arquitectura del marco y la experiencia acumulada por el equipo están disponibles para iniciar ese proceso sin partir desde cero.
- **Formalización del rol de QA Champion:** Tres de los cuatro participantes de la evaluación identificaron este rol como condición necesaria para la sostenibilidad autónoma del marco. Su institucionalización, con dedicación mínima del 20 % de la carga semanal, es la acción organizacional de mayor impacto en el corto plazo.
- **Programa de capacitación progresivo en diseño de pruebas:** La brecha de conocimiento en técnicas de diseño de casos fue la barrera técnica más recurrente. Se recomienda un plan escalonado que avance de pruebas unitarias básicas hacia técnicas ISTQB y pruebas *end-to-end*, distribuido en ciclos consecutivos.
- **Implementación de monitoreo objetivo de disponibilidad:** La ausencia de métricas formales de disponibilidad limitó la evaluación de confiabilidad a evidencia declarativa. Implementar indicadores como el Tiempo Medio Entre Fallos (MTBF) y el Tiempo Medio de Reparación (MTTR) permitirá cuantificar el impacto del marco sobre la continuidad operativa con evidencia empírica sostenida.

- **Visibilidad estratégica automatizada:** Un reporte periódico generado desde SonarQube o GitLab, que consolide cobertura, estado del *Quality Gate* e *issues* por sistema, institucionalizaría la visibilidad del valor del proceso hacia el nivel directivo sin depender de comunicaciones informales.

5.3. Lecciones aprendidas

La ejecución del proyecto en un contexto organizacional real dejó aprendizajes que trascienden los resultados numéricos. El primero de ellos tiene que ver con el punto de partida: construir la línea base *AS-IS* desde las voces del propio equipo, y no únicamente desde la observación técnica externa, fue determinante para que el proceso *TO-BE* fuera adoptado como respuesta a una necesidad reconocida y no como imposición. Cuando las personas se reconocen en el diagnóstico, la resistencia al cambio disminuye de manera significativa.

En cuanto a la selección de herramientas, la experiencia mostró que priorizar compatibilidad con el ecosistema existente sobre sofisticación técnica produce mejores resultados en equipos pequeños con alta carga operativa. El uso de Newman como extensión natural de Postman es el ejemplo más claro: eliminó la curva de aprendizaje en esa capa y permitió obtener resultados visibles desde las primeras semanas. Una herramienta desconocida, por más potente que sea, compite con el tiempo productivo del equipo.

Respecto al *Quality Gate*, los bloqueos documentados durante el piloto generaron tensión inicial, pero el análisis posterior confirmó que todos correspondían a situaciones reales de incumplimiento. La lección no es suavizar el mecanismo sino anticipar el momento de mayor fricción: una sesión de calibración participativa, en la que el equipo comprende los criterios antes de que el sistema los aplique por primera vez, reduce considerablemente la resistencia de los primeros ciclos.

Sobre la cobertura, el 41.1 % en *backend* y el 21.9 % en *frontend* no representan niveles de excelencia sino puntos de partida verificables. Establecer umbrales alcanzables —y no aspirar a porcentajes elevados desde el inicio— fue lo que hizo posible que el piloto generara resultados perceptibles en el corto plazo y mantuviera la motivación del equipo durante la implementación. Finalmente, en organizaciones con alta rotación de personal, integrar la generación de documentación técnica como actividad formal de la Fase C, y no como tarea pendiente para después, resultó tan determinante para la sostenibilidad del marco como la propia implementación de las pruebas.

5.4. Reflexión final

Este proyecto demostró que es posible instalar prácticas formales de aseguramiento de calidad en organizaciones de salud con baja madurez en *testing*, sin grandes inversiones ni equipos especializados, cuando la implementación parte de un diagnóstico honesto, se enfoca en los componentes de mayor riesgo y aprovecha la infraestructura existente. La EPSI AIC atiende a comunidades indígenas del Cauca cuyo acceso a servicios de salud depende de la estabilidad de sus sistemas de información. Cada despliegue bloqueado por el *Quality Gate* antes de llegar a producción es una

contribución concreta a esa continuidad, y esa conexión entre calidad del software y bienestar de una población vulnerable es la razón más sólida para continuar escalando el trabajo iniciado.

Bibliografía

Ambler, S. W. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. John Wiley & Sons.

Ambur, N. and Devaraj, N. (2018). [escribe aquí el título del artículo sobre devops o pruebas automatizadas]. *[Escribe aquí el nombre de la revista o conferencia]*, [Volumen, si aplica]([Número, si aplica]):[Páginas, si aplica].

Asociación Indígena del Cauca EPS-I (2025). *Manual de Armonía: Estructura y Procesos*. EPSI AIC, Popayán, Colombia. Documento institucional interno.

Baqar, M. and Khanda, S. (2024). Challenges in traditional test case generation and validation. *Journal of Software Testing and Quality Assurance*, 12(1):1–12.

Beck, K. (2003). *Test-Driven Development: By Example*. Addison-Wesley.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5):61–72.

Cook, T. D. and Campbell, D. T. (1979). *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin.

Denzin, N. K. and Lincoln, Y. S. (2011). *The SAGE Handbook of Qualitative Research*. SAGE Publications, 4 edition.

Dustin, E., Rashka, J., and Paul, J. (1999). *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley.

Fewster, M. and Graham, D. (2019). *Software Test Automation: Effective Use of Test Execution Tools*. Addison-Wesley, 2nd edition.

Garousi, V., Felderer, M., and Mäntylä, M. V. (2019). The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. *Information and Software Technology*, 106:101–121.

Hamburg, M. and Roman, A. (2025). Black-box testing for practitioners: A case of the new istqb test analyst syllabus. In *2025 IEEE Conference on Software Testing, Verification and Validation (ICST 2025)*, pages 634–645. Institute of Electrical and Electronics Engineers Inc.

Healthcare Information and Management Systems Society (2019). Measuring the impact of software testing in healthcare systems. https://www.himss.org.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75–105.

Hillah, L. M., De Rosa, F., Maesano, A.-P., and Kordon, F. (2016). Service functional testing automation with intelligent scheduling and planning. In *2016 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 135–138. IEEE.

International Organization for Standardization (2018). Iso 31000:2018 risk management — guidelines. Technical report, ISO.

International Software Testing Qualifications Board (2023). Certified tester foundation level syllabus v4.0.1. Accedido en 2026.

ISO (2011). Iso/iec 25010:2011 - systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models. Technical report, International Organization for Standardization.

ISO/IEC/IEEE (2013). Iso/iec/ieee 29119-3:2013 - software and systems engineering – software testing – part 3: Test documentation. Technical report, International Organization for Standardization.

Kaner, C., Falk, J., and Nguyen, H. Q. (1999). *Testing Computer Software*. Wiley, 2nd edition.

Khankhoje, R. (2022). Dominando la automatización de pruebas: superando las brechas para un control de calidad sin interrupciones. *Revista de Inteligencia Artificial y Computación en la Nube*, 1(3):1–6.

Knight, J. C. and Wika, K. G. (1995). Software safety in medical applications. *Journal of Image Guided Surgery*, 1(3):121–132.

Leite, L., Pinto, G., Kon, F., and Meirelles, P. (2021). The organization of software teams in the quest for continuous delivery: A grounded theory approach. *Information and Software Technology*, 139:106672.

Meszaros, G. (2007). *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley.

Microsoft (2024). Playwright: Fast and reliable end-to-end testing for modern web apps. Accedido en 2024.

Ministerio de Salud y Protección Social (2021). Resolución 866 de 2021: Por la cual se adoptan lineamientos de seguridad de la información en salud. Diario Oficial de Colombia. Disponible en: https://www.minsalud.gov.co/Normatividad_Nuevo/Resolucion%20866%20de%202021.pdf.

Mintzberg, H. (1979). *The Structuring of Organizations: A Synthesis of the Research*. Prentice-Hall.

Myers, G. J., Sandler, C., and Badgett, T. (2004). *The Art of Software Testing*. John Wiley & Sons, 2nd edition.

NIST (2017). Nist special publication 800-115 - technical guide to information security testing and assessment. Technical report, National Institute of Standards and Technology.

North, D. (2006). Introducing bdd. *Better Software Magazine*.

Okken, B. (2022). *Python Testing with pytest: Simple, Rapid, Effective, and Scalable*. Pragmatic Bookshelf, 2nd edition.

Pressman, R. S. and Maxim, B. (2020). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.

Saxena, A. (2025). Ai and machine learning in software testing: Trends and challenges. *International Journal of Software Engineering Research*, 14(2):45–59.

Schwaber, K. and Sutherland, J. (2020). The scrum guide: The definitive guide to scrum: The rules of the game. https://scrumguides.org.

Shahin, M., Babar, M. A., and Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Sommerville, I. (2016). *Software Engineering*. Pearson, 10th edition.

Urbano Guevara, E. and Valencia García, D. M. (2023). Artefacto para evaluar la madurez del proceso de pruebas de software basado en el modelo tmmi. Master's thesis, Pontificia Universidad Javeriana Cali.

Viana, R., Oliveira, F., and Chaves, L. (2024). Streamlining test execution: A case study on the use of automated testing to enhance productivity. In *2024 13th International Conference on Software and Information Engineering (ICSIE)*, pages 33–37. ACM.

Vitest Team (2024). Vitest guide. Accedido en 2024.

Yin, R. K. (2018). *Case Study Research and Applications: Design and Methods*. Sage Publications, 6th edition.

Zamli, K. Z., Kendall, E. A., and Isa, N. A. (2007). Automation issues in software testing: A review. In *Proceedings of the 2007 International Conference on Software Engineering Advances (ICSEA)*, pages 54–60. IEEE.