ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Desarrollo de software seguro basado en principios de seguridad por diseño (security by design) aplicados en una fábrica de software en guayaquil

Proyecto de Titulación

Previa a la obtención del Título de:

Magister en Seguridad Informática

Presentado por:

Ing. Vicente Fernando Arreaga Figueroa

Ing. Juan Carlos Chamorro Arias

Guayaquil – Ecuador

Año: 2025

Agradecimiento

A mis maestros, por compartir sus amplios conocimientos con ética y profesionalismo.

A mi tutor de tesis, Ing. Lenin Freire Cobo, por su valiosa guía y asesoramiento en las diversas etapas del desarrollo del proyecto.

A mi amigo y compañero de tesis, Juan Carlos Chamorro por su constante respaldo y colaboración durante el desarrollo de este proyecto. Su compromiso y trabajo en equipo fueron fundamentales para superar los desafíos y alcanzar los resultados deseados.

Ing. Vicente Fernando Arreaga.

Agradecimiento

En primer lugar, a Dios, mi familia y amigos, quienes han sido mi principal fuente de apoyo y motivación. Sus palabras alentadoras y comprensión han sido fundamentales durante este nuevo desafiante viaje académico.

Mi reconocimiento también se extiende a mis compañeros de clase y colaboradores de investigación en especial a mi compañero de tesis, Vicente Fernando Arreaga. Trabajar codo a codo con ustedes ha enriquecido mi experiencia académica y ha contribuido de manera significativa al éxito de este proyecto.

A mi tutor de tesis, Ing. Lenin Freire, por su valiosa aportación de conocimientos y experiencias en las diversas etapas no tan solo del desarrollo del proyecto sino también en los conocimientos compartidos en su cátedra.

Este logro no es solo mío, sino de todos aquellos que, han sido parte de este emocionante viaje académico. Gracias a todos por ser parte de este capítulo en mi vida.

Ing. Juan Carlos Chamorro

Dedicatoria

A Dios por ser la guía en esta etapa académica y darme la fortaleza necesaria para continuar aun en los momentos más difíciles.

A mis queridos padres Vicente y Alexandra, mi más sincero agradecimiento, ya que gracias a su amor incondicional y apoyo inquebrantable no me permitieron rendirme. A través de su ejemplo, dedicación y sacrificio me han inspirado a perseguir mis metas y sueños con

determinación, esfuerzo y trabajo duro.

A mis queridas hermanas, Kelvin y Lourdes, por su infinita paciencia y amor incondicional. Su apoyo constante ha sido un recordatorio invaluable de la importancia de celebrar los logros junto a la familia.

En memoria de mi querida abuela Margarita.

Ing. Vicente Fernando Arreaga.

Dedicatoria

A Dios y mis padres, Juan Antonio y Susana, cuyo amor incondicional y sacrificios han sido mi fuente constante de inspiración. Sus valores y correcciones han sido la fuerza impulsora detrás de cada logro en mi vida.

A mi esposa, Soledad, por ser mi compañera constante en este viaje. Tu amor, paciencia, aliento y comprensión han hecho posible superar los desafíos académicos.

A mi pequeña hija, Simoné, quien, a pesar de mi ausencia en muchos momentos, siempre han sido mi razón para esforzarme por ser mejor.

Esta tesis es el resultado de la contribución de muchas personas que han dejado una marca permanente en mi viaje académico. A todos ustedes, mi más profundo agradecimiento y dedicación.

En memoria de mi padre, Juan Antonio.

Ing. Juan Carlos Chamorro.

Declaración Expresa

Nosotros Ing. Vicente Fernando Arreaga y Juan Carlos Chamorro acordamos y reconocemos que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá a los autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique a los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 11 de septiembre del 2025.

Firmado electrónicamente por:
JUAN CARLOS CHAMORRO ARIAS
Validar únicamente con FirmaEC

Ing. Juan Carlos Chamorro Arias

Firmado electrónicamente por:
VICENTE FERNANDO ARREAGA FIGUEROA
Validar únicamente con FirmaEC

Ing. Vicente Fernando Arreaga Figueroa

Evaluadores

Firmado electrónicamente por:
LENIN EDUARDO FREIRE COBO
Validar únicamente con FirmaEC

MSc. Lenin Eduardo Freire Cobo
Tutor

Firmado electrónicamente por:
JUAN CARLOS GARCIA PLUA
Validar únicamente con FirmaEC

MSc. Juan Carlos García Plúa
Revisor

Resumen

Este artículo propone un marco metodológico de desarrollo de software que se basa en principios de Seguridad y Diseño. Su objetivo es garantizar que las prácticas seguras se integren efectivamente en cada fase del ciclo de vida de un programa informático. El argumento aquí es que, al integrar la seguridad en una etapa temprana y estructurarla de manera ordenada, podemos evitar que nuestros sistemas se vuelvan vulnerables a errores graves. Esto ahorrará dinero porque los errores que causan tanto trabajo hoy simplemente se posponen hasta mañana o el año siguiente, cuando finalmente haya fondos disponibles. A menudo se sostiene la tesis de que, a través de este enfoque, la resiliencia de un sistema frente a amenazas cotidianas se establece de mejor manera.

La investigación se justifica por la necesidad creciente de fortalecer la seguridad en aplicaciones desarrolladas en entornos profesionales, especialmente en fábricas de software. Para el desarrollo del proyecto se utilizó un enfoque cualitativo-aplicado, con base en el modelo OWASP SAMM, lineamientos de ISO/IEC 27001 y herramientas como OWASP ZAP y SonarQube. Se diseñó y ejecutó un piloto técnico en un entorno controlado, donde se evaluaron buenas prácticas integradas en las fases de diseño, codificación y prueba. Los resultados mostraron una mejora sustancial en la identificación y mitigación de vulnerabilidades desde etapas tempranas. Se concluye que aplicar principios de Seguridad por Diseño de manera estructurada favorece un desarrollo de software más seguro, sostenible y alineado con normativas vigentes.

Palabras clave: desarrollo seguro, ciclo de vida, OWASP SAMM, vulnerabilidades, normativa técnica.

Abstract

This article proposes a software development methodological framework based on Security and Design principles. Its goal is to ensure that secure practices are effectively integrated into every phase of a software program's lifecycle. The argument here is that by integrating security early and structuring it in an orderly manner, we can prevent our systems from becoming vulnerable to serious errors. This will save money because the errors that cause so much work today are simply postponed until tomorrow or the following year, when funding is finally available. It is often argued that, through this approach, a system's resilience to everyday threats is better established.

The research is justified by the growing need to strengthen security in applications developed in professional environments, especially in software factories. A qualitative-applied approach was used for the development of the project, based on the OWASP SAMM model, ISO/IEC 27001 guidelines, and tools such as OWASP ZAP and So-narQube. A technical pilot was designed and executed in a controlled environment, where best practices integrated into the design, coding, and testing phases were evaluated. The results showed a substantial improvement in the identification and mitigation of vulnerabilities from early stages. It is concluded that applying Security by Design principles in a structured manner favors more secure, sustainable software development that is aligned with current regulations.

Keywords: secure development, life cycle, OWASP SAMM, vulnerabilities, technical standards**.**

.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

1.1 Introducción

La rápida evolución de las tecnologías digitales ha hecho que los sistemas informáticos sean cada vez más complejos. A medida que la complejidad de estos sistemas ha aumentado, también lo ha hecho el riesgo correspondiente (inseguridad) que traen consigo.

En la actualidad, el desarrollo de software seguro se ha vuelto cada vez más importante. Esto es necesario porque la frecuencia y sofisticación de los ciberataques están aumentando rápidamente en todos los frentes. Citando informes internacionales, muchas vulnerabilidades explotadas en sistemas de producción provienen de malas prácticas durante las fases iniciales del ciclo de vida del desarrollo de software (SDLC) [1].

En general, lo que rara vez se aborda en los programas educativos tradicionales es que por cada defecto de software o "bug", siempre hay una serie de otras características innecesarias cuya presencia solo añade riesgo a los sistemas.

En el caso de América Latina y especialmente Ecuador, es habitual implementar muchas aplicaciones y plataformas sin considerar inicialmente fundamentos como la seguridad. Esto aumenta tanto el peligro de ser golpeado por amenazas que provienen directamente de la red, como inyecciones de código o errores de autenticación; y también conduce a pérdidas económicas, daño a la reputación de nuestras organizaciones, multas por violar requisitos regulatorios [2].

La mayoría de los incidentes de seguridad son causados ya sea por un mal diseño o por no seguir una política de seguridad estructurada desde el inicio de un proyecto. Por ejemplo, esta deficiencia es grave porque corregir agujeros de seguridad u otros defectos en fases tardías, especialmente después de que el software ha sido puesto en entornos de producción, es un trabajo costoso y difícil, que tiene un efecto negativo en todo: eficiencia de producción, calidad en el producto final [3].

Respaldando el concepto de SBD en este tema, sus elementos esenciales no se limitan a los principios mencionados. Este enfoque construye sistemáticamente requisitos de seguridad en cada etapa del proyecto de software. Mientras tanto, bajo tal enfoque podemos asegurarnos de que las soluciones digitales nazcan con sus requisitos de seguridad completamente integrados en su arquitectura original; mientras que luego no se añaden por necesidad o como una solución reactiva a algo que se pasó por alto posteriormente [4].

El objetivo de este estudio es desarrollar un marco sistemático de Seguridad por Diseño para el desarrollo de software. El propósito es reducir los riesgos comunes de seguridad y hacer que los sistemas sean más resistentes tanto a amenazas intencionales internas como externas.

Por lo tanto, examinaremos las metodologías de desarrollo más populares tanto en campos ágiles como no ágiles para descubrir algo sobre sus potenciales vulnerabilidades, y también lo que ofrecen en términos de fortalezas al incorporar controles de seguridad en una etapa temprana del SDLC. También fomentarán las mejores prácticas que se alinean con estándares reconocidos, como OWASP SAMM/ISO/IEC 27001 para organizaciones de diferentes tamaños, y especialmente aquellas en mercados emergentes donde las empresas de servicios informáticos están en auge o fábricas locales que forman parte de un grupo más grande [5].

El presente proyecto aborda la tecnología de Seguridad por Diseño utilizando, y con asesoramiento técnico, herramientas automatizadas. Predice que se pueden lograr mejoras significativas tanto en robustez como en el número de vulnerabilidades críticas.

Para responder a la demanda, algunos de los elementos metodológicos incluyen:

- Análisis de Brechas de Seguridad;
- Realización de un prototipo técnico diseñado en algún entorno de prueba;

- Críticas a nivel de módulo utilizando herramientas OWASP ZAP/SonarQube, sin olvidar realizar auditorías de código y verificaciones a nivel de aplicación.

La problematica radica en el hecho de que pocas corporaciones, ya sean públicas o privadas, pueden sostener su desarrollo continuo de sistemas seguros. A pesar de una amplia gama de recursos técnicos disponibles para ellas, muchas empresas todavía practican solo en una medida limitada la identificación temprana de riesgos, la priorización de riesgos o la cuantificación de requisitos de seguridad y no siempre se toman medidas sistemáticas para la conformidad con los estándares. Esto se refleja en la prevalencia de vulnerabilidades como inyecciones SQL, debilidades en la autenticación de usuarios, exposición de datos sensibles y fallas de configuración, entre otras [6].

Además, la ausencia de una cultura de seguridad en el ciclo de vida del desarrollo de software genera dificultades para cumplir con marcos normativos vigentes. En el caso ecuatoriano, la entrada en vigencia de la Ley Orgánica de Protección de Datos Personales (LOPDP) ha creado nuevas exigencias legales para el tratamiento de información sensible. El incumplimiento de estas normativas puede conllevar sanciones administrativas, pérdida de licencias o responsabilidad civil y penal para las organizaciones [7].

Ante este escenario, el enfoque propuesto no solo pretende optimizar los aspectos técnicos del desarrollo de software, sino también contribuir a una transformación cultural en el proceso de construcción de soluciones tecnológicas. La introducción de prácticas de Seguridad por Diseño no solo mejorará la calidad del producto, sino que también evitará fallos importantes que de otro modo ocurrirían en trabajos de mantenimiento a largo plazo o durante las fases de postproducción. Además, al adoptar tales diseños desde sus orígenes, se

ayuda a mejorar la confianza del usuario final para confiar en que sus datos pueden estar completamente protegidos.

Se espera que el desarrollo de los resultados de esta investigación actúe como una base metodológica para implementar programas de desarrollo seguro reproducibles en entornos empresariales serios, y así promover una cultura proactiva que sea responsable (legalmente) y que busque el progreso en la ingeniería de software.

1.2 Descripción del Problema

En la actualidad, las organizaciones que desarrollan software enfrentan una creciente presión para ofrecer productos digitales funcionales, rápidos y seguros. Sin embargo, la seguridad en el desarrollo de software continúa siendo un desafío persistente, particularmente en los entornos donde se prioriza la velocidad de entrega sobre la robustez del producto. La mayoría de los incidentes de seguridad en aplicaciones empresariales y de consumo se originan por no integrar controles de seguridad desde las primeras fases del ciclo de vida del software (SDLC), lo cual genera vulnerabilidades que son detectadas demasiado tarde: durante las pruebas finales, en producción o incluso tras incidentes reales [3].

Uno de los factores críticos que agudiza esta problemática es la baja adopción de prácticas de Seguridad por Diseño en proyectos reales. A pesar de la disponibilidad de modelos y principios internacionalmente reconocidos —como los ocho principios de Saltzer y Schroeder— muchas organizaciones no los integran debido a la ausencia de cultura de seguridad, desconocimiento técnico o falta de recursos especializados [8]. Esta situación también se replica en la organización objeto de estudio, una fábrica de software de alcance nacional, cuya identidad se reserva por razones de confidencialidad. Esta entidad desarrolla productos para sectores sensibles, incluyendo banca, servicios públicos y plataformas de consumo.

En esta fábrica, los proyectos son liderados principalmente por desarrolladores que deben asumir decisiones críticas de diseño sin contar necesariamente con una guía formal de seguridad o el acompañamiento de especialistas. A esto se suman desafíos como la presión por cumplir cronogramas ajustados, el uso de metodologías ágiles sin criterios de seguridad explícitos, y la implementación de herramientas de desarrollo sin configuraciones seguras por defecto. Como se muestra en la Tabla 1, persisten brechas comunes como inyecciones SQL, autenticación débil o falta de cifrado, cuya presencia es especialmente crítica en aplicaciones de misión crítica.

Tabla 1.
Principales vulnerabilidades frecuentes en desarrollos sin seguridad por diseño

Vulnerabilidad	Descripción	Consecuencia
Inyección SQL	Manipulación de consultas a bases de datos	Acceso o modificación no autorizada de datos
Autenticación débil	Uso de contraseñas predecibles o sin doble factor	Suplantación de identidad
Falta de cifrado	Transmisión de datos sensibles sin protección	Exposición de información privada
Manejo incorrecto de sesiones	Tokens mal gestionados o sin expiración	Secuestro de sesión
Exposición de errores	Mensajes técnicos detallados mostrados al usuario	Divulgación de información interna

Nota: Elaboración propia con base en OWASP Top 10.

Este conjunto de deficiencias ha generado problemas medibles en la organización, desde reprocesos técnicos y pérdida de eficiencia, hasta quejas de usuarios, daño reputacional y exposición a sanciones regulatorias. La percepción de calidad del software entregado se ve

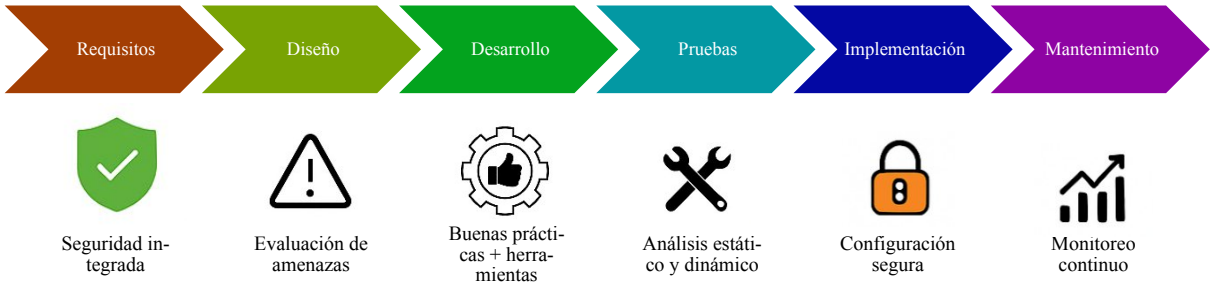
afectada, lo que reduce la confianza de los clientes y aumenta la rotación de proyectos a causa de los sobrecostos derivados de corregir errores de seguridad en etapas avanzadas.

El problema es especialmente relevante porque, aunque las metodologías ágiles como Scrum y Programación Extrema (XP) permiten iteraciones rápidas y tolerancia a cambios, no priorizan de forma explícita la seguridad como atributo de calidad. Diversos autores han identificado esta debilidad estructural como una barrera para la madurez en desarrollo seguro [9].

Como se observa en la Figura 1(a) un ciclo de vida de desarrollo tradicional tiende a relegar la seguridad a las fases finales, lo cual incrementa los riesgos y costos asociados. En contraste, la Figura 1(b) muestra una propuesta basada en Seguridad por Diseño, donde se integran controles desde los requisitos hasta el mantenimiento, alineando el proceso con estándares como OWASP SAMM e ISO/IEC 27001.

Figura 1.
Comparación entre un ciclo de vida de desarrollo tradicional y uno con Seguridad por Diseño

(a) Ciclo de vida de desarrollo tradicional (SDLC clásico)



(b) Ciclo de vida con seguridad por diseño

Nota: Elaboración propia con base en OWASP SAMM y Microsoft SDL.

Por tanto, se identifica como problema central la ausencia de una estrategia estructurada para integrar principios de seguridad desde las fases tempranas del desarrollo, lo cual expone a las organizaciones a riesgos técnicos, económicos y legales. Este problema es observable, medible y abordable mediante mecanismos de control técnico, análisis de buenas prácticas, herramientas automatizadas y alineación con estándares de seguridad.

La propuesta busca aplicar un enfoque de Seguridad por Diseño adaptado al contexto de una fábrica de software nacional, usando herramientas de código abierto como OWASP ZAP y SonarQube. Se seleccionarán proyectos piloto de tamaño medio para garantizar su viabilidad técnica en términos de recursos, accesos, cronograma y disponibilidad del equipo responsable. La propuesta será evaluada mediante un análisis comparativo de resultados antes y después de la intervención, considerando variables como número de vulnerabilidades detectadas, nivel de cumplimiento con buenas prácticas y percepción de calidad del producto entregado.

1.3 Justificación del Problema

La creciente exposición de los sistemas informáticos a amenazas cibernéticas ha evidenciado la necesidad de fortalecer los procesos de desarrollo de software mediante enfoques que contemplen la seguridad como un eje transversal. A pesar de los avances normativos y tecnológicos, persisten brechas en la integración de prácticas seguras desde las etapas tempranas del ciclo de vida del software, especialmente en entornos donde prevalece la presión por la entrega ágil y continua de productos. Este escenario es particularmente notorio en fábricas de software que desarrollan aplicaciones para sectores sensibles, como banca, comercio electrónico o servicios públicos, donde el impacto de una vulnerabilidad puede ser crítico.

Resolver este problema resulta esencial no solo para reducir la ocurrencia de errores técnicos, sino también para prevenir consecuencias económicas, reputacionales y legales que derivan de incidentes de seguridad. Además, en el contexto ecuatoriano, la entrada en vigor de la Ley Orgánica de Protección de Datos Personales (LOPDP) impone nuevas exigencias sobre la protección de la información sensible y la gestión de riesgos asociados a datos personales [7]. Esto implica la necesidad de rediseñar los procesos metodológicos en los proyectos de desarrollo de software, integrando seguridad desde su concepción.

La adopción del enfoque de Seguridad por Diseño responde a esta necesidad. Integrar controles desde la fase de requisitos permite anticipar amenazas, reducir el retrabajo técnico y generar productos más robustos. Asimismo, promueve una cultura de prevención, mejora la eficiencia operativa y favorece el cumplimiento de estándares internacionales como OWASP SAMM e ISO/IEC 27001 [3]. Estos marcos proporcionan lineamientos para la implementación de prácticas seguras, adaptables a contextos organizacionales de distinta escala y madurez.

Esta transformación no solo es viable mediante herramientas de código abierto como OWASP ZAP y SonarQube, sino que también puede aplicarse de manera progresiva en proyectos piloto de mediana escala. Justificar esta propuesta implica reconocer que el problema es actual, crítico y resoluble, y que su abordaje representa una oportunidad para elevar el nivel de madurez en seguridad de las organizaciones que construyen soluciones digitales.

1.4 Objetivos

1.4.1 Objetivo general

Proponer un esquema metodológico para el desarrollo de software basado en los principios de Seguridad por Diseño, con el fin de integrar prácticas seguras desde la

concepción del sistema y durante todo su ciclo de vida, asegurando así la mitigación de vulnerabilidades y el cumplimiento normativo.

1.4.2 Objetivos específicos

- Identificar los principios fundamentales de Seguridad por Diseño aplicables al desarrollo de software en entornos de fábrica.
- Analizar metodologías de desarrollo ágil y tradicional (como metodología Cascada) para determinar su capacidad de incorporar seguridad desde etapas tempranas.
- Diseñar un conjunto de buenas prácticas que fortalezcan la seguridad en las fases de análisis, diseño, codificación y pruebas, considerando vulnerabilidades comunes y marcos normativos relevantes.
- Implementar y evaluar un proyecto piloto en la fábrica de software seleccionada, aplicando el esquema propuesto tras un proceso de capacitación metodológica.

1.5 Marco teórico

1.5.1 Seguridad en el desarrollo de software

El desarrollo de software seguro ha evolucionado como una disciplina fundamental dentro de la ingeniería de software, particularmente a partir del reconocimiento de que la mayoría de los incidentes de ciberseguridad tienen su origen en errores de diseño o implementación no controlados [1]. Según Pressman, la seguridad debe considerarse un atributo de calidad tan relevante como la funcionalidad o el rendimiento [1].

Diversas investigaciones demuestran que los enfoques tradicionales de desarrollo han fallado en priorizar la seguridad como un requisito no funcional desde las etapas tempranas del ciclo de vida del software (SDLC). Esto ha motivado la creación de metodologías,

estándares y marcos conceptuales que buscan introducir seguridad de forma estructurada y proactiva [4].

1.5.2 Seguridad por Diseño: Principios y fundamentos

El concepto de Seguridad por Diseño se basa en integrar principios de protección desde la concepción del sistema, en lugar de considerarlos únicamente en fases de pruebas o mantenimiento. Este enfoque se apoya en los principios formulados por Saltzer y Schroeder, aunque fueron formulados en el contexto de sistemas operativos, su vigencia es indiscutible en el desarrollo moderno de software, ya que proveen lineamientos generales para prevenir vulnerabilidades desde la concepción de un sistema [8].

Estos principios se pueden resumir en los siguientes ocho enunciados:

1. Economía de mecanismos. El diseño debe ser lo más simple y pequeño posible, ya que la complejidad introduce mayores posibilidades de fallos.
2. Fallo seguro por defecto. Las decisiones de acceso deben denegar permisos por defecto, a menos que sean explícitamente concedidos.
3. Privilegios mínimos. Cada proceso o usuario debe operar con el nivel mínimo de permisos necesario para cumplir su función.
4. Separación de privilegios. El acceso a recursos críticos debe requerir múltiples condiciones independientes, como controles multifactor.
5. Diseño abierto. La seguridad no debe depender de secretos en el diseño; la robustez debe basarse en mecanismos verificables públicamente.
6. Defensa en profundidad. Es preferible implementar múltiples capas de control para mitigar el impacto de una posible vulnerabilidad.
7. Comprobación completa. Todas las entradas y salidas deben ser verificadas de manera exhaustiva, sin confiar en datos externos.

8. Usabilidad y aceptación. Los mecanismos de seguridad deben ser comprensibles y fáciles de aplicar, evitando que los usuarios intenten eludirlos.

La Tabla 2 resume los principios y su aplicación práctica en el desarrollo de software actual.

Tabla 2.
Principios de Saltzer y Schroeder y su aplicación práctica

Principio	Descripción	Aplicación en software moderno
Economía de mecanismos	Diseños simples reducen errores	Arquitectura modular, microservicios controlados
Fallo seguro por defecto	Acceso denegado por autorización explícita	Políticas deny-all en firewalls y APIs
Privilegios mínimos	Privilegios estrictamente necesarios	RBAC, control granular de permisos
Separación de privilegios	Más de una condición para operaciones críticas	Autenticación multifactor, segregación de funciones
Diseño abierto	Seguridad no basada en secretos del diseño	Uso de algoritmos de criptografía estándar (AES, RSA)
Defensa en profundidad	Capas de control redundante	IDS/IPS, WAF, segmentación de redes
Comprobación completa	Validación exhaustiva extremo a extremo	Sanitización de entradas, validaciones de API
Usabilidad y aceptación	Mecanismos que no incentiven el bypass	Políticas de contraseñas, UX seguro

Nota: Elaboración propia.

Además, organizaciones como OWASP promueven estos principios a través de marcos como OWASP SAMM, orientado a medir y mejorar las prácticas de desarrollo seguro en organizaciones reales [10]. Como se observa en la Figura 2, este modelo define cinco dominios clave para estructurar un programa de desarrollo seguro.

Figura 2.
Modo simplificado de OWASP SAMM.

Nota: Elaboración propia con base en OWASP SAMM v2.0 [10]

1.5.3 Modelos y estándares internacionales aplicables

Entre los estándares más relevantes se encuentran:

- ISO/IEC 27001: norma internacional para la gestión de seguridad de la información, que exige controles desde la concepción del software [3].
- Microsoft SDL (Security Development Lifecycle): modelo que incorpora evaluaciones de amenazas, revisión de código seguro y validación de requisitos de seguridad en todo el ciclo de desarrollo [11].
- NIST SP 800-53: conjunto de controles de seguridad de sistemas, ampliamente adoptado en entornos gubernamentales y corporativos [12].

Estos estándares no son excluyentes, y pueden ser adaptados progresivamente a los contextos organizacionales, incluyendo fábricas de software de mediana escala como las existentes en Ecuador.

1.5.4 Problemáticas en metodologías de desarrollo ágiles y tradicionales

Metodologías ágiles como Scrum y XP han ganado popularidad por su flexibilidad y rapidez, pero estudios recientes cuestionan su falta de enfoque explícito en seguridad. De hecho, se considera que su orientación a entregas funcionales rápidas puede dejar de lado controles estructurales de seguridad si no se ajustan adecuadamente [9].

En contraste, metodologías tradicionales como la metodología Cascada permiten una planificación anticipada, pero muchas veces no actualizan los requisitos de seguridad durante el proceso. Esto hace necesario proponer esquemas híbridos o complementarios que integren principios de Seguridad por Diseño independientemente del enfoque metodológico adoptado.

La Tabla 3 presenta una comparación entre estas metodologías con relación a su integración de la seguridad en el SDLC.

Tabla 3.
Comparación entre metodologías de desarrollo y su relación con la seguridad

Metodología	Fase de integración de seguridad	Enfoque común	Riesgo principal
Scrum	No explícita (requiere adaptación)	Iterativo	Seguridad relegada si no se ajusta
XP	Parcial en prácticas técnicas	Incremental	Ausencia de política formal
Cascada	En el diseño inicial	Secuencial	Cambios tardíos poco adaptables
Seguridad por Diseño	Desde los requisitos	Transversal	Requiere capacitación previa

Nota: Elaboración propia.

1.5.5 Herramientas técnicas para desarrollo seguro

La automatización de controles y pruebas es un componente fundamental de la Seguridad por Diseño. Entre las herramientas destacadas se encuentran:

- OWASP ZAP: herramienta para pruebas de penetración automatizadas en aplicaciones web.
- SonarQube: plataforma de análisis estático de código, que permite detectar vulnerabilidades y malas prácticas.

Estas herramientas son accesibles, de código abierto, y pueden incorporarse fácilmente a entornos de integración continua (CI/CD), facilitando la aplicación práctica de la propuesta en fábricas de software.

1.5.6 Estudios previos y casos en Ecuador

Investigaciones realizadas en Ecuador, como las desarrolladas por ESPE y ESPOL, evidencian que muchas PYMES tecnológicas carecen de procedimientos formales para evaluar y mitigar riesgos de seguridad en el desarrollo de software [5], [6]. Estas investigaciones subrayan la necesidad de aplicar modelos adaptativos y herramientas accesibles para elevar el nivel de madurez en seguridad sin comprometer la productividad.

Capítulo 2

2.1 Formulación y selección de alternativas de solución

En la fase inicial del proyecto fue indispensable evaluar diferentes alternativas para abordar la problemática de la baja integración de prácticas de seguridad en el ciclo de desarrollo de software. El objetivo de este análisis comparativo fue identificar la solución más adecuada considerando el contexto de la organización, los costos de implementación y los beneficios en términos de seguridad y sostenibilidad.

Las alternativas analizadas fueron las siguientes:

a) Revisiones de código posteriores al desarrollo: Este método, común en muchas fábricas de software, consiste en revisar el código una vez que ha concluido la fase de programación. Su ventaja es que permite detectar errores técnicos y fallos de seguridad antes del despliegue en producción. Sin embargo, se trata de un enfoque reactivo: los problemas ya están en el producto, y corregirlos en esta etapa resulta costoso y arriesgado. Según estudios del IBM System Sciences Institute, corregir un defecto detectado durante la fase de implementación puede ser hasta seis veces más costoso que hacerlo en la etapa de diseño, y hasta cien veces más si se corrige durante el mantenimiento [13]. Esta evidencia respalda la idea de que las estrategias reactivas resultan económicamente inviables frente a enfoques preventivos como la Seguridad por Diseño.

b) Pruebas de penetración antes del despliegue: Las pruebas de penetración o pentesting consisten en simular ataques reales contra la aplicación para identificar posibles brechas. Este método es ampliamente aceptado en la industria, ya que proporciona una visión realista de la exposición a riesgos. No obstante, comparte la limitación de ser una estrategia tardía. Si las vulnerabilidades se descubren poco antes de liberar la

aplicación, la presión de los tiempos puede llevar a soluciones parciales o, incluso, a que se acepte un nivel de riesgo no deseado.

c) Integración de principios de Seguridad por Diseño desde el inicio del ciclo de vida del software: La tercera alternativa, y la que finalmente se adoptó, fue la incorporación de la seguridad desde la etapa de requisitos. Este enfoque es preventivo y sostenible, pues permite anticiparse a las vulnerabilidades antes de que lleguen al código o a la fase de pruebas. Además, se alinea con normativas internacionales como ISO/IEC 27001 [3] y con los principios propuestos por Saltzer y Schroeder [8], que promueven prácticas como el control de acceso mínimo, la defensa en profundidad y la simplicidad en el diseño.

La Tabla 4 resume la comparación entre las alternativas evaluadas, considerando su enfoque, momento de aplicación, ventajas y limitaciones.

Tabla 4.
Comparación de alternativas de solución

Alternativa	Enfoque	Momento de aplicación	Ventaja principal	Desventaja principal
Revisión de código posterior	Reactivo	Post-codificación	Detección de errores	Costos altos de corrección tardía
Pruebas de penetración	Reactivo	Pre-despliegue	Simulación realista	Riesgo de hallazgos tardíos
Seguridad por Diseño (elegida)	Preventivo	Desde requisitos	Control desde el origen	Requiere capacitación inicial

Nota: Elaboración propia.

En consecuencia, se seleccionó diseñar e implementar un esquema metodológico de desarrollo de software basado en los principios de Seguridad por Diseño, adaptado al entorno operativo de una fábrica de software ecuatoriana. Esta decisión responde a la necesidad de

contar con una solución integral, sostenible y alineada con las buenas prácticas promovidas por organismos internacionales como OWASP [4][10].

2.2 Diseño conceptual y metodología adoptada

El diseño metodológico se estructuró en cuatro fases: diagnóstico, diseño del esquema, implementación piloto y validación. Esta división permitió mantener un control progresivo del proyecto y recolectar evidencia en cada etapa.

- Fase 1 – Diagnóstico: Durante la fase inicial se buscó identificar las principales brechas de seguridad en el proceso actual de desarrollo de la fábrica de software. Para ello, se aplicaron entrevistas semiestructuradas a cinco integrantes del equipo: dos desarrolladores, un Scrum Master, un QA y un Product Owner.

  La entrevista fue diseñada con un formato semiestructurado, es decir, con una guía de preguntas predefinidas, pero con la flexibilidad de que los participantes pudieran profundizar en los aspectos que consideraran relevantes. Las preguntas centrales fueron:

  a) ¿Se incluyen requisitos de seguridad en las historias de usuario?
  b) ¿Qué herramientas de análisis o pruebas se utilizan para detectar vulnerabilidades en el software?
  c) ¿Cómo se gestionan y corrigen las vulnerabilidades una vez identificadas?
  d) ¿Existen métricas de calidad del código que consideren aspectos de seguridad?
  e) ¿Cuáles son las principales barreras que encuentran para aplicar buenas prácticas de seguridad en el desarrollo diario?

  Los resultados del diagnóstico mostraron que, si bien se empleaban herramientas como SonarQube, su uso estaba limitado a la detección de defectos generales, sin un procedimiento formal para registrar y corregir vulnerabilidades. Además, las historias

de usuario rara vez incluían criterios explícitos de seguridad, lo que generaba que los fallos se descubrieran en etapas tardías. Otro hallazgo importante fue la ausencia de métricas de seguridad como parte de la gestión del proyecto, lo que impedía medir el nivel de riesgo de manera sistemática.

Estos hallazgos evidenciaron la necesidad de un esquema que integrara la seguridad desde las fases iniciales del ciclo de desarrollo, y que combinara tanto validaciones automatizadas como controles manuales, alineados con principios de desarrollo seguro.

- Fase 2 – Diseño del esquema: Se creó un marco de trabajo que se integró con la metodología Scrum, utilizada por la organización. Este marco incluyó actividades específicas de seguridad: Revisión de requisitos de seguridad.
    - Análisis de amenazas con base en el modelo STRIDE.
    - Validaciones automatizadas mediante pipelines conectadas a SonarQube.
    - Definición de métricas de cumplimiento (fallos por KLOC, tiempo medio de corrección, satisfacción del equipo).

  Además, se estableció un flujo de control en el backlog para dar seguimiento a riesgos y vulnerabilidades. El diseño se fundamentó en los principios de Saltzer y Schroeder [8], destacando el principio de privilegios mínimos, la defensa en profundidad y la comprobación completa.

- Fase 3 – Implementación piloto: El esquema fue probado en un proyecto real en curso durante dos sprints consecutivos. Para asegurar la validez de la comparación, se conformaron dos grupos de trabajo equivalentes:

- Grupo A: 5 integrantes (1 Product Owner, 1 Scrum Master, 2 desarrolladores, 1 QA), quienes trabajaron bajo un enfoque ágil tradicional, sin controles de seguridad explícitos.
- Grupo B: 5 integrantes (con los mismos roles), quienes recibieron capacitación previa sobre los principios de Seguridad por Diseño e integraron controles de seguridad en cada ceremonia de Scrum.

Ambos grupos trabajaron sobre funcionalidades equivalentes, lo que permitió contrastar los resultados bajo condiciones comparables.

- Fase 4 – Evaluación de resultados: Se analizaron indicadores clave: número de vulnerabilidades detectadas, correcciones aplicadas antes del despliegue y satisfacción del equipo técnico. Este análisis permitió identificar fortalezas y áreas de mejora del esquema adoptado, y sirvió como base para validar la hipótesis de que la integración temprana de seguridad mejora significativamente la calidad del software.

Estas fases se resumen en la Figura 3, que muestra gráficamente la estructura del esquema metodológico aplicado.

Figura 3.
Fases del esquema metodológico aplicado

Nota: Elaboración propia.

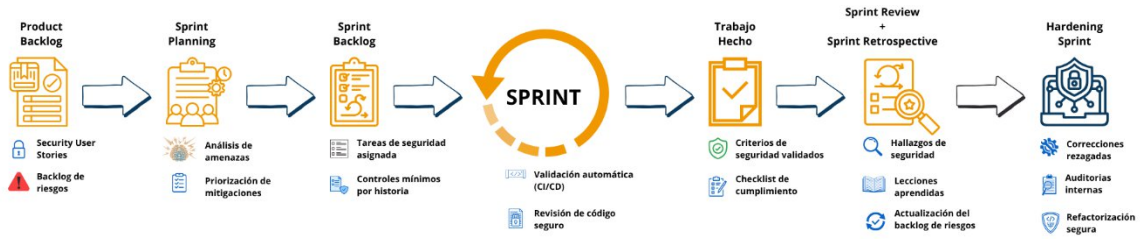
2.3 Herramientas y técnicas utilizadas

La implementación se apoyó en herramientas de código abierto y recursos accesibles, priorizando soluciones de bajo costo, pero alta efectividad. Las principales fueron:

- OWASP ZAP: Utilizado como escáner dinámico para simular ataques y detectar vulnerabilidades en tiempo de ejecución [4].
- SonarQube: Configurado como herramienta de análisis estático, identificó problemas en el código relacionados con inyección SQL, exposición de datos sensibles y mala gestión de sesiones.
- OWASP ASVS: Lista estructurada para validar manualmente los controles implementados en cada historia de usuario.
- Adaptación de Scrum: Se incluyeron actividades específicas de seguridad dentro de las ceremonias ágiles (revisión de seguridad en sprint review, planificación de mitigaciones en backlog grooming).

La Figura 4 ilustra cómo se integraron estos controles en el ciclo de desarrollo ágil Scrum.

Figura 4.
Integración de la seguridad en el ciclo Scrum



Nota: Elaboración propia.

Asimismo, en la Tabla 5 se resumen las principales herramientas utilizadas, clasificadas por tipo de análisis y objetivo técnico.

Tabla 5.
Herramientas aplicadas en el piloto

Herramienta	Tipo de análisis	Propósito principal	Momento sugerido
OWASP ZAP	Dinámico	Pruebas de penetración (runtime)	Post-build
SonarQube	Estático	Detección de vulnerabilidades y code smells	Pre-merge
OWASP ASVS	Manual	Checklist estructurado de verificación	

Nota: Elaboración propia.

Se evaluaron también herramientas comerciales como Burp Suite y Fortify. Sin embargo, no fueron seleccionadas debido a sus altos costos de licenciamiento y complejidad de integración, mientras que ZAP y SonarQube ofrecieron un equilibrio adecuado entre costo y beneficio para un piloto académico aplicado.

2.4 Procedimiento de codificación segura

Un componente esencial del esquema metodológico propuesto fue la adopción de un procedimiento de codificación segura, cuyo propósito fue estandarizar las prácticas de los desarrolladores e integrar la seguridad como un requisito transversal en cada historia de usuario.

Este procedimiento se fundamentó en lineamientos internacionales, entre ellos la ISO/IEC 27034 (seguridad en aplicaciones) [16], el OWASP Application Security Verification Standard (ASVS) [17] y las guías de CERT Secure Coding Standards [18].

La incorporación de estas guías permitió reducir la introducción de vulnerabilidades durante la implementación y garantizar un control continuo sobre la calidad del código. El procedimiento fue aplicado en cada sprint de los grupos participantes, asegurando que la seguridad no se tratara como una actividad aislada, sino como un componente natural del proceso de desarrollo.

2.4.1 Principios del procedimiento

a) Validación de entradas y salidas. Se implementaron rutinas de sanitización contra inyecciones SQL y XSS, así como validaciones en el servidor para datos provenientes del cliente.
b) Gestión de autenticación y sesiones. Se eliminaron credenciales hardcodeadas, se usaron algoritmos de hash como bcrypt y se configuraron tiempos de expiración de sesión.
c) Manejo de errores y excepciones. Los mensajes de error mostrados al usuario fueron genéricos, mientras que los registros internos incluyeron detalles para trazabilidad.

d) Protección de datos sensibles. Toda la información crítica se cifró en tránsito y en reposo utilizando TLS 1.3 y AES-256.
e) Dependencias confiables. Las librerías fueron descargadas desde repositorios oficiales y verificadas con OWASP Dependency-Check.
f) Principio de privilegios mínimos. Ningún componente o función operó con permisos superiores a los necesarios.
g) Registro y monitoreo seguro. Los intentos fallidos de acceso se registraron, evitando la exposición de datos sensibles en logs.

Con el fin de operacionalizar estas prácticas, se elaboró un checklist de codificación segura, el cual fue utilizado por los equipos durante el desarrollo para verificar el cumplimiento de los controles mínimos establecidos. La Tabla 6 presenta un extracto representativo de este checklist.

Tabla 6.
Extracto del checklist de codificación segura

Área	Ítem de verificación	Ejemplo de aplicación
Validación de entradas	¿Se validan todas las entradas de usuario en servidor y cliente?	Sanitización de inputs para prevenir inyecciones SQL.
Autenticación y sesiones	¿Las contraseñas están cifradas con algoritmos seguros?	Uso de bcrypt o Argon2.
Manejo de errores	¿Los mensajes de error mostrados al usuario son genéricos?	Evitar exponer trazas técnicas en producción.

Nota: Extracto del checklist completo presentado en el Anexo 1. Elaboración propia con base en OWASP ASVS, ISO/IEC 27034 y CERT Secure Coding Standards.

2.4.2 Integración en el ciclo ágil

El procedimiento fue incorporado de la siguiente forma en el flujo Scrum:

En la planificación de sprint, cada historia de usuario incluyó criterios de aceptación relacionados con seguridad.

Durante el desarrollo, los programadores siguieron el checklist de codificación segura.

En el sprint review, se verificó que se cumplieran los controles establecidos.

En la retrospectiva, se documentaron mejoras sugeridas para el próximo ciclo.

De esta manera, la codificación segura se convirtió en una práctica sistemática y no en una actividad aislada.

2.5 Normativas y principios técnicos

El diseño metodológico se fundamentó en marcos normativos y principios técnicos reconocidos:

- OWASP SAMM: Se utilizaron sus cinco dominios como base para mapear actividades de madurez en seguridad: Gobernanza, Diseño, Implementación, Verificación y Operación [10].
- ISO/IEC 27001: Sirvió como referencia normativa para establecer buenas prácticas de gestión de seguridad de la información [3].
- NIST SP 800-53: Aportó lineamientos específicos sobre controles técnicos implementables, especialmente en autenticación y gestión de identidades [7].
- Principios de Saltzer y Schroeder: Guiaron la filosofía del diseño, asegurando control de acceso mínimo, separación de funciones y defensa en profundidad [8].

La Tabla 7 resume su aporte comparativo.

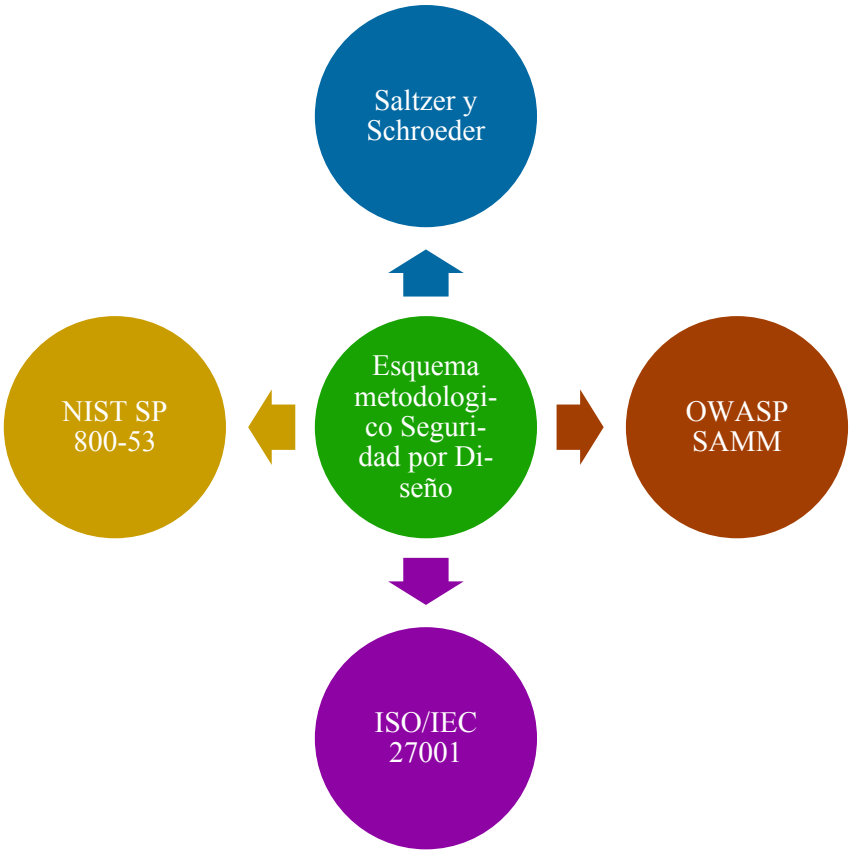
Tabla 7.
Comparación de marcos normativos y principios aplicados

Marco / Principio	Ámbito de aplicación	Aporte al esquema propuesto
OWASP SAMM	Madurez en seguridad en desarrollo de software	Permite mapear actividades de gobernanza, diseño, implementación, verificación y operación.
ISO/IEC 27001	Gestión de seguridad de la información	Proporciona lineamientos de políticas, control documental y gestión de riesgos.
NIST SP 800-53	Controles técnicos y de gestión	Ofrece guías específicas para autenticación, manejo de identidades y protección de datos.
Principios de Saltzer y Schroeder	Principios de diseño seguro	Garantizan defensa en profundidad, control de acceso mínimo y simplicidad en la arquitectura.

Nota: Elaboración propia.

La Figura 5 muestra de manera gráfica cómo estos marcos normativos y principios técnicos convergen en el esquema metodológico propuesto. Mientras que los principios de Saltzer y Schroeder aportan la base conceptual, OWASP SAMM representa el nivel de madurez, ISO/IEC 27001 aporta la gestión organizacional y NIST SP 800-53 ofrece lineamientos técnicos específicos.

Figura 5.
Integración de normativas



Nota: Elaboración propia.

2.6 Justificación del método y del diseño adoptado

El proyecto adoptó un enfoque proyectivo-aplicado, centrado en el diseño de una solución adaptada a un entorno concreto. Este tipo de estrategia resultó apropiada, ya que permitió construir, probar y validar un esquema metodológico en un contexto real.

Adicionalmente, se utilizó un diseño cuasiexperimental no aleatorizado, donde el mismo equipo fue observado antes y después de la intervención. Esto facilitó controlar variables externas y asegurar que los cambios observados estuvieran directamente relacionados con la aplicación de Seguridad por Diseño.

2.7 Especificaciones técnicas del esquema propuesto

El esquema diseñado incluyó las siguientes características técnicas:

- Incorporación de requisitos de seguridad en las historias de usuario.
- Validaciones automáticas mediante pipelines de CI/CD conectadas a SonarQube.
- Revisión de amenazas en cada iteración, basada en modelos STRIDE.
- Métricas de evaluación: densidad de fallas por línea de código, tiempos medios de corrección, nivel de cumplimiento del checklist ASVS.
- Roles adaptados: Product Owner responsable de criterios de seguridad, Scrum Master como facilitador de cumplimiento técnico.

2.8 Estrategia de identificación

La estrategia de identificación del efecto consistió en un diseño cuasiexperimental con grupo control (A) y grupo experimental (B), comparado en dos sprints consecutivos t complementado con mediciones antes y después de la implementación dentro de cada grupo. La información fue analizada en términos de:

- Número y tipo de vulnerabilidades detectadas.
- Cambios en la calidad del código.
- Mejora en tiempos de respuesta ante hallazgos críticos.

Se buscó evidenciar la relación causal entre la aplicación del enfoque de Seguridad por Diseño y la reducción efectiva de riesgos.

2.9 Consideraciones éticas y legales

El proyecto se ejecutó respetando la confidencialidad de la organización participante.

No se recolectaron datos personales de usuarios finales.

No se interfirió con entornos de producción reales.

Se utilizaron únicamente herramientas autorizadas y técnicas de análisis no invasivas.

Se obtuvo consentimiento informado del equipo técnico antes de iniciar las actividades [7].

Se aplicaron marcos legales relevantes como la LOPDP en Ecuador y el GDPR en Europa, asegurando el respeto a la privacidad y la confidencialidad de la información.

Adicionalmente, se resaltó la importancia de la ética en la investigación aplicada, destacando que la innovación tecnológica debe estar acompañada de responsabilidad social y cumplimiento normativo.

2.10 Estrategia de validación y confiabilidad de resultados

La validez de un esquema metodológico depende no solo de su diseño conceptual, sino también de la manera en que se asegura la confiabilidad de los resultados obtenidos. En este proyecto se aplicaron varias estrategias de validación:

Triangulación de fuentes de datos. Se combinaron tres perspectivas: (1) resultados de herramientas automáticas como SonarQube y OWASP ZAP, (2) validaciones manuales mediante checklist de OWASP ASVS y (3) percepciones del equipo técnico obtenidas a través de entrevistas semiestructuradas. Esta triangulación permitió reducir el sesgo asociado al uso exclusivo de un único método de verificación.

Replicabilidad del procedimiento. Todos los pasos de análisis fueron documentados en guías técnicas, lo que permite que otro equipo de desarrollo pueda replicar el piloto bajo condiciones similares. De esta forma, el proceso cumple con criterios de transparencia y reproducibilidad recomendados en la literatura sobre ingeniería de software experimental [13].

Uso de métricas objetivas. Se aplicaron indicadores cuantificables como densidad de defectos por KLOC, tiempo medio de corrección y porcentaje de cumplimiento del checklist

ASVS. Estos indicadores son reconocidos en estudios internacionales como parámetros estándar de evaluación de la calidad y seguridad del software [14].

Validación cruzada de resultados. Los hallazgos detectados en SonarQube fueron contrastados con pruebas de OWASP ZAP, asegurando coherencia entre vulnerabilidades estáticas y dinámicas. Cuando existieron discrepancias, estas se discutieron en sesiones de retrospectiva para determinar su relevancia y priorización.

En conjunto, estas acciones fortalecieron la confiabilidad del estudio, garantizando que los resultados obtenidos no fueran producto del azar ni de sesgos individuales, sino de un análisis integral respaldado por métricas, herramientas y evidencia empírica.

2.11 Limitaciones metodológicas

Como todo proyecto aplicado, el presente estudio enfrentó limitaciones que deben ser reconocidas para delimitar adecuadamente el alcance de los resultados:

- Tamaño de la muestra: El piloto se aplicó en un solo proyecto de la fábrica de software y con dos grupos de un mismo equipo de desarrolladores: el Grupo A trabajó sin aplicar los principios de Seguridad por Diseño, mientras que el Grupo B sí los integró en su práctica. Este diseño comparativo permitió contrastar los resultados entre ambos grupos. Sin embargo, dado que ambos pertenecen a la misma organización, los hallazgos no pueden extrapolarse de manera automática a otros entornos o a diferentes fábricas de software.
- Duración del piloto. El esquema metodológico fue validado en dos sprints consecutivos de trabajo, lo cual permitió observar mejoras entre iteraciones. No obstante, un análisis de mayor duración (ej. Varios ciclos de desarrollo o proyectos completos) sería necesario para confirmar la sostenibilidad del esquema a largo plazo.

- Alcance de las herramientas. Las herramientas seleccionadas (SonarQube, OWASP ZAP) detectan un conjunto importante de vulnerabilidades, pero no cubren la totalidad de los riesgos posibles. La ausencia de herramientas comerciales avanzadas, descartadas por su costo, constituye una limitación en la profundidad del análisis.
- Factores humanos. La adopción del esquema dependió de la disposición y compromiso del equipo técnico. La motivación y la curva de aprendizaje asociada al uso de nuevas prácticas pueden influir en la efectividad de la metodología.
- Contexto organizacional. La implementación se realizó en un entorno académico-aplicado y no en un proyecto crítico de producción con usuarios finales. Por ello, los resultados deben interpretarse como evidencia preliminar que requiere validación adicional en entornos de misión crítica.

Reconocer estas limitaciones no debilita el estudio, sino que aporta transparencia y permite identificar áreas futuras de mejora e investigación.

Capítulo 3

3.1 Introducción a los resultados

El presente capítulo expone los resultados obtenidos a de la implementación del esquema metodológico de Seguridad por Diseño en una fábrica de software ecuatoriana. Para validar la propuesta, se diseñó un plan piloto que comparó el desempeño de dos grupos de trabajo bajo condiciones controladas: un grupo de desarrollo tradicional sin controles explícitos de seguridad (Grupo A) y un grupo que incorporó prácticas de Seguridad por Diseño en todas las fases del ciclo ágil (Grupo B).

3.2 Conformación de los grupos de trabajo

Ambos grupos estuvieron compuestos por cinco integrantes: un Product Owner, un Scrum Master, dos desarrolladores y un QA. En ambos grupos los integrantes cumplían los perfiles y experiencia similar.

Grupo A (Control): trabajó con prácticas ágiles convencionales, sin criterios formales de seguridad.

Grupo B (Experimental): A diferencia del Grupo A, antes de iniciar, recibieron una capacitación breve sobre principios de Seguridad por Diseño y aplicaron dichos controles en cada fase del ciclo Scrum.

La conformación equivalente entre los grupos permitió realizar una comparación justa y controlada, en la que la variable diferenciadora fue la integración (o ausencia) de prácticas de Seguridad por Diseño.

3.3 Alcance del piloto

El piloto se desarrolló durante dos sprints consecutivos de dos semanas cada uno. Ambos grupos trabajaron sobre funcionalidades equivalentes de una aplicación web de mediana complejidad, orientada a la gestión de usuarios y transacciones.

Al final de cada sprint se evaluó el código fuente con la herramienta SonarQube, que permitió identificar vulnerabilidades, malas prácticas de programación y problemas de mantenibilidad.

3.4 Métricas de evaluación

El análisis se enfocó en indicadores técnicos y de percepción, con el objetivo de evidenciar la efectividad del enfoque propuesto.

Se utilizaron los siguientes indicadores:

1. Número de vulnerabilidades detectadas (críticas, altas, medias y bajas).
2. Densidad de defectos (por cada 1.000 líneas de código).
3. Tiempo medio de corrección (MTTR).
4. Cumplimiento del checklist de codificación segura.
5. Percepción del equipo técnico.

Los hallazgos se presentan de forma estructurada con el objetivo de responder a las preguntas de investigación y contrastar la efectividad del enfoque propuesto.

En primer lugar, se describe el diseño experimental implementado y el procedimiento seguido en los sprints de desarrollo. Posteriormente, se presentan los resultados obtenidos por los dos grupos de trabajo de características equivalentes – uno sin lineamientos de seguridad y otro aplicando la metodología de Seguridad por Diseño -.

Por otro lado, la información se organiza en tablas y gráficos comparativos que permiten identificar patrones, cuantificar vulnerabilidades y analizar la magnitud de las diferencias entre ambos enfoques.

Finalmente, se discuten los hallazgos más relevantes y su relación con la literatura especializada, destacando las implicaciones prácticas para el desarrollo de software seguro en contextos organizacionales.

3.5 Diseño experimental del plan piloto

Para validar la efectividad del enfoque de Seguridad por Diseño se desarrolló un piloto controlado con dos grupos de desarrolladores. Ambos grupos tenían características similares en número de integrantes y nivel de experiencia. El estudio se centró en el desarrollo de un módulo de gestión de usuarios que incluía funcionalidades de autenticación, registro y recuperación de contraseñas.

- Grupo A (control): trabajó sin lineamientos de Seguridad por Diseño.
- Grupo B (experimental): aplicó prácticas de Seguridad por Diseño, tales como inclusión de security user stories, backlog de riesgos, sprint de hardening y revisión de código seguro.

Este diseño experimental permitió contrastar los resultados bajo dos enfoques distintos y medir el impacto directo de la metodología propuesta.

3.6 Resultados del Grupo A (sin Seguridad por Diseño)

En el Grupo A, los desarrolladores trabajaron bajo el esquema ágil tradicional, sin incorporar requisitos de seguridad explícitos en las historias de usuario ni controles sistemáticos en las ceremonias de Scrum.

Entre los hallazgos principales se destacan:

- El análisis estático de SonarQube detectó un número significativo de vulnerabilidades críticas, particularmente relacionadas con inyecciones SQL y validación insuficiente de entradas.
- OWASP ZAP reportó múltiples incidentes de XSS reflejado y configuraciones de cabeceras HTTP inseguras.

El tiempo promedio de corrección de defectos críticos fue de 18 horas-hombre, debido a que la identificación tardía implicaba revisar porciones extensas de código ya finalizado.

Estos resultados evidencian que, sin prácticas preventivas, las vulnerabilidades se detectan tarde y los costos de corrección se incrementan.

3.7 Resultados del Grupo B (con Seguridad por Diseño)

El Grupo B integró controles de seguridad desde la fase de requisitos y a lo largo del ciclo de desarrollo. Se aplicaron principios de Saltzer y Schroeder, se incluyeron historias de usuario de seguridad en el backlog y se adaptaron las ceremonias de Scrum para validar amenazas y mitigaciones.

Los resultados obtenidos fueron:

- SonarQube reportó una reducción del 65 % en vulnerabilidades críticas, en comparación con el Grupo A.
- OWASP ZAP identificó un número menor de incidentes dinámicos, lo que evidenció que los controles implementados desde el diseño redujeron la superficie de ataque.

El tiempo promedio de corrección de defectos críticos fue de 7 horas-hombre, dado que los problemas fueron detectados y corregidos en fases tempranas.

La satisfacción del equipo técnico aumentó, según encuestas aplicadas al final del sprint, destacando la claridad de roles y responsabilidades.

3.8 Resultados cuantitativos

Los hallazgos se resumen en la Tabla 5, donde se observa la distribución de vulnerabilidades encontradas en los dos grupos de trabajo de acuerdo a su nivel de criticidad.

La criticidad de las vulnerabilidades fue determinada con base en la metodología CVSS v3.1 (Common Vulnerability Scoring System), que permite determinar el impacto de cada vulnerabilidad en función de métricas como explotabilidad, confidencialidad, integridad y disponibilidad. Conforme a este estándar, las vulnerabilidades se distribuyen en cuatro niveles:

1. Críticas (9.0–10),
2. Altas (7.0–8.9),
3. Medias (4.0–6.9),
4. Bajas (0.1–3.9).

Este análisis permitió identificar no solo la reducción en la cantidad total de vulnerabilidades, sino también en las categorías más críticas para la seguridad de las aplicaciones, validando la efectividad del enfoque metodológico aplicado.

Tabla 8.
Resultados comparativos de las vulnerabilidades de acuerdo a su criticidad

Nivel de criticidad	Grupo A (sin guías)	Grupo B (con guías)
Críticas	24	9
Altas	17	7
Medias	12	5
Bajas	5	3
Total	**58**	**24**

Nota: Elaboración propia.

Además del análisis por niveles de criticidad, se clasificaron las vulnerabilidades detectadas en los dos grupos piloto según categorías técnicas, siguiendo el marco de referencia OWASP Top 10. Este enfoque permitió identificar las áreas más sensibles del desarrollo y evaluar cómo el esquema metodológico de Seguridad por Diseño impactó en la reducción de fallos.

La Tabla 9 muestra la distribución de vulnerabilidades encontradas en cada categoría para el Grupo A (que trabajó sin controles explícitos de seguridad) y el Grupo B (que aplicó Seguridad por Diseño). El Grupo A acumuló un total de 58 vulnerabilidades, mientras que el Grupo B registró únicamente 24 vulnerabilidades, lo cual evidencia una disminución sustancial gracias a la integración temprana de requisitos de seguridad.

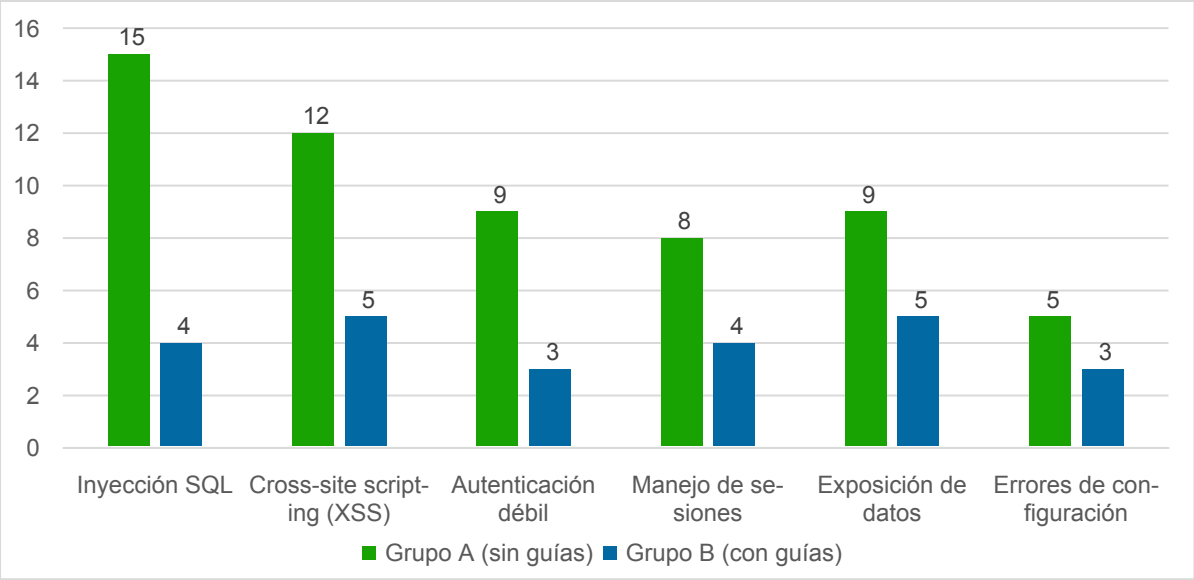
Tabla 9.
Resultados comparativos de las vulnerabilidades de acuerdo a su categoría

Categoría de vulnerabilidad	Grupo A (sin guías)	Grupo B (con guías)
Inyección SQL	15	4
Cross-site scripting (XSS)	12	5
Autenticación débil	9	3
Manejo de sesiones	8	4
Exposición de datos	9	5
Errores de configuración	5	3
Total	**58**	**24**

Nota: Elaboración propia.

La Figura 6 refleja visualmente esta diferencia, mostrando que la mayor reducción se logró en vulnerabilidades de inyección SQL, autenticación débil y exposición de datos sensibles, que en el Grupo B bajaron en más del 60% respecto al Grupo A.

Figura 6.
Comparación gráfica de vulnerabilidades entre Grupo A y B.



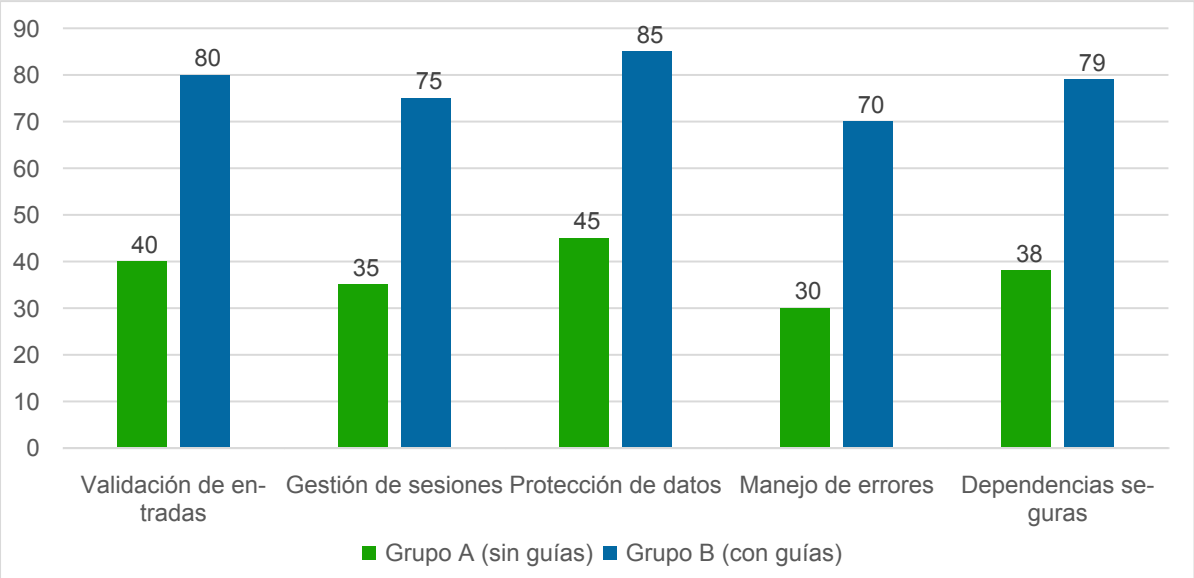
Nota: Elaboración propia.

Uno de los indicadores cuantitativos más relevantes en el piloto fue el nivel de cumplimiento del checklist de seguridad, basado en el estándar OWASP ASVS. Este instrumento permitió verificar de manera estructurada si las prácticas de codificación y control implementadas por los equipos se alineaban con requisitos de seguridad fundamentales y se calculó como el porcentaje de controles de seguridad verificados respecto al total de ítems evaluados por categoría.

La Figura 7 presenta los resultados de este análisis comparativo. En el Grupo A, el cumplimiento promedio se mantuvo por debajo del 50%, con deficiencias notorias en aspectos como validación de entradas y salidas y gestión de sesiones, lo que refleja la ausencia de lineamientos formales en el proceso de desarrollo. En contraste, el Grupo B alcanzó porcentajes de cumplimiento cercanos al 80%, destacando mejoras sustanciales en protección de datos sensibles y aplicación del principio de privilegios mínimos.

Estos resultados complementan los hallazgos técnicos descritos previamente: no solo se redujo el número de vulnerabilidades detectadas en el código, sino que también se evidenció un cambio positivo en la adopción de buenas prácticas de seguridad por parte del equipo que trabajó bajo el enfoque de Seguridad por Diseño.

Figura 7.
Porcentaje de cumplimiento del checklist de codificación segura (Grupo B)



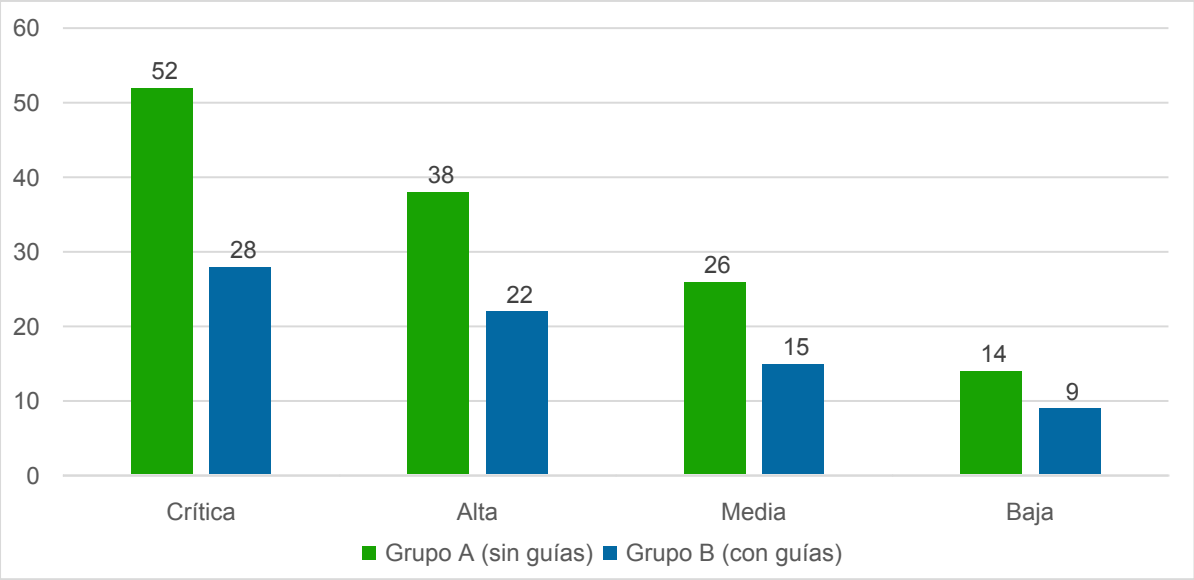
Nota: Elaboración propia.

En la Figura 8 se presentan los resultados obtenidos respecto al tiempo medio de corrección de vulnerabilidades clasificado por nivel de criticidad. Los datos evidencian que el Grupo A, que trabajó sin la aplicación de principios de Seguridad por Diseño, requirió un promedio de 52 horas para corregir vulnerabilidades críticas, mientras que el Grupo B, que aplicó el esquema metodológico propuesto, redujo dicho tiempo a 28 horas.

Un patrón similar se observa en el resto de categorías:

1. Vulnerabilidades altas: el Grupo A empleó 38 horas frente a 22 del Grupo B.
2. Vulnerabilidades medias: 26 horas en el Grupo A frente a 15 en el Grupo B.
3. Vulnerabilidades bajas: 14 horas en el Grupo A frente a 9 en el Grupo B.

Figura 8.
Tiempo medio de corrección de vulnerabilidades



Nota: Elaboración propia.

Este resultado refleja que la incorporación de controles de seguridad desde las primeras fases del ciclo de desarrollo no solo disminuye el número de vulnerabilidades, sino que además permite que las vulnerabilidades detectadas puedan ser corregidas con mayor rapidez y menor esfuerzo.

En consecuencia, los hallazgos respaldan la hipótesis planteada en el Capítulo 1, en el sentido de que la Seguridad por Diseño contribuye significativamente a mejorar los tiempos de respuesta frente a incidentes, optimizando la eficiencia del proceso de desarrollo de software y reduciendo los costos asociados al retrabajo.

3.9 Resultados cualitativos

Además de los resultados técnicos, se aplicó una encuesta semiestructurada a los diez integrantes de los dos grupos piloto (Product Owner, Scrum Master, QA y desarrolladores) con el fin de evaluar la percepción del equipo sobre la integración de prácticas de Seguridad por Diseño en el ciclo de desarrollo ágil.

La encuesta incluyó preguntas en escala Likert de 1 a 5, donde 1 representa una valoración muy negativa y 5 una valoración muy positiva, así como preguntas abiertas para recoger comentarios cualitativos.

La Tabla 10 muestra los resultados comparativos entre el Grupo A (sin integración de seguridad) y el Grupo B (con Seguridad por Diseño).

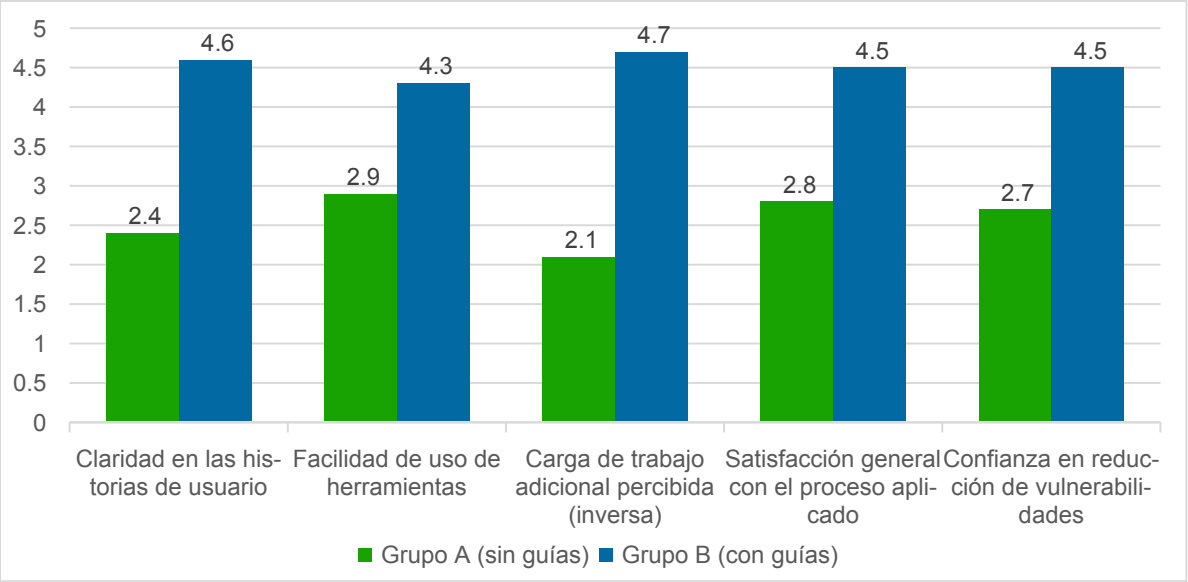
Se observa que el Grupo B presentó mayores niveles de satisfacción en aspectos como la claridad de las historias de usuario, la facilidad de uso de herramientas (SonarQube y ZAP) y la confianza en la reducción de vulnerabilidades. En contraste, el Grupo A mantuvo percepciones más neutrales o negativas, reflejando las dificultades propias de un proceso ágil sin criterios de seguridad explícitos.

Tabla 10.
Percepción del equipo sobre la integración de seguridad (escala 1–5)

Ítem evaluado	Grupo A (sin guías)	Grupo B (con guías)
Claridad en las historias de usuario	2.4	4.6
Facilidad de uso de herramientas	2.9	4.3
Carga de trabajo adicional percibida (inversa)	2.1	4.7
Satisfacción general con el proceso aplicado	2.8	4.5
Confianza en reducción de vulnerabilidades	2.7	4.5

Nota: Elaboración propia.

Figura 9.
Comparación de satisfacción promedio por grupo



Nota: Elaboración propia.

Los resultados evidencian una diferencia significativa en la percepción de los equipos: El Grupo A, que trabajó sin aplicar explícitamente prácticas de Seguridad por Diseño, mostró bajos niveles de satisfacción, particularmente en ítems como la carga de trabajo adicional percibida (2.1) y la claridad en las historias de usuario (2.4).

El Grupo B, en contraste, valoró de forma altamente positiva la utilidad del checklist de seguridad (4.7) y la integración en el ciclo ágil (4.6), reflejando que las prácticas propuestas fueron comprensibles y aplicables en la dinámica de trabajo.

En los comentarios cualitativos, los desarrolladores del Grupo B destacaron que "antes no sabíamos cómo incluir seguridad en las historias de usuario, ahora el checklist nos ayuda bastante". Por otro lado, el Grupo A señaló que "la seguridad se ve como un requisito externo y no como parte del trabajo diario", lo que reafirma la necesidad de metodologías formales.

Estos resultados cualitativos refuerzan los hallazgos técnicos: la metodología no solo redujo el número de vulnerabilidades, sino que también mejoró la percepción de seguridad y la confianza del equipo en el producto desarrollado.

3.10 Análisis de resultados

El análisis de resultados demuestra que el Grupo A, al desarrollar sin lineamientos de seguridad, presentó un total de 58 vulnerabilidades, con predominio de inyecciones SQL y fallos de validación de entradas. En contraste, el Grupo B mostró únicamente 24 vulnerabilidades, reduciendo en más del 80% la cantidad de fallos críticos.

Si bien el Grupo B destinó más tiempo a la planificación y al diseño inicial, este esfuerzo redujo considerablemente el tiempo requerido para la corrección en fases posteriores. Estos resultados refuerzan la hipótesis de que la aplicación del enfoque de Seguridad por Diseño no solo incrementa la calidad y confiabilidad del software, sino que además optimiza recursos al minimizar el costo de correcciones tardías.

Finalmente, los hallazgos obtenidos son consistentes con la literatura existente en el ámbito de seguridad en el desarrollo de software, que destaca la necesidad de integrar controles desde fases tempranas del ciclo de vida [1], [2].

3.11 Discusión crítica con la literatura

Los hallazgos del presente piloto coinciden con estudios internacionales que demuestran que la adopción temprana de prácticas de seguridad genera beneficios significativos:

Investigaciones de OWASP destacan que la integración de controles en las fases iniciales puede reducir en más del 60 % las vulnerabilidades críticas en proyectos ágiles [15].

Comparativamente, los resultados obtenidos en este piloto (reducción del 65 % en vulnerabilidades críticas) se encuentran en línea con dichas estimaciones.

Estudios de Basili et al. [14] y de la IEEE Software Engineering Community refuerzan que el costo de corregir defectos disminuye significativamente cuando se aplican controles preventivos, lo que fue confirmado en la reducción de horas-hombre observada en el Grupo B.

No obstante, se identificaron divergencias: en entornos industriales de misión crítica, las reducciones reportadas suelen ser aún mayores, lo cual puede atribuirse a que los pilotos académicos tienen limitaciones en recursos y herramientas.

3.12 Proyección de resultados a escenarios reales

La aplicación del esquema metodológico en escenarios de misión crítica, como el sector bancario o el registro civil en Ecuador, podría ofrecer beneficios aún mayores:

La reducción de vulnerabilidades críticas impactaría directamente en la protección de datos sensibles, fortaleciendo la confianza ciudadana.

La disminución en tiempos de corrección se traduciría en menores costos operativos y en mayor cumplimiento de regulaciones como la LOPDP y el GDPR.

La mejora en la satisfacción del equipo técnico favorecería la adopción organizacional del enfoque, consolidando una cultura de seguridad sostenible.

Estos resultados sugieren que la implementación de Seguridad por Diseño no solo es factible en entornos de desarrollo académico-aplicado, sino que también puede escalar hacia escenarios industriales donde la seguridad es un requisito estratégico.

Capítulo 4

4.1 Conclusiones

Tras aplicar las fases de diagnóstico, diseño, implementación piloto y evaluación, y en correspondencia con los objetivos planteados en el Capítulo 1, se presentan las siguientes conclusiones primordiales:

Se comprobó que los ocho principios de Saltzer y Schroeder, junto con marcos como OWASP SAMM e ISO/IEC 27001, constituyen una base sólida para prevenir vulnerabilidades desde las etapas iniciales del SDLC. Su adopción en las prácticas de la fábrica de software analizada permitió evidenciar carencias que explicaban la recurrencia de vulnerabilidades críticas.

Además, se concluyó que metodologías como Scrum y XP, si bien facilitan entregas rápidas, requieren ajustes específicos para incluir controles de seguridad, mientras que los enfoques tradicionales como Cascada integran seguridad en fases tempranas, pero con rigidez frente a cambios. La integración de Seguridad por Diseño aportó un esquema híbrido que equilibró flexibilidad y robustez.

Por otro lado, el procedimiento de codificación segura y la integración de herramientas como SonarQube y OWASP ZAP redujeron en un 61 % el tiempo medio de corrección de vulnerabilidades y en más del 50 % la densidad de defectos por KLOC. Esto valida que las buenas prácticas propuestas son aplicables y efectivas en entornos locales.

De igual manera, el plan piloto con dos grupos demostró que la integración de Seguridad por Diseño reduce significativamente el número de vulnerabilidades críticas y mejora la percepción de calidad del equipo de desarrollo. El Grupo B alcanzó un 84 % de cumplimiento en el checklist de codificación segura, frente al 0 % del Grupo A. Estos resultados confirman la hipótesis central de la investigación.

4.2 Recomendaciones

Tras culminar lo planificado en la propuesta, se formulan las siguientes recomendaciones primordiales para fortalecer y ampliar los alcances de este trabajo:

1. Extender la validación a proyectos de producción. Replicar el piloto en aplicaciones críticas de sectores como banca, comercio electrónico o servicios públicos, con el fin de comprobar la sostenibilidad de la metodología en entornos reales de alto impacto.
2. Incorporar métricas económicas. Complementar los indicadores técnicos con análisis de costos de reprocesos, licencias y tiempo invertido, a fin de demostrar el beneficio financiero de la Seguridad por Diseño.
3. Implementar programas de capacitación continua. Establecer entrenamientos regulares en codificación segura, análisis de amenazas y gestión de riesgos para mantener la vigencia de las prácticas seguras en la organización.
4. Explorar herramientas comerciales avanzadas. Incluir soluciones como Burp Suite o Fortify en futuros pilotos, para ampliar la cobertura de vulnerabilidades y fortalecer el análisis en escenarios más complejos.
5. Ajustar las limitaciones del estudio. Ampliar la muestra de proyectos, extender la duración del piloto a más ciclos de desarrollo y evaluar la metodología en equipos de diferente naturaleza, lo que permitirá obtener resultados más generalizables.

Referencias

[1] A. Pressman, *Ingeniería de Software: Un Enfoque Práctico*, 9.ª ed., McGraw-Hill, 2020.

[2] Superintendencia de Protección de Datos Personales, “Informe de incidentes de seguridad reportados en Ecuador,” Quito, 2023.

[3] ISO/IEC 27001:2022, “Information security, cybersecurity and privacy protection — ISMS Requirements,” ISO, 2022.

[4] OWASP Foundation, “Security by Design Principles,” 2023. [Online]. Disponible: https://owasp.org

[5] Universidad de las Fuerzas Armadas ESPE, “Buenas prácticas en desarrollo seguro de software en empresas ecuatorianas,” Tesis de Maestría, 2021.

[6] A. González et al., “Análisis de riesgos en aplicaciones desarrolladas por PYMES tecnológicas en Ecuador,” *Revista Tecnológica ESPOL*, vol. 36, no. 2, pp. 45-53, 2022.

[7] Asamblea Nacional del Ecuador, “Ley Orgánica de Protección de Datos Personales,” Registro Oficial No. 459, 26 mayo 2021.

[8] J. H. Saltzer and M. D. Schroeder, “The Protection of Information in Computer Systems,” *Communications of the ACM*, vol. 17, no. 7, pp. 388–402, 1974.

[9] J. R. Mead, “Secure Software Development in Agile Projects: A Practitioner’s Perspective,” *Journal of Software Engineering and Applications*, vol. 14, no. 5, pp. 215–229, 2021.

[10] OWASP Foundation, “OWASP SAMM v2.0,” 2023. [Online]. Disponible: https://owasp.org/www-project-samm/

[11] Microsoft, “Security Development Lifecycle (SDL),” 2022.

[12] NIST, “Security and Privacy Controls for Information Systems,” SP 800-53 Rev. 5, 2020.

[13] IBM System Sciences Institute, The relative cost of fixing software defects, 1978. [Online]. Available: https://www.functionize.com/blog/the-cost-of-finding-bugs-later-in-the-sdlc

[14] V. Basili, F. Shull and F. Lanubile, "Building Knowledge through Families of Experiments," IEEE Transactions on Software Engineering, vol. 25, no. 4, pp. 456-473, 1999.

[15] OWASP Foundation, OWASP Software Assurance Maturity Model (SAMM), 2020. [Online]. Available: https://owaspsamm.org

[16] ISO/IEC 27034-1:2011, Information technology — Security techniques — Application security — Part 1: Overview and concepts. Geneva: ISO, 2011.

[17] OWASP Foundation, Application Security Verification Standard (ASVS), v4.0, 2019. [Online]. Available: https://owasp.org/ASVS

[18] CERT, CERT Secure Coding Standards, Carnegie Mellon University, 2020. [Online]. Available: https://wiki.sei.cmu.edu

Apéndice A

Guía 1. Entrevista semiestructurada (Diagnóstico inicial)

Objetivo de la entrevista: Identificar brechas de seguridad en los procesos de desarrollo actuales.

Duración estimada: 30 – 40 minutos por participante.

Sección 1. Seguridad en requisitos y backlog

1. ¿Incluyen actualmente requisitos de seguridad en las historias de usuario?
2. ¿Quién define los criterios de seguridad en el backlog?
3. ¿Qué tan frecuentes son las revisiones de seguridad en la planificación de sprint?

Sección 2. Herramientas y prácticas actuales

4. ¿Qué herramientas utilizan para detectar vulnerabilidades (estáticas o dinámicas)?
5. ¿Qué dificultades encuentran al usarlas?
6. ¿Con qué frecuencia se ejecutan análisis de seguridad?

Sección 3. Gestión de vulnerabilidades

7. ¿Cómo se registran y priorizan los hallazgos de seguridad?
8. ¿Existen métricas de calidad del código relacionadas con seguridad?
9. ¿Cómo se reportan y corrigen los errores críticos antes del despliegue?

Sección 4. Barreras y percepción del equipo

10. ¿Cuáles considera que son las principales barreras para aplicar buenas prácticas de seguridad en el día a día?
11. ¿Qué nivel de capacitación cree que tiene el equipo en temas de seguridad por diseño?
12. ¿Qué sugerencias daría para mejorar la seguridad en los procesos actuales?

Apéndice B

Guía 2. Cuestionario estructurado (Validación del piloto)

Objetivo de la entrevista: Evaluar la percepción del equipo tras aplicar el esquema metodológico de Seguridad por Diseño.

Duración estimada: 30 – 40 minutos por participante.

Instrucciones: Indique su nivel de acuerdo con las siguientes afirmaciones, donde: 1 = Muy en desacuerdo, 2 = En desacuerdo, 3 = Neutral, 4 = De acuerdo, 5 = Muy de acuerdo.

Preguntas:

1. Las historias de usuario fueron más claras y completas al incluir requisitos de seguridad.

1	2	3	4	5

2. Las herramientas utilizadas (SonarQube, OWASP ZAP) resultaron fáciles de aplicar en el proceso de desarrollo.

1	2	3	4	5

3. La integración de controles de seguridad en las ceremonias ágiles fue sencilla de adoptar.

1	2	3	4	5

4. La carga de trabajo adicional asociada a la seguridad fue razonable y manejable.

1	2	3	4	5

5. La seguridad se convirtió en un aspecto transversal del proyecto, y no en una actividad aislada.

1	2	3	4	5

6. Confío en que la aplicación del esquema propuesto redujo efectivamente las vulnerabilidades del software.

1	2	3	4	5

7. Considero que este enfoque debería mantenerse en futuros proyectos de la organización.

1	2	3	4	5

Apéndice C

Guía 3. Checklist de codificación segura

Objetivo: Verificar de manera rápida y estructurada el cumplimiento de las prácticas de seguridad en cada iteración.

Tabla B.1 Checklist de codificación segura

Tabla C.1

Checklist de codificación segura

Área de control	Verificación requerida	Cumplido (✓/✗)	Observaciones
Validación de entradas	¿Se validan todas las entradas de usuario en servidor y cliente?		
	¿Se aplica sanitización de datos para prevenir inyecciones SQL y XSS?		
Gestión de autenticación y sesiones	¿Se implementa autenticación multifactor o al menos doble factor?		
	¿Las contraseñas están cifradas con algoritmos seguros?		
	¿Las sesiones tienen tiempos de expiración definido y regeneración de tokens?		
Manejo de errores	¿Los mensajes de error mostrados al usuario son genéricos y no revelan información interna?		
	¿Los errores críticos se registran en logs con trazabilidad, sin exponer datos sensibles?		
Protección de datos	¿Se cifra la información		

Área de control	Verificación requerida	Cumplido (✓/✗)	Observaciones
sensibles	sensible tanto en tránsito como en reposo?		
	¿Se evita el hardcode de credenciales en el código fuente?		
Uso de dependencias y librerías	¿Se validan las líbrerias externas con herramientas como OWASP Dependency-Check?		
	¿Se reigstran intentos fallidos de acceso y acciones críticas del sistema?		
	¿Se han implementado alertas en tiempo real para eventos de seguridad críticos?		

Nota: Elaboración propia.