INCEPTUM
Revista de Investigación en Ciencias de la Administración
Vol. XIX No. 37 Julio – Diciembre 2024

Propiedades del software seguro en el ciclo de vida del desarrollo de sistemas: una fundamentación teórica

Secure software properties in the systems' development lifecycle: a theoretical foundation

DOI: https://doi.org/10.33110/ inceptum.v19i2.457
(Recibido: 29/07/2024; Aceptado: 11/12/2024)

Fernando Villaseñor Béjar[1]

Víctor Hugo Vieyra Avilés[2]

Miryam Georgina Alcalá Casillas[3]

Resumen

Este artículo pretende aportar una fundamentación teórica sobre el impacto positivo de integrar la ciberseguridad en el ciclo de vida del desarrollo de un *software* -programa o sistema informático-; ésta debe ser incluida en las diversas propiedades del programa informático, así como en cada una de las fases de las metodologías de desarrollo ágil. De tal forma, en este análisis cualitativo se abordarán modelos, estándares y metodologías, cuyo enfoque es precisamente la ciberseguridad en el ciclo de vida de desarrollo de *software* (SDLC, por sus siglas en inglés). Este análisis cualitativo intenta puntualizar aspectos como las necesidades particulares de la organización, indicadores, tiempo de implementación, recursos, rendimiento, curva de aprendizaje y otros elementos relevantes que se deben considerar en el SDLC.

[1] Instituto Tecnológico de Morelia, México. Correo electrónico: fvillasenor@morelia.tecnm.mx
[2] Universidad Michoacana de San Nicolás de Hidalgo, México. ORCID: https://orcid.org/0009-0004-6462-2076 Correo electrónico: victor.vieyra@umich.mx
[3] Universidad Michoacana de San Nicolás de Hidalgo, México. ORCID: https://orcid.org/0000-0001-6963-8991 Correo electrónico: miryam.alcala@umich.mx

Palabras Clave: ciberseguridad, desarrollo de software seguro.

Abstract

This article aims to provide a theoretical foundation on the positive impact of integrating cybersecurity into the life cycle of software development -computer program or system-. IT security must be included in the various properties of a computer program, as well as in the different stages of agile development methodologies. Thus, various models, standards, frameworks and methodologies, whose focus is security in the software development life cycle (SDLC), will be addressed. This qualitative analysis attempts to cover aspects such as particular needs of the organization, indicators, implementation time, resources, performance, learning curve and other relevant elements that must be considered in the SDLC.

Keywords: cybersecurity, secure software development life cycle.

JEL: L86, M15.

Introducción

Para construir un "*software* seguro" se tienen enormes desafíos: por un lado, el desarrollo de programas sin protocolos de ciberseguridad ha incrementado las vulnerabilidades y los riesgos de seguridad tanto en sistemas y aplicaciones, como en las redes de cómputo; por otro lado, la ausencia de ciberseguridad contribuye a que se propaguen *softwares* maliciosos como gusanos o virus diseminados por usuarios denominados *hackers* (Al-Matouq *et al.*, 2020).

De acuerdo con Khan *et al.*, (2022), dentro del paradigma de ingeniería de programas informáticos, la seguridad se debe incorporar desde el comienzo del ciclo de vida de su desarrollo: desde el proceso de diseño y construcción, en los procesos y métodos, hasta la fase de pruebas; al mismo tiempo, los modelos de desarrollo deben integrar funcionalidad y respuestas preventivas a problemas de seguridad. No obstante, muchas organizaciones relegan las tareas de seguridad a las etapas finales del SDLC o como una etapa posterior, teniendo como resultado que los desarrolladores de los programas informáticos ejecuten el proceso de codificación sin considerar defectos de seguridad, vulnerabilidades y fallas que no pueden ser rastreadas o detectadas, además de análisis incompletos y diseños o métodos de desarrollo deficientes (Tung *et al.*, 2016).

Por su parte, Guio (2020) coincide en que, cuando se despliegan los proyectos en producción de programas informáticos, un factor común es que se realizan correcciones de errores en lugar de integrar la prevención desde las fases de desarrollo, ocasionando ciclos repetitivos, revisiones de diseño innecesarios y ajustes o actualizaciones que retrasan las entregas, afectan acuerdos contractuales e incrementan costos.

En los 70´s, con la creación de los primeros sistemas operativos especialmente en el sector comercial, se llevaron a cabo los primeros esfuerzos para integrar seguridad en el desarrollo de software; desde entonces, se ha experimentado una constante evolución debido al incremento de amenazas cibernéticas, además de que se han sofisticado los ataques informáticos pues, a partir de los 90´s con la expansión de la internet en todo tipo de organizaciones, se intensificó el enfoque y atención en la ciberseguridad, por la necesidad de proteger los sistemas y datos de cualquier ataque o injerencia (OEA, 2019).

Desde entonces, se han establecido diversas metodologías, prácticas y estándares para integrar la seguridad en todas las etapas del SDLC (Micucci, 2023), como la gestión de vulnerabilidades, las pruebas de seguridad y la educación continua del personal de desarrollo (IBM I. B., 2023).

Las metodologías de desarrollo han sido muy claras y concretas en tener como objetivo primordial el construir un producto que cumpla básicamente con las necesidades de la lógica del negocio, sin embargo, como se ha mencionado, están dejando a la seguridad de la información en una fase final del desarrollo o, incluso, posterior; olvidando que la ciberseguridad es un elemento de gran relevancia. Por ello, el objetivo de este análisis es describir el fundamento teórico de la seguridad en el SDLC identificando, desde las perspectivas de diferentes autores y marcos internacionales, la integración de las propiedades de seguridad y las etapas del desarrollo seguro. Para lograr dicho objetivo se plantean las siguientes preguntas de investigación:

- *¿Cuáles son las propiedades de seguridad que debe tener un sofware?*
- *¿Qué metodologías, modelos, marcos de trabajo o estándares se enfocan en la seguridad del ciclo de vida del desarrollo de software?*
- *¿Cuáles son las fases y actividades del desarrollo de software seguro?*

Materiales y métodos

La metodología aplicada en esta investigación fue interpretativa y descriptiva, con técnicas de recopilación de información, de selección, de análisis de diversas fuentes documentales científicas y revisión bibliográfica, sometiendo la información recabada a un proceso de exploración, interpretación y argumentación (Guirao et al., 2008).

Por ser una revisión del fundamento teórico, el tipo de fuente es secundaria; porque el origen de los documentos corresponde a diferentes bases de información y motores de búsqueda, de revistas académicas y científicas, con validez y publicación.

Tabla 1. *Principales fuentes utilizadas para en el análisis de la literatura*

Fuente	Denominación
Librerías Digitales	EEE Xplore, ACM Digital Library, Scholar Google

Fuente: Elaboración propia con información tomada de EEE Explore, ACM Digital Library y Google Académico (2024)

El objetivo del presente documento es identificar las metodologías de desarrollo de software seguro, puntualizando en los desarrollos ágiles, el fundamento teórico de sus elementos, así como los aspectos y propiedades de seguridad que son esenciales para los modelos, marcos de trabajo o metodologías, para lo cual se han realizado las siguientes fases:

Figura 1. *Fases o etapas realizadas en la metodología de la investigación*

Fase 1: Revisión sistemática de la Literatura	Fase 2: Selección y síntesis de la información	Fase 3: Análisis y Comparación de información	Fase 4: Discusión y Conclusiones
• Antecedentes y estudios previos. • Publicaciones. • Artículos científicos. • Libros. • Congresos y conferencias.	• Creación de fichas bibliográficas. • Organización de la información.	• Síntesis de los temas con sustento de autores. • Análisis comparativo de los tópicos en general.	• Discusión, argumentación y conclusión de las propiedades del sofware seguro, características de las metodologías, modelos o marcos de trabajo.

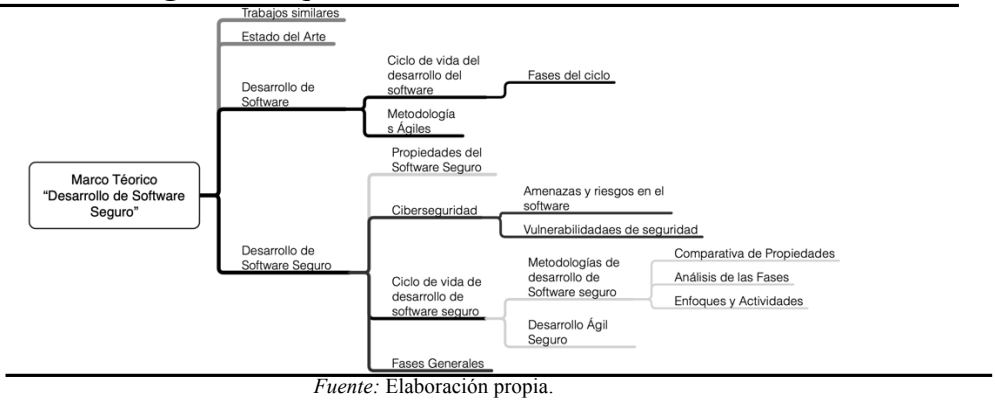
Fuente: Elaboración propia basado en Hernández et al (2014)

Es importante enfatizar y contextualizar el fundamento teórico que soporta los supuestos de investigación, de tal manera que al haber realizado una revisión de la literatura se han detectado las categorías que por el método deductivo se habían establecido así como categorías emergentes.

Resultados

El resultado principal de este análisis es el marco teórico del Desarrollo de Software Seguro, fundamentado en autores, estándares y organizaciones internacionales que tienen como objetivo implementar y proporcionar buenas prácticas y protocolos de seguridad en la construcción del software. De manera organizada, en la Figura 2 se muestran las categorías y subcategorías de investigación mapeadas como tópicos obligados para el sustento teórico.

Figura 2. *Esquematización del Marco Teórico*



Fuente: Elaboración propia.

Antecedentes y trabajos relacionados

Durante el análisis de la literatura, se detectó una cantidad considerable de trabajos o investigaciones realizadas con objetivos similares a esta investigación. Por ello, la información se organiza de acuerdo con diversos enfoques en torno a la seguridad en el desarrollo de software.

Desde un enfoque del *Ciclo de vida del desarrollo de software seguro (S-SDLC)* McGraw y Chess (2003), de los más mencionados en la literatura previa, abogan por la

integración de prácticas de seguridad en todo el ciclo de vida del desarrollo de software. Parte de sus trabajos incluye "*Secure Programming Cookbook for C and C++*" y "*Software Security: Building Security In*". Mead y Allen (2007) han llevado a cabo una amplia investigación sobre la *Seguridad en la Ingeniería de requisitos*, incluyendo el desarrollo de métodos y técnicas para obtener, analizar y especificar los requisitos de seguridad en los sistemas de software.

Por su parte, Anderson (2001), desde la perspectiva de la *Seguridad en el Diseño y arquitectura,* realizó trabajo relevante en el campo con la obra "*Ingeniería de Seguridad: Una Guía para la Construcción de Sistemas Distribuidos Confiables*". Por su parte Howard y LeBlanc (2002), aportaron la obra "*Writing Secure Code*" o *Prácticas de codificación seguras*, que se publicó por primera vez en 2002 y se ha actualizado en ediciones posteriores. En el tema de *Pruebas y verificación de seguridad,* McGraw y Hoglund (2006) publican "*Exploiting Software: How to Break Code" y "Software Security: Building Security In*". En cuanto a la *Capacitación y concientización en seguridad,* Sasse y Furnell (1990) realizan trabajo que incluye varias publicaciones desde finales de la década de 1990 hasta la actualidad, centradas en la educación y la concienciación en materia de seguridad.

Uno de los tópicos de gran relevancia hoy día es *DevOps seguro (DevSecOps),* al cual Kim, Humble y Debois (2016) aportan con su libro "The DevOps Handbook" donde analizan la integración de las prácticas de seguridad en los procesos de DevOps. De igual manera, Hill y Miner (2014) publican aspectos relacionados con *Métricas y medición de seguridad,* ejemplo de esto es "Measuring and Managing Information Risk: A FAIR Approach" en el 2014.

El Cumplimiento y requisitos reglamentarios son factores necesarios en la seguridad del software. Así, para la primera década del 2000 ya existían algunas publicaciones centradas en el cumplimiento y los requisitos normativos en ciberseguridad, ejemplo de tales casos son Chuvakin y Gula (2001). Shostack y Stewart (2014) publican su libro "Threat Modeling: Designing for Security", el cual se enfoca en el tema de *Inteligencia de amenazas y gestión de riesgos.*

Robayo, B. (2020) propone una "Guía de principios y buenas prácticas para pruebas de seguridad de software en aplicaciones web para una empresa del sector privado", en la que desarrolla aplicaciones web que prestan servicios a diferentes entidades, se aborda la caracterización de los estándares, modelos y esquemas para pruebas de seguridad de software de aplicaciones web, entre los cuales se destacan: 1**. ISO/IEC 25010,** modelo de calidad para establecer el sistema para evaluar las propiedades de un producto software, el grado en el que satisface los requisitos de funcionalidad, rendimiento, seguridad, mantenibilidad, definido por la ISO/25000*; **2. Software Assurance Maturity Model / OWASP Open Web Application Security Project***, proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro; ***3. MITRE ATT&CK®,*** marco para modelar, detectar, prevenir y combatir las amenazas de ciberseguridad; y, ***4. ISO/IEC 27001***, estándar internacional que establece los requisitos para la implementación, mantenimiento y mejora continua de un Sistema de Gestión de la Seguridad de la Información, para proteger la confidencialidad, integridad y disponibilidad de la información (ISO, 2023).

Ciclo de vida del desarrollo de software (SDLC)

Como punto de referencia, es importante comprender qué es el desarrollo de software desde diferentes perspectivas; para esto, Pressman (2010) lo define como "el *proceso de*

especificación, diseño, programación, documentación y prueba de sistemas informáticos, que incluye aplicaciones, sistemas y programas en general, que resuelven un problema o realizan una tarea específica".

Por su parte, Sommerville (2011) lo conceptualiza como "*el establecimiento de un sistema que está diseñado y construido para cambiar el mundo físico o virtual que lo rodea, de una manera específica y deseada*". Mientras que para Boehm (1988) es "*el establecimiento de un conjunto de programas de software que realicen un conjunto de funciones definidas en forma de especificaciones precisas y que los programas se entreguen a los usuarios finales para su operación y uso*". Con un enfoque similar, David A. Patterson y John L. Hennessy (1994) afirman que es "*el proceso de crear o modificar los sistemas de software, incluyendo la construcción de sistemas nuevos, la renovación de sistemas existentes y el mantenimiento de sistemas existentes*".

Sin embargo, para la comprensión de este artículo, se toma como referente principal a Pressman (2010), quien explica de manera sencilla pero enfática que el SDLC está integrado por un proceso con etapas o fases para su construcción; y por ende, esas fases son la base de los modelos para el desarrollo de sistemas (Delgado & Díaz, 2020).

El proceso de construcción o desarrollo del software se denomina ciclo de vida y se compone de un conjunto de fases perfectamente delimitadas. Según el orden creciente, estas fases son: la toma de requisitos, el análisis, el diseño, la implementación o codificación, las pruebas, el despliegue y el mantenimiento. Dando sustento a las teorías del ciclo de vida del software, la siguiente tabla muestra una relación de autores con sus descripciones.

Tabla 2. *Definición de varios autores sobre el Ciclo de vida del software*

Autor	Definición
Pressman (2010)	Estructura de actividades requeridas para desarrollar un sistema de software.
Sommerville (2011)	Describe los procesos involucrados en la construcción de un software, desde los requisitos iniciales hasta la entrega del producto final.
Boehm (1988)	Proceso de desarrollo y mantenimiento de software desde la concepción inicial hasta el retiro final
Norma 1074 IEEE (1997)	Es una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software.
Norma ISO 12207 (2017)	Marco de referencia, que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso.

Fuente: Diseño del autor a partir de las referenciass citadas.

Posterior a las definiciones de la tabla anterior, se puede constatar que el proceso del ciclo de vida del software, consiste de un conjunto de fases perfectamente delimitadas, sin embargo, existen distintos modelos de ciclo de vida que determinan cuáles son y cómo se ejecutan las etapas del desarrollo de software. Cada modelo presenta sus propias características y alcances (Diéguez and Cares, 2012). Sin embargo se han considerado las siguientes etapas como un común denominador para el proceso de desarrollo:

Figura 3. *Fases generales del ciclo de vida del desarrollo de software*

Fuente: Elaboración propia, con información de Pressman (2010)

De acuerdo con Vijayasarathy y Butler (2015), las metodologías de desarrollo de software proveen un marco de trabajo para planificar, ejecutar y gestionar el proceso de desarrollo de sistemas. Adicionalmente, sirven como guía para los desarrolladores, equipo de control de calidad y líderes de proyectos; definiendo el ciclo de vida del software desde la obtención de los requisitos, análisis, diseño, implementación, pruebas hasta el despliegue (Delgado & Díaz, 2020).

Actualmente, las metodologías ágiles ofrecen un proceso de desarrollo de software muy flexible y rápido; sin embargo, se descuida la seguridad en favor de la adaptabilidad a los nuevos requisitos. Varios marcos de trabajo, enfoques y modelos de madurez de desarrollo de software seguro proporcionan cierto apoyo a los desarrolladores para incorporar la seguridad en entornos ágiles (Valdés *et al.*, 2023).

Adicionalmente, ofrecen apoyo a los clientes de forma continua, con entregas frecuentes, a tiempo y adaptables a los ambientes altamente cambiantes de hoy, en un ambiente colaborativo; pues predominan las interacciones entre los individuos, sinergizando procesos y herramientas; y documentando lo necesario, durante y al finalizar las implementaciones; con ello, incrementando la eficiencia y productividad de la orgganización. Así, las metodologías ágiles han remplazado a las metodologías tradicionales en organizaciones, proyectos o negocios donde las especificaciones no son claras, o bien, muy cambiantes; debido a la utilización de un enfoque iterativo e incremental, a la integración continua de código y a la aceptación de cambios constantes en los requerimientos de negocio (Cueva & Sucunuta, 2014).

Desde las etapas iniciales del desarrollo de un producto software, se trabaja enormemente en la funcionalidad que debe ofrecer el producto, el diseño de la interfaz de usuario, el tiempo de respuesta o la capacidad para atender la demanda. Todo ello para obtener un producto con la mayor calidad posible y con el objetivo de satisfacer los requerimientos, necesidades y expectativas del cliente.

Sin embargo, en el desarrollo de software, hoy en día se sigue descuidando con bastante frecuencia un aspecto que cada vez toma mayor relevancia: la seguridad (Casas, 2021). Gran parte del problema podría mitigarse teniendo en cuenta los aspectos de seguridad desde el inicio del proceso de desarrollo de software. Es decir, adoptando una visión preventiva en lugar de correctiva. Lo cual se puede lograr a través de la gestión de vulnerabilidades, que es una rama muy importante de la ciberseguridad. MITRE Corporation, en 1999 y con apoyos por parte de la División Nacional de Seguridad Cibernética del Departamento de Seguridad Nacional de los Estados Unidos, fue la que comenzó a asignar claves únicas, bajo el esquema CVE (Common Vulnerabilities and Exposures), a las vulnerabilidades más comunes o conocidas en ese momento. Lo anterior para facilitar la identificación de potenciales debilidades o riesgos de seguridad, para evitarlos, o bien,

mitigarlos. Estos números o claves identificadoras (ID de CVE) también son utilizados en el protocolo SCAP (Security Content Automation Protocol), utilizado por las herramientas de gestión de vulnerabilidades (BST, 2024).

Ciclo de vida de desarrollo de software seguro

El *ciclo de vida de desarrollo de software seguro (SSDLC - Secure Software Development Life Cycle)* es uno de los temas de mayor relevancia en esta investigación puesto que, como se ha señalado, existen modelos, marcos de trabajo, estándares y metodologías que proponen actividades y controles específicos para la seguridad durante las fases tradicionales del ciclo de vida; de forma que el SSDLC puede vislumbrarse a través de seis fases (CyberSecurity Malaysia, 2020):

- Fase de requisitos de seguridad
- Diseño de seguridad
- Desarrollo de seguridad
- Pruebas de seguridad
- Implementación de seguridad
- Mantenimiento de seguridad (Sancho, 2020)

El estándar ISO/IEC 27034 (2018) proporciona directrices específicas para la seguridad del desarrollo de software, de tal manera que se toman como referencia las propiedades para el análisis y comparación entre las diferentes metodologías ya existentes. Por lo tanto, las propiedades y atributos que debe tener el software seguro, de acuerdo con Marulanda y Ceballos (2012) son:

- Ejecución Predecible. La certeza de que cuando se ejecute, funcione como se había previsto. Reducir o eliminar la posibilidad de que entradas maliciosas alteren la ejecución o las salidas.
- Fiabilidad. La meta es que no haya vulnerabilidades que se puedan explotar.
- Conformidad. Actividades planeadas, sistemáticas y multidisciplinarias que aseguren que los componentes y productos de software están conforme a los requisitos, procedimientos y estándares aplicables para su uso específico.
- Confidencialidad. Debe asegurar que cualquiera de sus características, activos y/o contenidos no sean accesibles para no autorizados.
- Integridad. El software y sus activos deben ser resistentes a subversión, es decir, llevar a cabo modificaciones no autorizadas al código fuente, activos, configuración o comportamientos, o cualquier modificación por entidades no autorizadas.
- Disponibilidad. El software debe ser funcional y accesible para usuarios autorizados cada vez que sea necesario. Al mismo tiempo, esta funcionalidad y privilegios deben ser inaccesibles por usuarios no autorizados
- Responsabilidad. Todas las acciones relevantes de seguridad del software o de los usuarios se deben almacenar y rastrear, con atribución de responsabilidad.
- No repudio. Esta propiedad le permite al software y a los usuarios refutar o denegar responsabilidades de acciones que ha ejecutado. Esto asegura que la responsabilidad no puede ser derribada o evadida (Marulanda & Ceballos, 2012)

Tabla 3. *Aspectos o características del desarrollo de software seguro*

Aspecto	Descripción
Gestión de riesgos de seguridad del software	Identificación, evaluación y tratamiento de los riesgos de seguridad del software a lo largo de todo el ciclo de vida del desarrollo.
Requisitos de seguridad del software	Definición clara de los requisitos de seguridad del software desde las etapas iniciales del ciclo de vida del desarrollo para garantizar que se tengan en cuenta en todas las fases del proceso (Berzal & Cortijo Bon, 2005).
Diseño seguro	Integración de prácticas de diseño seguro que aborden los riesgos de seguridad del software y aseguren la protección de los activos de información. (Matei, 2015)
Implementación segura	Implementación de controles de seguridad apropiados durante el desarrollo del software para mitigar riesgos y prevenir vulnerabilidades. (Thales Group, 2024)
Pruebas de seguridad	Realización de pruebas de seguridad regulares para identificar y corregir vulnerabilidades en el software antes de su implementación. (Thales Group, 2024)
Gestión de cambios seguros	Implementación de un proceso de gestión de cambios que tenga en cuenta los aspectos de seguridad y garantice que los cambios no introduzcan nuevas vulnerabilidades.
Seguridad del suministro	Evaluación y gestión de la seguridad de los componentes de software suministrados por terceros para mitigar riesgos asociados con la dependencia de software externo.
Mantenimiento seguro	Implementación de prácticas de mantenimiento seguro para garantizar que el software siga siendo seguro y protegido contra nuevas amenazas y vulnerabilidades a lo largo del tiempo. (SENTRIO, 2023)
Monitorización y mejora continua	Establecimiento de procesos para monitorizar el rendimiento del software en términos de seguridad y realizar mejoras continuas para abordar nuevas amenazas y desafíos.

Fuente: Diseño del autor a partir de RedHat (2024), SENTRIO (2023), Thales Group (2024), Matei (2015), Berzal y Cortijo (2005)

Modelos, Metodologías y Marcos de trabajo del Desarrollo de Software Seguro

Existen varios modelos, metodologías y marcos de trabajo enfocados a integrar prácticas de seguridad en todas las etapas del SDLC y, aunque existe confusión en la terminología con la que se les denomina, de acuerdo con Shaw (2003), es necesario categorizar el tipo de contribución que se tiene, con base en estas prácticas de seguridad en el desarrollo del software. La Tabla 4 muestra la terminología que se puede emplear para denominar estas prácticas (Microsoft, 2024).

Tabla 4. *Terminología para categorizar las prácticas de seguridad*

Tipo de contribución	Definición	Sinónimo
Modelo	Representación de una realidad observada por conceptos o	Modelo, conceptos, proceso,

	conceptos relacionados después de un proceso de conceptualización	conceptualización de la estrategia de enseñanza.
Teoría	Constructo de relaciones causa-efecto de resultados determinados	Teoría, causa-efecto
Marco/métodos	Modelos relacionados con la construcción de software o la gestión de procesos de desarrollo	Marco de trabajo, arquitectura, implementación, esquema
Encuesta	Investigación empírica a través de cuestionarios, entrevistas	Cuestionario, entrevista, empírico
Guías	Listado de consejos, síntesis de los resultados de investigación obtenidos	Asesoramiento, síntesis, mejores prácticas
Lección aprendida	Conjunto de resultados, analizados directamente a partir de los resultados de investigación obtenidos (Montes, 2018)	Opinión discursiva, genérica, personal
Consejos/implicaciones	Recomendaciones discursivas y genéricas, consideradas a partir de opiniones	Opinión discursiva, genérica, personal
Herramienta	Tecnología, programa o aplicación que se utiliza para crear, depurar, mantener o dar soporte a los procesos de desarrollo (Hosting Web Premium, 2023)	Herramienta, demostración, implementación, desarrollo, evaluación

Fuente: Elaboración propia a partir de Sancho N,. et al (2020)

Microsoft Security Development Lifecycle (SDL) es un marco de trabajo desarrollado por Microsoft que proporciona directrices detalladas para integrar la seguridad en todas las etapas del ciclo de vida del desarrollo de software. Se centra en actividades como la identificación de requisitos de seguridad, el diseño seguro, la codificación segura, las pruebas de seguridad y la respuesta a incidentes (Sancho, 2020).

El Modelo de Madurez de Software Assurance SAMM (2023) es un marco de trabajo abierto para ayudar a las organizaciones a formular e implementar una estrategia de seguridad para Software que sea adecuada a las necesidades específicas que está enfrentado la organización. Su objetivo es generar una postura general que capture la meta de aseguramiento al alcanzar el nivel de madurez asociado. Se basa en diferentes dimensiones de seguridad como la gobernanza, el diseño, la implementación y las operaciones, y proporciona una hoja de ruta para la implementación de prácticas de seguridad (Robayo, B, 2020).

Building Security In Maturity Model (BSIMM) es un marco de trabajo desarrollado por Synopsys que describe prácticas y actividades observadas en organizaciones que tienen programas de seguridad del software maduros. Se basa en un conjunto de dimensiones de seguridad y proporciona una evaluación comparativa de las actividades de seguridad del software en diferentes organizaciones (Synopsys, 2023) & (Thales Group, 2024).

Secure Software Development Framework (SSDF) de NIST, propuesto por el Instituto Nacional de Estándares y Tecnología (NIST). El SSDF proporciona un marco de

trabajo para integrar la seguridad en el ciclo de vida del desarrollo de software. Se centra en actividades como la gestión de riesgos, la identificación de requisitos de seguridad, el diseño seguro, la implementación segura y las pruebas de seguridad (Souppaya, Scarfone and Dodson, 2022) & (Matei, 2015).

ISO/IEC 27034 es una norma internacional que proporciona directrices para el aseguramiento de la seguridad en el desarrollo de aplicaciones. Esta norma, titulada "Tecnologías de la información - Técnicas de seguridad - Gestión de seguridad en aplicaciones", establece un marco de trabajo para ayudar a las organizaciones a integrar la seguridad en todas las etapas del ciclo de vida del desarrollo de software (ISO/IEC 27034-7:2018, 2018).

La metodología *Comprehensive Lightweight Application Security Process* (CLASP) trata la construcción de software como un conjunto de piezas especializadas en seguridad. Busca integrarse de manera sencilla con el resto de los procesos de desarrollo (Sancho, 2020). Destaca por la adaptación y conexión entre los procedimientos de construcción del software (Gregoire *et al.*, 2007).

Oracle Software Security Assurance (OSSA) es la metodología de Oracle para crear seguridad en el diseño, el desarrollo, la prueba y el mantenimiento de los productos; abarca cada fase del ciclo de vida del desarrollo del producto (Oracle, 2024).

Cada modelo de desarrollo de software descrito anteriormente tiene enfoques y características distintos, por lo que es necesario establecer a detalle un análisis que brinde información clara sobre sus propiedades (sharma, 2022). Las Tablas 5 y 6 muestran las propiedades y fases de las metodologías o modelos o marcos de trabajo.

Tabla 5. *Propiedades o cualidades de los marcos de trabajo del desarrollo de software seguro*

OWASP	SSDLC	BSIMM	Microsoft SDL	SAMM	NIST
· Autenticación y autorización seguras (Secure Authentication and Authorization): Se deben implementar mecanismos seguros de autenticación y autorización para garantizar que los usuarios solo tengan acceso a las funciones y datos que les corresponden.	· Requerimientos de seguridad claros (Clear Security Requirements): Establecer requisitos de seguridad específicos desde el principio del ciclo de desarrollo, de modo que los desarrolladores comprendan las expectativas en términos de seguridad.	· Gestión de la Seguridad: Establecimiento de una dirección estratégica y un liderazgo efectivo para el programa de seguridad de software. Esto incluye la adopción de políticas de seguridad y la asignación de recursos adecuados.	· Definición de Requerimientos de Seguridad: Establecer y documentar requisitos de seguridad claros desde el inicio del ciclo de desarrollo, para garantizar que la seguridad se incorpore en todas las etapas del proceso.	· Estrategia y Gobernanza (Strategy & Metrics): Establecer una estrategia de seguridad para el software y métricas para medir y mejorar la seguridad a lo largo del tiempo.	· Confidencialidad (Confidentiality): Asegurar que la información sensible o confidencial no se divulgue a usuarios no autorizados. Esto se logra mediante técnicas de cifrado y control de acceso. (Conzultek, 2016)

· Validación de entradas (Input Validation): Las aplicaciones deben validar y sanitizar adecuadamente todas las entradas del usuario para prevenir ataques de inyección, como la inyección de SQL y la inyección de código.	· Evaluación de riesgos (Risk Assessment): Realizar una evaluación de riesgos para identificar amenazas potenciales y vulnerabilidades en el software, lo que ayuda a priorizar las áreas que requieren una atención especial.	· Gestión de Estrategia y Métricas: La definición de una estrategia clara de seguridad de software y la medición del progreso a lo largo del tiempo utilizando métricas relevantes.	· Evaluación de Riesgos: Realizar evaluaciones de riesgos para identificar amenazas y vulnerabilidades potenciales en el software, lo que ayuda a priorizar la mitigación de riesgos. (Faster Capital, 2024)	· Concienciación (Governance & Policy Compliance): Crear concienciación en toda la organización sobre la importancia de la seguridad del software y garantizar el cumplimiento de políticas de seguridad.	· Integridad (Integrity): Garantizar que los datos y el software no sean modificados de manera no autorizada. Se utilizan mecanismos como las firmas digitales y la detección de cambios no autorizados. (Oficina de Seguridad para las Redes Informáticas, 2007)
· Gestión segura de sesiones (Session Management): Asegurar que las sesiones de usuario se gestionen de manera segura, lo que incluye la protección contra ataques de secuestro de sesión y la gestión adecuada de tokens de sesión.	· Diseño seguro (Secure Design): Diseñar el software con la seguridad en mente, considerando aspectos como la arquitectura segura, la segregación de datos y la minimización de la superficie de ataque. (Matei, 2015)	· Gobierno de Seguridad: Establecimiento de un gobierno efectivo para la seguridad del software, que incluye la toma de decisiones, la supervisión y la rendición de cuentas en toda la organización.	· Diseño Seguro: Diseñar el software con la seguridad en mente, incluyendo la definición de arquitecturas seguras y la identificación de áreas críticas para la seguridad. (Auditoría de Código, 2021)	· Desarrollo y Verificación (Software Development and Verification): Implementar prácticas de desarrollo seguro, como la revisión de código, la gestión de vulnerabilidades y las pruebas de seguridad.	· Disponibilidad (Availability): Asegurar que el software esté disponible y funcione correctamente cuando sea necesario. Esto incluye la redundancia y la planificación de la recuperación de desastres. (Alteryx, 2023)
· Control de acceso a nivel de objeto (Object-Level Access Control): Las aplicaciones deben aplicar controles de acceso para proteger los objetos y datos dentro de la aplicación, asegurando que los usuarios solo puedan acceder a lo que les corresponde.	· Desarrollo seguro (Secure Development): Aplicar buenas prácticas de desarrollo seguro, que incluyen la validación de	· Revisión y Cumplimiento Legal: Cumplimiento de requisitos legales y regulatorios relacionados con la seguridad de software, y realización de revisiones periódicas para garantizar el cumplimiento. (Faster Capital, 2024)	· Desarrollo Seguro: Aplicar buenas prácticas de desarrollo seguro, como la validación de entradas, la prevención de inyección de código y la codificación segura. (Omatech, 2022)	· Gestión de Datos (Data Security): Asegurar la seguridad de los datos almacenados y transmitidos por el software, incluyendo el cifrado de datos y la gestión segura de bases de datos.	· Autenticación (Authentication): Verificar la identidad de los usuarios y sistemas para garantizar que solo las personas o sistemas autorizados tengan acceso al software.
		· Seguridad en el Desarrollo: Integración de prácticas de seguridad en el ciclo de vida del	· Pruebas de Seguridad: Realizar pruebas de seguridad, como análisis estático		

	desarrollo de software, incluyendo	y dinámico del código, pruebas de penetración y evaluaciones de seguridad, para identificar y remediar (IBM)	

Fuente: Elaboración propia a partir de Owasp (2024), BSIMM, SAMM (2024) y Nist, 2024) , Microsoft SDL (2024) y adicionalmente los referentes citados.

Tabla 6. *Fases de los marcos de trabajo del desarrollo de software seguro*

Etapas o fases	Microsoft SDL/ASDL	OSSA	CLASP	NIST	SAMM	BSIMM
Gobernanza			x		x	x
Formación y Capacitación	x	x	x	x	x	x
Análisis	x	x	x	x	x	x
Diseño	x	x	x	x		
					x	x
Implementación	x	x				x
			x	x		
					x	
Pruebas	x		x	x	x	x
Lanzamiento o entrega (previa)	x	x	x			
Después del lanzamiento	x	x	x	x	x	x

Fuente: Elaboración propia a partir de Owasp (2024), BSIMM, SAMM (2024) y Nist, 2024) , Microsoft SDL (2024) y adicionalmente los referentes citados.

Discusión

Se observa que los modelos de desarrollo de software seguro son marcos y enfoques que buscan integrar la seguridad en todas las etapas del ciclo de vida del desarrollo de software. Sin embargo, el SDL de Microsoft es uno de los modelos de desarrollo seguro más conocidos y ampliamente adoptados. Se centra en la integración de prácticas de seguridad en todas las etapas del ciclo de vida del desarrollo de software, desde la planificación y el diseño hasta la implementación y el mantenimiento. Ha sido fundamental para mejorar la seguridad de los productos de Microsoft y ha influido en la industria en general al establecer un estándar para el desarrollo seguro de software.

En contraste, SAMM proporciona un marco de trabajo para evaluar y mejorar las capacidades de seguridad del software en una organización. Se centra en las prácticas de seguridad en los aspectos organizativos, de diseño y de implementación del desarrollo de software. Entre su documentación, se afirma que ayuda a las organizaciones a evaluar su postura de seguridad en el desarrollo de software y a priorizar las áreas de mejora, lo que resulta en aplicaciones más seguras y resistentes (TARLOGIC SECURITY, 2023).

Por su parte, BSIMM es un modelo que describe las prácticas de seguridad observadas en organizaciones que tienen programas de seguridad de software maduros. Ofrece una visión de las actividades de seguridad de diferentes organizaciones y permite a otras identificar áreas para mejorar. Un impacto positivo es que proporciona una guía útil para las organizaciones que desean mejorar su postura de seguridad en el desarrollo de

software, permitiéndoles compararse con otras empresas y adoptar las mejores prácticas de la industria.

La seguridad ágil se centra en integrar la seguridad en los procesos ágiles de desarrollo de software. Busca abordar los desafíos de seguridad de manera iterativa y colaborativa, garantizando que las prácticas de seguridad estén alineadas con los principios de agilidad.

Permite a las organizaciones desarrollar software de manera rápida y segura, abordando las preocupaciones de seguridad desde el principio y adaptándose a los cambios en los requisitos y amenazas durante el ciclo de vida del desarrollo (Laoyan, 2024).

La implementación de modelos de desarrollo de software seguro puede ser costosa y compleja. Esto se debe a la necesidad de capacitar al personal, establecer procesos y herramientas específicas, y realizar auditorías y evaluaciones de seguridad, lo que puede aumentar los costos y la complejidad del desarrollo de software. Se considera también que algunos modelos de desarrollo de software seguro pueden ser percibidos como rígidos y poco adaptables a los entornos ágiles y dinámicos de desarrollo de software. La rigidez en los procesos y controles de seguridad puede obstaculizar la innovación y la velocidad de entrega del software, lo que lleva a la resistencia por parte de los equipos de desarrollo.

Algunos modelos hacen hincapié en el uso de herramientas automatizadas para pruebas de seguridad y análisis estático de código. Sin embargo, *la sobre-dependencia en estas herramientas puede llevar a una falsa sensación de seguridad y pasar por alto ciertas vulnerabilidades que solo pueden ser detectadas mediante pruebas manuales y revisión humana del código* (TARLOGIC SECURITY, 2023).

Todos los modelos de desarrollo de software seguro juegan un papel crucial en la mejora de la seguridad del software al proporcionar marcos estructurados y enfoques sistemáticos para integrar la seguridad en todas las etapas del ciclo de vida del desarrollo. Al adoptar estos modelos, las organizaciones pueden reducir los riesgos de seguridad y construir softwares más seguros y resilientes. El desarrollo de software seguro presenta una serie de desafíos, como la complejidad de las aplicaciones modernas, la falta de conciencia sobre las mejores prácticas de seguridad entre los desarrolladores, la presión para entregar productos rápidamente y la falta de recursos dedicados a la seguridad. Pero también se puede argumentar que factores como el tiempo, el recurso humano, la cantidad de proyectos y la habilidad del desarrollador son necesarios considerar durante el desarrollo (Zavala-Quinones, 2024).

El uso de estándares y marcos de trabajo de seguridad como ISO 27001, NIST SP 800-53, y marcos como OWASP, BSIMM, CLASP, pueden proporcionar una guía estructurada para integrar la seguridad en el desarrollo de software y garantizar que se sigan las mejores prácticas. La seguridad en el desarrollo de software es un tema complejo que requiere una atención cuidadosa y un enfoque proactivo por parte de las organizaciones. Al integrar la seguridad desde el inicio, fomentar la colaboración entre equipos, mejorar la educación y concienciación sobre seguridad, y utilizar estándares y marcos de trabajo de seguridad, las organizaciones pueden mitigar los riesgos de seguridad y construir sistemas de software más seguros y resilientes.

Conclusiones

Las metodologías de desarrollo de software seguro son fundamentales para garantizar que los sistemas de software se construyan con un enfoque proactivo en la seguridad desde el principio hasta el final del ciclo de vida del desarrollo; adicionalmente, promueven la mejora continua en las prácticas de seguridad, al revisar y actualizar regularmente las políticas, procedimientos y herramientas de seguridad para hacer frente a las amenazas emergentes y los cambios en el panorama de seguridad; aunado a ello, fomentan la colaboración entre equipos multidisciplinarios, que incluyen desarrolladores, ingenieros de seguridad, administradores de sistemas y otros profesionales; colaboración que garantiza que se aborden elementos de seguridad de manera integral y se compartan las responsabilidades (Zavala-Quinones, 2024)

Por su parte, la automatización de pruebas de seguridad es una parte fundamental de muchas metodologías de desarrollo seguro, pues permite identificar vulnerabilidades y debilidades de seguridad de manera rápida y eficiente, lo que facilita su corrección antes de que lleguen a la producción (Plaza, 2024).

En resumen, las metodologías de desarrollo de software seguro proporcionan un marco estructurado y sistemático para construir sistemas de software resilientes y protegidos contra las amenazas cibernéticas (Micucci, 2023). Al adoptar consideraciones de seguridad desde la etapa de concepción y diseño del sistema en estas metodologías, las organizaciones pueden reducir el riesgo de vulnerabilidades y mejorar la confiabilidad y seguridad de sus aplicaciones y sistemas (Azurza, 2020)**.**

Referencias

ACM Digital Library (2024) *ACM Digital Library*. Available at: https://dl.acm.org/ (Accessed: 4 June 2024).

Al-Matouq, H. *et al.* (2020) 'A Maturity Model for Secure Software Design: A Multivocal Study', *IEEE Access*, 8, pp. 215758–215776. Available at: https://doi.org/10.1109/ACCESS.2020.3040220.

Campos, F. *et al.* (2013) 'Enfoque de sistemas: Proceso seguro para el desarrollo de software ágil'.

Casas, M. (2021) 'Madurez del ciclo de vida del desarrollo de software seguro: OWASP Software Assurance Maturity Model (SAMM)'.

Clasp Owasp - CLASP OWASP Autores: Beiker Santorum, Jaime Paqui Julio 2022 1. Introducci´on En la - Studocu (no date). Available at: https://www.studocu.com/ec/document/universidad-nacional-de-loja/seguridad-de-la-informacion/clasp-owasp/39430268 (Accessed: 19 March 2024).

CyberSecurity Malaysia (2020) 'Guidelines for Secure Software Development Life Cycle (SSDLC) Reference'.

Diéguez, M. and Cares, C. (2012) 'De la Gestión de Seguridad en el Ciclo de Vida del Software', in.

Fujdiak, R. *et al.* (2019) 'Managing the Secure Software Development', in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, CANARY ISLANDS, Spain: IEEE, pp. 1–4. Available at: https://doi.org/10.1109/NTMS.2019.8763845.

Google Académico (2024) *Google Académico*. Available at: https://scholar.google.es/schhp?hl=es (Accessed: 4 June 2024).

Gregoire, J. *et al.* (2007) 'On the Secure Software Development Process: CLASP and SDL Compared', in, pp. 1–1. Available at: https://doi.org/10.1109/SESS.2007.7.

Guio, N.S.B. (2020) 'ANÁLISIS COMPARATIVO ENTRE METODOLOGÍAS PARA EL DESARROLLO SOFTWARE SEGURO DE ACUERDO CON EL ESTÁNDAR ISO/IEC 15408'.

IEEE Standards Association (no date) *IEEE Standards Association*. Available at: https://standards.ieee.org (Accessed: 4 June 2024).

IEEE Xplore (2024) *IEEE Xplore*. Available at: https://ieeexplore.ieee.org/Xplore/home.jsp (Accessed: 4 June 2024).

ISO/IEC 27034-7:2018 (2018) *ISO/IEC 27034-7:2018(en), Information technology — Application security — Part 7: Assurance prediction framework*. Available at: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:27034:-7:ed-1:v1:en (Accessed: 19 March 2024).

Jarzębowicz, A. and Weichbroth, P. (2021) 'A Qualitative Study on Non-Functional Requirements in Agile Software Development', *IEEE Access*, 9, pp. 40458–40475. Available at: https://doi.org/10.1109/ACCESS.2021.3064424.

Khan, R.A. *et al.* (2022) 'Systematic Literature Review on Security Risks and its Practices in Secure Software Development', *IEEE Access*, 10, pp. 5456–5481. Available at: https://doi.org/10.1109/ACCESS.2022.3140181.

Kudriavtseva, A. and Gadyatskaya, O. (2023) 'Secure Software Development Methodologies: A Multivocal Literature Review'. arXiv. Available at: http://arxiv.org/abs/2211.16987 (Accessed: 18 May 2024).

Microsoft Security Development Lifecycle Practices (no date). Available at: https://www.microsoft.com/en-us/securityengineering/sdl/practices (Accessed: 11 March 2024).

Oracle (2024) *Software Security Assurance | Oracle*. Available at: https://www.oracle.com/corporate/security-practices/assurance/ (Accessed: 19 March 2024).

OWASP (2024) *SAMM, The Model*, *SAMM, The Model*. Available at: https://owaspsamm.org/model// (Accessed: 19 March 2024).

OWASP SAMM (2024) *OWASP SAMM*. Available at: // (Accessed: 19 March 2024).

Pressman, R.S. (2010) *Ingenieria del Software. Un Enfoque Practico*.

Robayo, B, E.C. (2020) 'GUÍA DE PRINCIPIOS Y BUENAS PRÁCTICAS PARA PRUEBAS DE SEGURIDAD DE SOFTWARE EN APLICACIONES WEB PARA UNA EMPRESA DEL SECTOR PRIVADO'.

Sancho Núñez, J.C., Caro, A. and Rodriguez, P. (2020) 'A Preventive Secure Software Development Model for a Software Factory: A Case Study', *IEEE Access*, PP, pp. 1–1. Available at: https://doi.org/10.1109/ACCESS.2020.2989113.

Schubert, M., Pagel, S. and Von Korflesch, H. (2023) 'Success Factors in Secure Software Development of Cloud Applications in Germany: A Qualitative-explorative Expert Study', in. *Hawaii International Conference on System Sciences*. Available at: https://doi.org/10.24251/HICSS.2023.805.

Sommerville, I. (2011) *Software engineering*. 9th ed. Boston: Pearson.

Souppaya, M., Scarfone, K. and Dodson, D. (2022) *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. NIST Special Publication (SP) 800-218. National Institute

of Standards and Technology. Available at: https://doi.org/10.6028/NIST.SP.800-218.

Synopsys (no date) *BSIMM14*, *Synopsys*. Available at: https://www.synopsys.com/software-integrity/engage/bsimm-web/bsimm-report (Accessed: 19 March 2024).

Team, C. 4 A. (2023) *NIST y el desarrollo seguro de software*, *Tarlogic Security*. Available at: https://www.tarlogic.com/es/blog/nist-desarrollo-seguro-de-software/ (Accessed: 30 September 2023).

Tung, Y.-H. *et al.* (2016) 'An integrated security testing framework for Secure Software Development Life Cycle', in, pp. 1–4. Available at: https://doi.org/10.1109/APNOMS.2016.7737238.

Valdés, Y. *et al.* (2023) 'Towards the Integration of Security Practices in Agile Software Development: A Systematic Mapping Review', *Applied Sciences*, 13(7), p. 4578. Available at: https://doi.org/10.3390/app13074578.