centro criptológico nacional

Guía de Seguridad de las TIC
CCN-STIC 482

Ciclo de Vida de Desarrollo de Software Seguro (SSDLC)



Noviembre 2026

MINISTERIO
DE DEFENSA

cpage.mpr.gob.es

Catálogo de Publicaciones de la Administración General del Estado
cpage.mpr.gob.es

Edita:

MINISTERIO DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA

CCN
centro criptológico nacional

Pº de la Castellana 109, 28046 Madrid
© Centro Criptológico Nacional, 2026
NIPO (edición en línea)

CCN
c=ES, l=MADRID, o=CCN,
ou=SELLO ELECTRONICO,
2.5.4.97=VATES-S2800155J,
serialNumber=S2800155J,
cn=CCN
2026.11.23 11:35:53 +01:00

LIMITACIÓN DE RESPONSABILIDAD

El presente documento se proporciona de acuerdo con los términos en él recogidos, rechazando expresamente cualquier tipo de garantía implícita que se pueda encontrar relacionada. En ningún caso, el Centro Criptológico Nacional puede ser considerado responsable del daño directo, indirecto, fortuito o extraordinario derivado de la utilización de la información y software que se indican incluso cuando se advierta de tal posibilidad.

AVISO LEGAL

Quedan rigurosamente prohibidas, sin la autorización escrita del Centro Criptológico Nacional, bajo las sanciones establecidas en las leyes, la reproducción parcial o total de este documento por cualquier medio o procedimiento, comprendidos la reprografía y el tratamiento informático, y la distribución de ejemplares del mismo mediante alquiler o préstamo públicos.

Esta publicación se distribuye bajo licencia CC BY-NC-ND 4.0 que permite compartir el material en cualquier medio o formato con la condición de reconocer adecuadamente la procedencia «Edita: Ministerio de Defensa. Secretaría General Técnica.» No se puede modificar. No se puede utilizar con fines comerciales.

ÍNDICE

1. INTRODUCCIÓN 1
1.1 OBJETIVO 2
1.2 ALCANCE 2
2. EL CICLO DE VIDA DEL DESARROLLO DE SOFTWARE SEGURO (SSDLC) 4
2.1 ¿QUÉ ES EL SSDLC? 4
2.2 MODELOS DE CICLO DE VIDA DEL DESARROLLO DE SOFTWARE (SDLC) Y SU ADAPTACIÓN AL SSDLC 4
2.2.1 MODELO EN CASCADA (WATERFALL) 5
2.2.2 MODELO EN ESPIRAL 6
2.2.3 METODOLOGÍAS ÁGILES 7
2.3 DIFERENCIAS FUNDAMENTALES ENTRE SDLC Y SSDLC 8
2.4 EL PRINCIPIO FUNDAMENTAL DEL "SHIFT LEFT" 9
2.5 CARACTERÍSTICAS DEL SSDLC 10
3. SSDLC EN METODOLOGÍAS TRADICIONALES 11
3.1 FASE 1: PLANIFICACIÓN Y ANÁLISIS DE REQUISITOS 11
3.2 FASE 2: DISEÑO Y ARQUITECTURA 12
3.3 FASE 3: DESARROLLO E IMPLEMENTACIÓN 12
3.4 FASE 4: PRUEBAS Y VERIFICACIÓN 13
3.5 FASE 5: DESPLIEGUE E IMPLEMENTACIÓN 13
3.6 FASE 6: OPERACIÓN Y MANTENIMIENTO 14
4. SSDLC EN METODOLOGÍAS ÁGILES 15
4.1 CONCEPTO DE LAS METODOLOGIAS ÁGILES 15
4.2 CARACTERÍSTICAS FUNDAMENTALES EN EL CONTEXTO DEL SSDLC 16
4.3 PRINCIPALES MODELOS ÁGILES Y SU IMPLICACIÓN CON EL SSDLC 16
4.3.1 SCRUM 16
4.3.2 KANBAN 18
4.3.3 EXTREME PROGRAMMING (XP) 18
4.3.4 LEAN DEVELOPMENT 18
4.4 INTEGRACIÓN DEL SSDLC EN EL CICLO ÁGIL 19
5. MARCOS DE REFERENCIA PARA LA IMPLEMENTACION DEL SSDLC 20
5.1 OWASP SAMM V2 21
5.1.1 GOBERNANZA 22
5.1.2 DISEÑO 23
5.1.3 IMPLEMENTACIÓN 25
5.1.4 VERIFICACIÓN 26
5.1.5 OPERACIONES 27
5.2 NIST SECURE SOFTWARE DEVELOPMENT FRAMEWORK (SSDF) 29
5.2.1 INTRODUCCIÓN AL MARCO NIST SSDF 29
5.2.2 AUDIENCIA Y APLICABILIDAD DEL SSDF 30
5.2.3 PRINCIPIOS FUNDAMENTALES DEL SSDF 31

5.2.4 ESTRUCTURA DEL SSDF: LAS CUATRO CATEGORÍAS DE PRACTICAS 32
5.2.5 IMPLEMENTACIÓN DEL MARCO SSDF: CONSIDERACIONES PRÁCTICAS 39
5.2.6 RELACIÓN DEL SSDF CON EL ESQUEMA NACIONAL DE SEGURIDAD 40
5.2.7 BENEFICIOS DE ADOPTAR EL NIST SSDF 40
5.3 OTROS FRAMEWORKS RELEVANTES 41
5.3.1 BUILDING SECURITY IN MATURITY MODEL (BSIMM) 41
5.3.2 MICROSOFT SECURITY DEVELOPMENT LIFECYCLE (SDL) 42
5.3.3 OWASP SECURE CODING PRACTICES 42
5.4 COMPARATIVA DE FRAMEWORKS Y MODELOS DE SEGURIDAD EN EL DESARROLLO DE SOFTWARE 43
5.4.1 TABLA COMPARATIVA DE FRAMEWORKS 43
6. PRUEBAS DE SEGURIDAD Y DESARROLLO SEGURO 45
6.1 HERRAMIENTAS DE PRUEBAS DE SEGURIDAD EN APLICACIONES (AST) 45
6.1.1 PRUEBAS ESTÁTICAS DE SEGURIDAD DE APLICACIONES (SAST) 45
6.1.2 ANÁLISIS DE LA COMPOSICIÓN DEL SOFTWARE (SCA) 47
6.1.3 PRUEBAS DINÁMICAS DE SEGURIDAD DE APLICACIONES (DAST) 48
6.1.4 PRUEBAS INTERACTIVAS DE SEGURIDAD DE APLICACIONES (IAST) 49
6.1.5 AUTOPROTECCIÓN DE APLICACIONES EN TIEMPO DE EJECUCIÓN (RASP) 50
6.1.6 PRUEBAS DE SEGURIDAD DE APLICACIONES MÓVILES (MAST) 51
6.2 PRÁCTICAS FUNDAMENTALES DE DESARROLLO SEGURO 52
6.2.1 PRINCIPIOS DE DESARROLLO SEGURO 52
6.2.2 RECURSOS Y ESTÁNDARES DE DESARROLLO SEGURO 53
7. INTEGRACIÓN DE DEVSECOPS EN EL SSDLC 55
7.1 INTRODUCCIÓN A DEVSECOPS EN EL SSDLC 55
7.1.1 DEVOPS 55
7.1.2 DEVSECOPS 56
7.1.3 CI/CD COMO HABILITADOR DE DEVSECOPS 57
7.1.4 INTEGRACIÓN CONTINUA (CI) 59
7.1.5 ENTREGA CONTINUA (CD) 59
7.1.6 DESPLIEGUE CONTINUO (CONTINUOUS DEPLOYMENT) 60
7.1.7 CI/CD Y DEVSECOPS 60
7.1.8 SEGURIDAD DE LA INFRAESTRUCTURA CI/CD 61
7.2 DEVSECOPS EN SSDLC EN METODOLOGÍAS TRADICIONALES 62
7.3 INTEGRACIÓN DE DEVSECOPS EN METODOLOGÍAS ÁGILES 62
7.4 AUTOMATIZACIÓN EN DEVSECOPS 63
7.4.1 FUNDAMENTOS DE LA AUTOMATIZACIÓN EN ENTORNOS ÁGILES 63
7.4.2 PIPELINE DE SEGURIDAD AUTOMATIZADO 64
7.4.3 AUTOMATIZACIÓN DE RESPUESTA A INCIDENTES 64
7.4.4 BENEFICIOS CUANTIFICABLES DE LA AUTOMATIZACIÓN 65
7.5 CULTURA Y CAPACITACIÓN 65
7.6 DESAFÍOS Y MITIGACIÓN 66
7.7 BENEFICIOS DE LA INTEGRACIÓN 66
7.8 INTEGRACIÓN DE DEVSECOPS EN OTRAS METODOLOGÍAS 67
7.8.1 SELECCIÓN DE METODOLOGÍA Y ADAPTACIÓN DEL SSDLC 68
8. MÉTRICAS DEL CICLO DE VIDA DEL SSDLC 69

9. ABREVIACIONES 71
ANEXO A. FUENTES CONSULTADAS 72

1. INTRODUCCIÓN

El desarrollo de software se ha consolidado como una actividad crítica en prácticamente todos los sectores de la sociedad moderna, constituyendo el núcleo de los sistemas que sustentan desde servicios públicos esenciales hasta infraestructuras críticas nacionales. Sin embargo, la creciente sofisticación de las amenazas cibernéticas y la evolución del panorama de ciberataques han evidenciado que el desarrollo tradicional de software, enfocado principalmente en funcionalidad y rendimiento, resulta insuficiente para garantizar la seguridad y resiliencia de los sistemas digitales contemporáneos.

El ciclo de vida del desarrollo de software **(*Software Development Lifecycle* o SDLC)** constituye una metodología formal para diseñar, crear y mantener software, proporcionando un proceso estructurado compuesto por fases bien definidas que garantizan la entrega de productos de calidad, alineados con las necesidades del negocio y desarrollados de manera eficiente y controlada. Tradicionalmente, el SDLC incluye fases de planificación, análisis de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento. Sin embargo, este enfoque convencional no integra de forma sistemática consideraciones de seguridad en todas sus etapas, lo que frecuentemente resulta en la identificación tardía de vulnerabilidades, de tal forma que su corrección implica costos significativamente elevados y puede comprometer tanto la seguridad como la continuidad operativa de los sistemas.

Por su parte, el **Ciclo de Vida del Desarrollo de Software Seguro (*Secure Software Development Lifecycle* o SSDLC)** representa una evolución fundamental del SDLC tradicional, integrando la seguridad como un componente esencial y transversal en cada etapa del proceso de desarrollo. Este modelo incorpora prácticas, herramientas y controles de seguridad a lo largo de todo el SDLC, con el objetivo de identificar y mitigar vulnerabilidades de manera proactiva, desde las fases iniciales de concepción y planificación de la arquitectura de software hasta la operación continuada y el mantenimiento del desarrollo en producción. El SSDLC se fundamenta en el principio de ***Shift Left*** o desplazamiento a la izquierda, que promueve la incorporación de prácticas de seguridad desde las primeras fases del proyecto, permitiendo detectar y corregir vulnerabilidades, logrando así un costo de remediación significativamente menor y un impacto en los plazos de entrega mínimo.

El presente documento proporciona una guía de referencia para la implementación efectiva del SSDLC en organizaciones del sector público y privado, detallando las seis fases fundamentales del ciclo de vida (planificación y análisis de requisitos, diseño y arquitectura, desarrollo, pruebas y verificación, despliegue, y operación y mantenimiento) junto con las prácticas de seguridad específicas aplicables en cada etapa. Para ello se desarrollarán los fundamentos del SSDLC, la aplicación práctica de los marcos de referencia internacionales, y su adaptación a diferentes contextos metodológicos (desde modelos tradicionales en cascada hasta enfoques ágiles).

1.1 OBJETIVO

La presente guía tiene como objetivo principal proporcionar a las organizaciones del sector público y privado un marco de referencia integral para la implementación efectiva de la seguridad sobre el Ciclo de Vida del Desarrollo de Software. Este documento busca establecer las bases conceptuales, metodológicas y prácticas necesarias para integrar la seguridad como un componente fundamental en todas las fases del desarrollo de software, desde la concepción inicial hasta la operación y el mantenimiento del producto final, promoviendo un enfoque proactivo que anticipe y mitigue vulnerabilidades antes de que puedan ser explotadas.

La guía pretende servir como instrumento de apoyo para equipos de desarrollo, responsables de seguridad, arquitectos de software, gestores de proyectos y técnicos de seguridad en aplicaciones, facilitando la adopción de prácticas de seguridad que permitan reducir significativamente las vulnerabilidades en el software desarrollado o adquirido. Asimismo, se orienta a promover una cultura organizacional donde la seguridad sea una responsabilidad compartida por todos los actores involucrados en el ciclo de vida del software, fomentando así la creación de productos digitales más resilientes, confiables y alineados con los requisitos normativos aplicables.

Finalmente, este documento aspira a constituirse en una herramienta práctica que facilite el cumplimiento de las obligaciones establecidas en el Esquema Nacional de Seguridad (ENS) y otras normativas relevantes, proporcionando orientación clara sobre la aplicación de marcos de referencia internacionales como el NIST Secure Software Development *Framework* (SSDF), el OWASP Software Assurance Maturity Model (SAMM v2), entre otros recursos especializados que complementan y enriquecen las capacidades de desarrollo seguro de las organizaciones.

1.2 ALCANCE

El alcance de esta guía CCN-STIC abarca la descripción integral del Ciclo de Vida del Desarrollo de Software Seguro (SSDLC), aplicable a cualquier tipo de software, independientemente de la tecnología, plataforma, lenguaje de programación o modelo de desarrollo empleado (desde metodologías tradicionales hasta más actuales y ágiles). La guía está diseñada para ser adoptada por organizaciones de cualquier sector o tamaño, permitiendo su implementación gradual según el nivel de madurez y digitalización, perfil de riesgo y recursos disponibles de cada organización.

El documento desarrolla en profundidad cada una de las seis fases del SSDLC (planificación y análisis de requisitos, diseño y arquitectura, desarrollo, pruebas y verificación, despliegue, y operación y mantenimiento), describiendo las prácticas de seguridad específicas aplicables en cada etapa. Asimismo, presenta un análisis comparativo de los principales marcos de referencia internacionales (NIST SSDF, OWASP SAMM v2, Microsoft SDL, BSIMM y OWASP *Secure Coding Practices*), proporcionando criterios para su selección e implementación según las necesidades específicas de cada organización y proyecto.

Finalmente, la guía incluye una sección específica dedicada a la integración del SSDLC en metodologías tradicionales y ágiles, enfocados al modelo DevSecOps, explicando cómo adaptar las prácticas de seguridad a entornos de desarrollo iterativo e incremental con entrega continua, y proporcionando orientación sobre la automatización de controles de seguridad en pipelines de CI/CD.

2. EL CICLO DE VIDA DEL DESARROLLO DE SOFTWARE SEGURO (SSDLC)

2.1 ¿QUÉ ES EL SSDLC?

El **Ciclo de Vida del Desarrollo de Software (SDLC)** es una metodología formal para diseñar, crear y mantener software de forma eficiente y con el mayor nivel de calidad posible [5]. Este modelo fue diseñado para reducir los tiempos de desarrollo y despliegue de software, disminuyendo costes e incrementando la calidad de software.

Por su parte, el **Ciclo de Vida del Desarrollo de Software Seguro (SSDLC)** representa una evolución del SDLC tradicional que integra la seguridad como un componente fundamental en cada etapa del proceso de desarrollo. Pocos modelos SDLC tradicionales abordaban explícitamente la seguridad del software en detalle, por lo que, debido al incremento de ciberataques y la necesidad imperiosa de asegurar el software, las empresas han tenido que implantar prácticas de desarrollo seguro en las fases de desarrollo de estos.

Independientemente del modelo SDLC que se utilice, las prácticas de desarrollo seguro de software deben integrarse a lo largo de todo el ciclo por cuatro razones principales:

- Reducir el número de vulnerabilidades en el software desplegado en producción.
- Reducir el riesgo de la explotación de vulnerabilidades no detectadas o no abordadas.
- Disminuir costes en la remediación de fallos de seguridad introducidos en las distintas fases del ciclo de vida.
- Cumplir normativas legales

Este enfoque sistemático tiene como objetivo identificar y mitigar vulnerabilidades de forma proactiva, desde un mal diseño de software seguro, fallos de seguridad en el código desarrollado o configuraciones inseguras, reduciendo significativamente los riesgos de seguridad antes de que el software llegue a los entornos de producción. El resultado de usar este modelo es obtener un software más resiliente, que no solo cumple con los requisitos funcionales, sino que también está diseñado para resistir amenazas y proteger sus servicios.

2.2 MODELOS DE CICLO DE VIDA DEL DESARROLLO DE SOFTWARE (SDLC) Y SU ADAPTACIÓN AL SSDLC

Los modelos descritos en este apartado son modelos de ciclo de vida del desarrollo de software (SDLC). Su relevancia en esta guía radica en que el SSDLC no sustituye necesariamente dichos modelos, sino que incorpora prácticas de seguridad en cada uno de ellos, adaptándose a sus fases, cadencias y artefactos. Existen muchos modelos de SDLC, aunque pocos modelos SDLC abordan explícitamente la seguridad del software en detalle, por lo que las prácticas de desarrollo seguro deben añadirse e integrarse en cada modelo para conseguir crear un modelo SSDLC maduro. Independientemente del

modelo SDLC que se utilice, las prácticas de desarrollo seguro de software deben integrarse a lo largo de todo el ciclo.

2.2.1 MODELO EN CASCADA (WATERFALL)

El modelo en cascada es un enfoque secuencial y lineal donde cada fase del desarrollo debe completarse antes de comenzar la siguiente. Originalmente fue desarrollado en 1970, y su objetivo es promover una progresión lógica de actividades que fluyen de manera unidireccional a través de las fases de análisis de requisitos, diseño, implementación, verificación, despliegue y mantenimiento.



Figura 1. SDLC – Modelo en Cascada

a) Ventajas del modelo en cascada:

- Estructura simple y fácil de comprender
- Hitos y entregables claramente definidos
- Facilita el control de gestión y seguimiento
- Documentación exhaustiva y trazabilidad completa
- Permite estimación precisa de costes y plazos

b) Inconvenientes del modelo en cascada:

- Rigidez ante cambios en requisitos
- Detección tardía de errores
- Cliente no ve el producto hasta fases avanzadas
- Dificultad para integrar seguridad de forma continua

c) Aplicabilidad:

- El modelo en cascada es más apropiado para sistemas y proyectos que tienen requisitos bien definidos, estables y completamente comprendidos desde el principio, así como para proyectos con tecnologías maduras y bien conocidas.

2.2.2 MODELO EN ESPIRAL

El modelo en espiral combina elementos del modelo en cascada con desarrollo iterativo, añadiendo un énfasis explícito y continuo en el análisis de riesgos. Cada iteración del ciclo (representada como una espiral) pasa por cuatro fases principales: determinación de objetivos y alternativas, análisis y evaluación de riesgos, desarrollo y pruebas del producto, y planificación de la siguiente iteración.

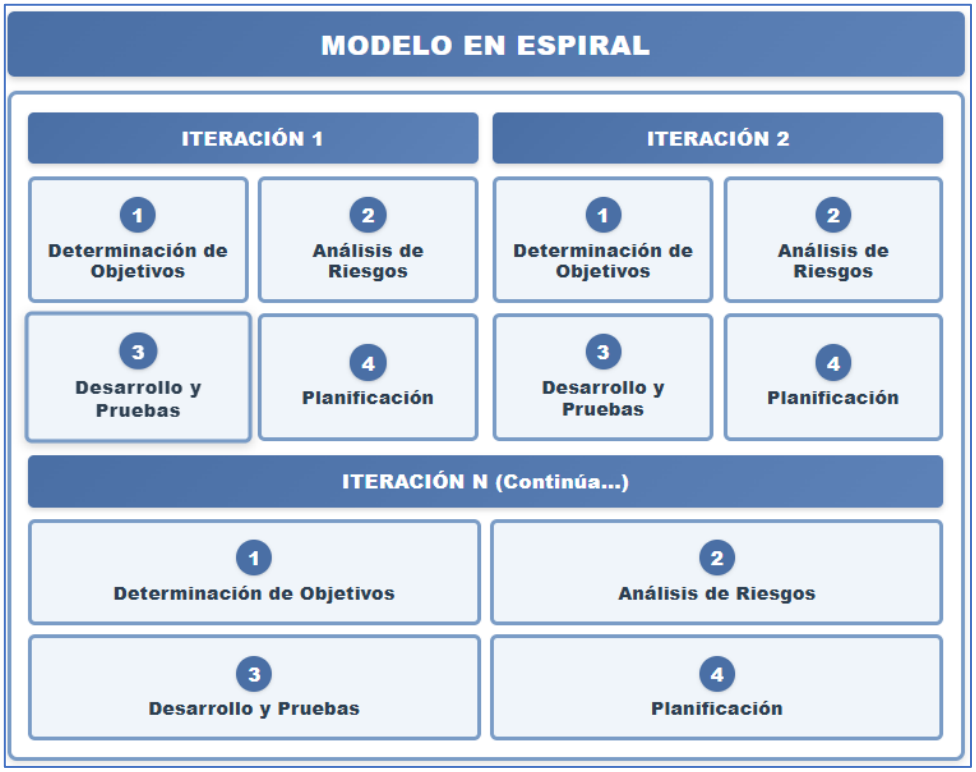


Figura 2. SDLC – Modelo en Espiral

a) Ventajas del modelo en espiral:

- Énfasis continuo en análisis y gestión de riesgos
- Flexibilidad para incorporar cambios entre iteraciones
- Entrega incremental con validación temprana
- Adecuado para proyectos grandes y complejos
- Refinamiento progresivo de requisitos y diseño

b) Inconvenientes del modelo en espiral:

- Alta complejidad en gestión y control
- Requiere experiencia significativa en análisis de riesgos
- Puede resultar costoso cuando se crean múltiples iteraciones
- Difícil estimar tiempo y coste total
- Requiere participación continua del cliente
- No óptimo para proyectos pequeños o de bajo riesgo

c) Aplicabilidad

- El modelo en espiral es más apropiado para proyectos grandes y complejos donde la gestión de riesgos es crítica, proyectos con requisitos que pueden evolucionar, y desarrollos donde los costes de fracaso serían muy elevados.

2.2.3 METODOLOGÍAS ÁGILES

Las metodologías ágiles representan un conjunto de enfoques iterativos e incrementales que enfatizan la flexibilidad, la colaboración continua con el cliente y la entrega frecuente de software funcional en incrementos pequeños [6]. El modelo ágil organiza las fases del SDLC en varios ciclos de desarrollo cortos, donde el equipo itera rápidamente a través de las fases, entregando cambios incrementales de software en cada ciclo. Entre las metodologías ágiles más conocidas se encuentran Scrum, Kanban, *Extreme Programming* (XP) y *Lean Development*.

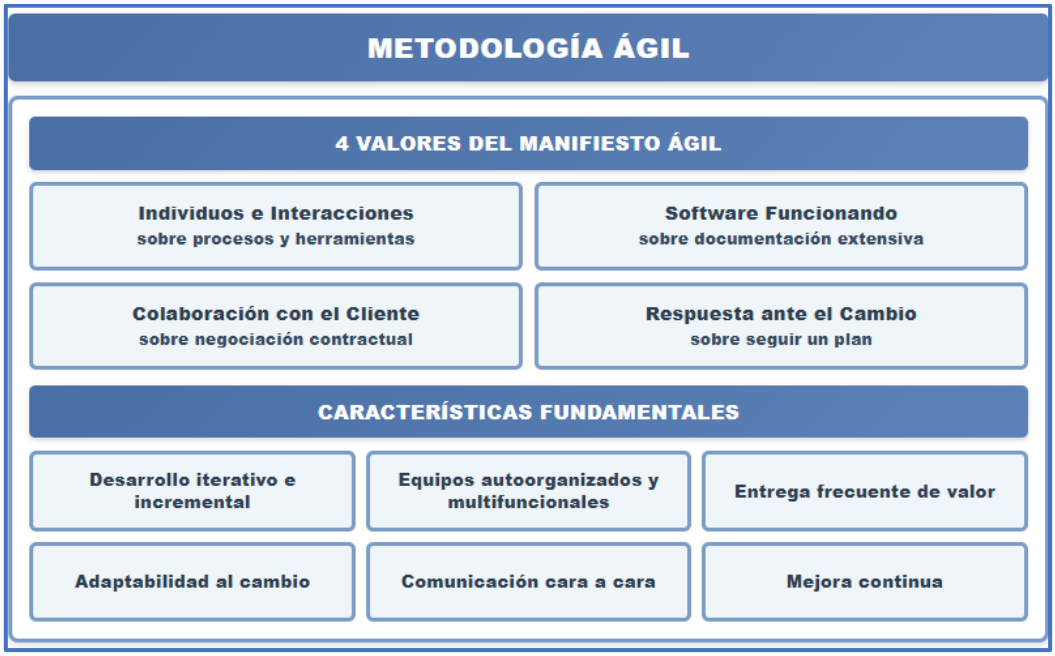


Figura 3. SDLC – Metodología Ágil

a) **Ventajas de las metodologías ágiles:**

- Alta adaptabilidad ante cambios en requisitos
- Entrega continua de valor en iteraciones cortas
- Colaboración estrecha entre equipos y *stakeholders*
- Detección y corrección temprana de errores
- Mayor satisfacción del cliente con entregas frecuentes
- Fomenta mejora continua mediante retrospectivas

b) **Inconvenientes de las metodologías ágiles:**

- Requiere alta disponibilidad del cliente
- Difícil estimar costes y plazos totales
- Puede generar documentación insuficiente
- Necesita equipos experimentados y autodisciplinados
- Difícil de escalar a proyectos muy grandes
- Requiere cambio cultural significativo

c) **Aplicabilidad:**

- Las metodologías ágiles son más apropiadas para proyectos con requisitos cambiantes o no completamente definidos al inicio, equipos pequeños o medianos con alta comunicación, y organizaciones que valoran la entrega rápida de valor y la adaptabilidad sobre la previsibilidad total.

2.3 DIFERENCIAS FUNDAMENTALES ENTRE SDLC Y SSDLC

Para comprender el valor y la necesidad del SSDLC es fundamental identificar las diferencias clave con el enfoque tradicional del SDLC. Mientras que ambos buscan producir software de calidad, el SSDLC transforma radicalmente el momento, la forma y la responsabilidad con la que se aborda la seguridad a lo largo del ciclo de vida del desarrollo.

- **Enfoque tradicional del SDLC:** En los modelos SDLC tradicionales, la seguridad típicamente se aborda de manera:
  - **Reactiva:** La seguridad se considera después de que el desarrollo ha avanzado significativamente o incluso después del despliegue.
  - **Aislada:** Tratada como una fase independiente, generalmente al final del ciclo, en lugar de como una práctica continua.
  - **Limitada:** Enfocada principalmente en pruebas de seguridad durante la fase de verificación.

- **Costosa de remediar:** Los problemas de seguridad detectados en fases tardías requieren retrabajo extensivo y cambios costosos en la arquitectura.

- **Enfoque del SSDLC:** El SSDLC transforma este paradigma mediante:
  - ✓ **Integración continua de la seguridad:** La seguridad no es un hito aislado, sino una práctica continua que acompaña al desarrollo en cada iteración.
  - ✓ **Proactividad:** Identificación y mitigación de vulnerabilidades antes de que el software llegue a producción.
  - ✓ **Responsabilidad compartida:** Todos los miembros del equipo de desarrollo comparten la responsabilidad de la seguridad, no solo el equipo de seguridad.
  - ✓ **Eficiencia económica:** La detección temprana de problemas reduce drásticamente los costos de corrección.

2.4 EL PRINCIPIO FUNDAMENTAL DEL "SHIFT LEFT"

La mayoría de los aspectos de seguridad pueden abordarse múltiples veces dentro de un SDLC, pero en general el mejor momento para hacerlo es en la etapa más temprana. Aplicando esta filosofía se logra un menor esfuerzo y costo para lograr el mismo nivel de seguridad. Este principio es conocido como **"desplazamiento a la izquierda" (*shifting left*)**, y su objetivo es minimizar cualquier deuda técnica que requeriría remediar fallos de seguridad después de que el software esté en producción, resultando en un software con mayor seguridad y resiliencia antes de ser publicado.

Al adoptar este enfoque, las organizaciones pueden:

- ✓ **Detectar y remediar problemas de seguridad cuando son más económicos de resolver:** Los costos de corrección aumentan exponencialmente con cada fase del desarrollo.
- ✓ **Reducir el coste y el tiempo asociados a la corrección de vulnerabilidades:** La remediación temprana evita el retrabajo extensivo y complejo.
- ✓ **Evitar la modificación de arquitecturas inseguras en fases avanzadas:** Las decisiones de diseño inseguras son muy costosas de corregir posteriormente.
- ✓ **Fomentar una cultura de seguridad compartida entre todos los miembros del equipo:** La seguridad se convierte en responsabilidad de todos según la fase del ciclo de vida en el que se sitúe.

2.5 CARACTERÍSTICAS DEL SSDLC

El SSDLC se caracteriza por los siguientes elementos fundamentales:

- **Integración continua de la seguridad:** La seguridad no se trata como un hito aislado, sino como una práctica continua que acompaña al desarrollo en cada iteración, incluyendo análisis de riesgos, modelado de amenazas, desarrollo seguro, pruebas de seguridad continuas o monitorización, entre otros.
- **Enfoque basado en riesgos:** El SSDLC reconoce que cada organización y proyecto tiene objetivos, supuestos, necesidades y contextos operativos únicos. Factores como el riesgo, el coste, la viabilidad y la aplicabilidad deben considerarse al decidir qué prácticas implementar y qué recursos dedicar a cada una.
- **Automatización y herramientas:** La automatización es esencial para implementar prácticas de seguridad a escala, permitiendo reducir el esfuerzo humano, mejorar la precisión y reproducibilidad, generar documentación y evidencias, y facilitar la integración de la seguridad en el flujo de trabajo de desarrollo.
- **Aplicabilidad universal:** Las prácticas son utilizables por organizaciones de cualquier sector, tamaño o nivel de sofisticación en ciberseguridad, siendo aplicables a cualquier tipo de software (TI, ICS, CPS, IoT…) y compatibles con cualquier tecnología, plataforma, lenguaje de programación, modelo SDLC o conjunto de herramientas existente.

3. SSDLC EN METODOLOGÍAS TRADICIONALES

El SSDLC integra prácticas de seguridad específicas en cada fase del ciclo de vida del desarrollo. En líneas generales, en la siguiente figura se muestran las 6 fases de los ciclos de vida de software más comunes y habituales en la mayoría de las metodologías de desarrollo.



Figura 4. Fases del Ciclo de Vida del Software

Dentro de las distintas fases del desarrollo, tal y como se expresaba en las secciones anteriores, podemos integrar distintas prácticas de seguridad que ayuden a los equipos de desarrollo a detectar posibles fallos de seguridad de la forma más temprana y eficiente posible. Aunque algunos materiales formativos se centran principalmente en las fases previas al despliegue, esta guía adopta una visión de ciclo de vida completo e incluye la fase de operación y mantenimiento, por ser esencial para la monitorización, la gestión de vulnerabilidades, el parcheo y la respuesta a incidentes.

3.1 FASE 1: PLANIFICACIÓN Y ANÁLISIS DE REQUISITOS

Esta fase tiene como finalidad establecer los fundamentos de seguridad del proyecto antes de iniciar las actividades de diseño y desarrollo, mediante la planificación del proyecto, la identificación de requisitos tanto funcionales como de seguridad, así como la evaluación de riesgos potenciales y el establecimiento de criterios de cumplimiento normativo. Para alcanzar este objetivo, se deben aplicar las siguientes prácticas de seguridad:

a) **Definición de requisitos de seguridad:** Identificar requisitos de seguridad sobre la infraestructura, el futuro software implementado y las tecnologías usadas.

b) **Análisis de riesgos inicial:** Evaluar posibles riesgos y amenazas iniciales del proyecto.

c) **Requisitos de cumplimiento:** Establecer criterios de cumplimiento normativo aplicables (tales como ENS, RGPD, PCI-DSS…) en base a diferentes características del desarrollo, como la sensibilidad de datos almacenados y/o procesados, necesidad de gestionar pagos o aplicabilidad al sector público.

d) **Clasificación de información:** Determinar qué tipos de datos manejará el software y sus requisitos de protección.

3.2 FASE 2: DISEÑO Y ARQUITECTURA

El propósito de esta fase es diseñar una arquitectura de software que integre la seguridad desde su concepción, asegurando que sea resistente a amenazas conocidas y emergentes mediante el análisis sistemático de riesgos, la aplicación de patrones de diseño seguro y la incorporación de controles defensivos apropiados. Para alcanzar este objetivo, se deben aplicar las siguientes prácticas de seguridad:

a) **Modelado de amenazas:** Identificar vectores de ataque potenciales antes de ser desarrollados y establecer contramedidas apropiadas que mitiguen la amenaza.

b) **Arquitectura de seguridad:** Evaluar decisiones arquitectónicas desde una perspectiva de seguridad, tales como exposición a la red, conexión entre servicios y despliegue de infraestructura.

c) **Principios de diseño seguro o *Security By Design*:** Aplicar principios como defensa en profundidad, privilegio mínimo, segregación de funciones y otras prácticas que permitan cumplir los requisitos de seguridad previamente definidos.

3.3 FASE 3: DESARROLLO E IMPLEMENTACIÓN

Esta fase tiene como finalidad asegurar que el código desarrollado sea inherentemente seguro y resistente a vulnerabilidades conocidas, mediante la implementación de prácticas de desarrollo seguro, el análisis de seguridad a través de herramientas AST, la gestión segura de componentes y secretos, y las revisiones sistemáticas de código. Las prácticas de seguridad aplicables en esta fase incluyen:

a) **Desarrollo seguro:** Aplicar prácticas de implementación segura (validación de entradas, manejo seguro de errores, implementación de una autenticación y autorización fuerte…).

b) **Análisis estático de código (SAST):** Utilizar herramientas automáticas sobre el código estático (sin haber sido desplegado) para detectar vulnerabilidades durante el desarrollo de este.

c) **Análisis de composición de software (SCA):** Verificar y gestionar componentes de terceros, detectando vulnerabilidades conocidas o uso de software con licencia incompatible.

d) **Detección de secretos en código:** Detección de credenciales, claves, tokens y otros secretos o información personal almacenados directamente en el código fuente.

e) **Revisiones de código de seguridad:** Realizar “peer reviews” enfocadas en aspectos de seguridad.

3.4 FASE 4: PRUEBAS Y VERIFICACIÓN

El propósito de esta fase es verificar la seguridad del software mediante pruebas dinámicas, pruebas de penetración, *fuzzing* y validación funcional de controles de seguridad, con el objetivo de identificar y remediar vulnerabilidades antes del despliegue en entornos de producción. Las prácticas de seguridad aplicables en esta fase incluyen:

a) **Análisis dinámico (DAST):** Probar la aplicación en ejecución sobre entornos preproductivos a través de herramientas automatizadas para detectar vulnerabilidades no detectadas previamente.

b) **Pruebas de penetración:** Simular ataques reales imitando el comportamiento de un atacante para identificar debilidades en el software.

c) **Fuzzing:** Probar el manejo de entradas inesperadas o malformadas sobre el software desplegado.

d) **Pruebas funcionales de seguridad:** Verificar que las características de seguridad funcionan como se espera a través de pruebas automáticas.

3.5 FASE 5: DESPLIEGUE E IMPLEMENTACIÓN

En esta fase se busca garantizar que el software se despliegue de forma segura en los entornos de producción, manteniendo su integridad mediante configuraciones seguras por defecto, gestión adecuada de secretos y credenciales, verificación de integridad y control de cambios, así como la generación de SBOM (*Software Bill of Materials*). Las prácticas de seguridad aplicables en esta fase incluyen:

a) **Configuración segura por defecto:** Asegurar que las configuraciones predeterminadas sean seguras, evitando la aparición de credenciales frágiles o configuraciones inseguras habilitadas por defecto, entre otras.

b) **Gestión de secretos y credenciales:** Implementar mecanismos seguros para gestionar secretos en producción, a través de gestores de credenciales protegidos y robustos.

c) **Verificación de integridad:** Proporcionar mecanismos para que los usuarios verifiquen la autenticidad del software (tales como firmas digitales o hashes criptográficos).

d) **Control de cambios:** Documentar y aprobar todos los cambios antes del despliegue a través de políticas de seguridad estrictas.

e) **Generación de SBOM:** Crear y distribuir los artefactos SBOM para mantener un inventario de los componentes de software usados por los distintos desarrollos. Esta práctica es especialmente relevante para la gestión inmediata de vulnerabilidades *0-day* o de día 0.

3.6 FASE 6: OPERACIÓN Y MANTENIMIENTO

Esta última fase busca mantener y mejorar continuamente la seguridad del software a lo largo de su vida útil mediante la monitorización permanente de los entornos de producción, la gestión proactiva de vulnerabilidades, la aplicación oportuna de parches de seguridad y la respuesta eficaz ante incidentes de seguridad, garantizando así la protección continua del software desplegado. Las prácticas de seguridad aplicables en esta fase incluyen:

a) **Monitorización continua:** Utilizar herramientas como CSPM (*Cloud Security Posture Management*) para detectar configuraciones incorrectas y actividades sospechosas en los entornos finales.

b) **Gestión de vulnerabilidades:** Identificar, evaluar y remediar vulnerabilidades de forma continua.

c) **Gestión de parches:** Mantener el software y sus dependencias actualizadas, así como la infraestructura donde ha sido desplegada.

d) **Respuesta a incidentes:** Tener procedimientos establecidos para responder a incidentes de seguridad, así como integraciones con los SIEM que recojan las alertas de seguridad.

e) **Divulgación responsable:** Gestionar la comunicación de vulnerabilidades con usuarios y clientes.

4. SSDLC EN METODOLOGÍAS ÁGILES

La implementación del Ciclo de Vida del Desarrollo de Software Seguro (SSDLC) en entornos ágiles requiere una adaptación específica de las prácticas de seguridad para alinearse con los valores de flexibilidad, iteración rápida y entrega continua. Este capítulo explora cómo los principios del SSDLC se integran en metodologías ágiles, culminando en la evolución hacia DevSecOps.

4.1 CONCEPTO DE LAS METODOLOGIAS ÁGILES

Las metodologías ágiles constituyen un conjunto de enfoques de desarrollo de software que priorizan la adaptabilidad, la colaboración continua con el cliente y la entrega incremental de software funcional en ciclos cortos. Este paradigma surgió formalmente en 2001 con el Manifiesto Ágil [6], que establece cuatro valores fundamentales:

- **Individuos e interacciones sobre procesos y herramientas**: El factor humano y la comunicación efectiva son más determinantes que la adherencia estricta a procesos formalizados.
- **Software funcionando sobre documentación extensiva**: La prioridad es producir software que funcione y entregue valor real al usuario.
- **Colaboración con el cliente sobre negociación contractual**: Se favorece el trabajo conjunto continuo permitiendo ajustes dinámicos basados en *feedback* real.
- **Respuesta ante el cambio sobre seguir un plan**: Se valora la capacidad de adaptación rápida por encima de la adherencia estricta a un plan inicial.

Estos valores se operacionalizan mediante doce principios básicos resumidos en la siguiente figura:



Figura 5. Principios de Metodologías Ágiles

4.2 CARACTERÍSTICAS FUNDAMENTALES EN EL CONTEXTO DEL SSDLC

- **Desarrollo iterativo e incremental**: El proceso se organiza en ciclos cortos de 1-4 semanas donde se completa planificación, diseño, desarrollo, pruebas e integración. Para el SSDLC, las prácticas de seguridad deben integrarse en cada iteración en lugar de constituir fases separadas.
- **Equipos autoorganizados y multifuncionales**: Los equipos poseen autonomía y cuentan con todas las habilidades necesarias. La responsabilidad de seguridad debe distribuirse dentro del equipo en lugar de depender exclusivamente de especialistas externos.
- **Entrega frecuente de valor**: Los equipos entregan software funcional en intervalos regulares. Las verificaciones de seguridad deben automatizarse para no convertirse en cuello de botella.
- **Adaptabilidad al cambio**: Los procesos ágiles acomodan requisitos cambiantes. El SSDLC debe ser flexible para adaptarse sin comprometer la postura de seguridad.

4.3 PRINCIPALES MODELOS ÁGILES Y SU IMPLICACIÓN CON EL SSDLC

4.3.1 SCRUM

Scrum es el marco de trabajo ágil más ampliamente adoptado, especialmente apropiado para el desarrollo de productos complejos en entornos donde los requisitos cambian rápidamente. Desarrollado por Ken Schwaber y Jeff Sutherland en los años 1990 [7], *Scrum* proporciona una estructura ligera pero suficiente para coordinar el trabajo de equipos de desarrollo. *Scrum* proporciona una estructura ligera que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos. El marco de trabajo *Scrum* es intencionalmente incompleto, definiendo solo las partes esenciales necesarias para implementar su teoría, y funciona bien como contenedor para otras técnicas, metodologías y prácticas.

Scrum se estructura en tres partes:

- **Estructura de roles:** *Scrum* define tres roles fundamentales:
  - El *Product Owner* actúa como la voz del cliente y es responsable de maximizar el valor del producto, manteniendo y priorizando el *Product Backlog*.
  - El *Scrum Master* es responsable de promover y apoyar *Scrum*, eliminando impedimentos y facilitando eventos.
  - El *Development Team* consiste en profesionales autoorganizados que realizan el trabajo de entregar incrementos de producto.
- **Eventos de Scrum:** El *Sprint* es el corazón de *Scrum*, un periodo de tiempo fijo de 1-4 semanas durante el cual se crea un incremento de producto "Terminado" usable. Además, aparecen ciertos eventos como:

- El *Sprint Planning*, que define qué se entregará y cómo se logrará.
- El *Daily Scrum*, que sincroniza actividades diariamente.
- La *Sprint Review*, que inspecciona el incremento con *stakeholders*.
- La *Sprint Retrospective*, que identifica mejoras al proceso.

- **Artefactos:**
  - *Product Backlog*, que es la lista ordenada de todo lo necesario en el producto;
  - *Sprint Backlog*, que contiene los elementos seleccionados para el Sprint más el plan de entrega;
  - Incremento, que es la suma de elementos completados que satisface el *Definition of Done*.

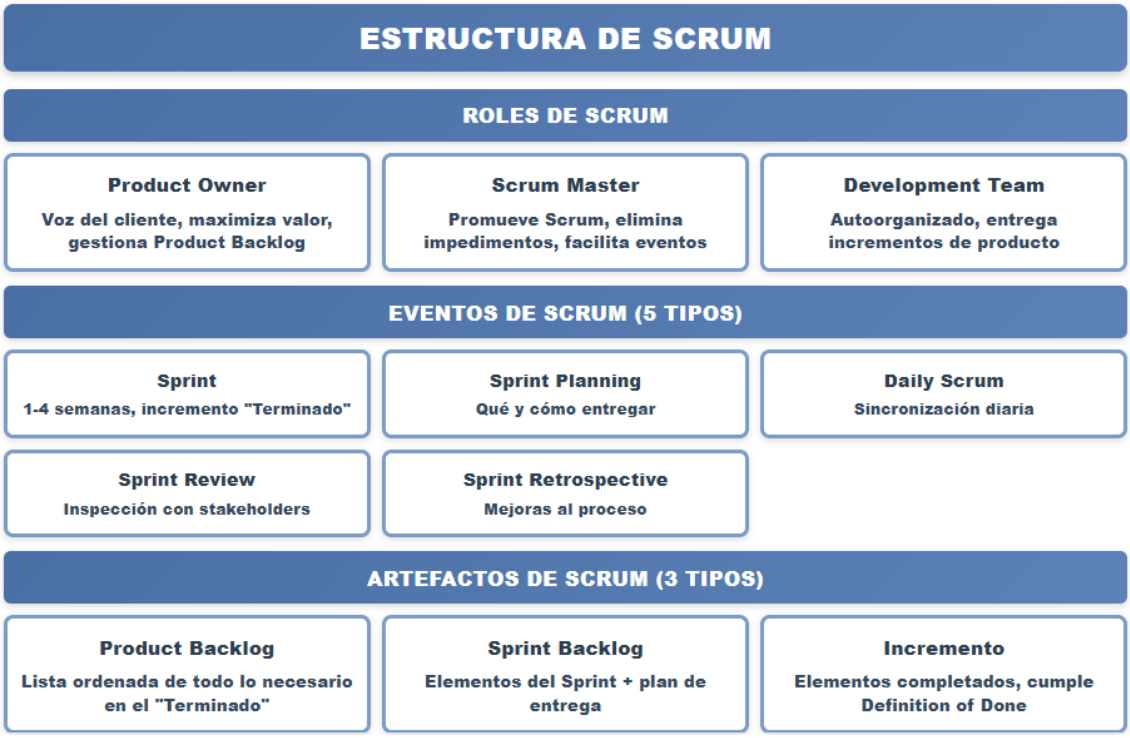


Figura 6. Estructura de *Scrum*

En cuanto a las **implicaciones para SSDLC** en *Scrum*, los requisitos de seguridad deben expresarse como historias de usuario en el *Product Backlog* y priorizarse junto con funcionalidades según su valor y riesgo. La *Definition of Done* debe incluir criterios de seguridad explícitos que no son negociables:

- ✓ Todas las pruebas de seguridad automatizadas son ejecutadas y se cumplen los *security gates*.
- ✓ Las dependencias están actualizadas sin vulnerabilidades conocidas.
- ✓ El código ha sido revisado por al menos otro desarrollador con conciencia de seguridad.

- ✓ Los secretos no están almacenados en código fuente.

Las prácticas de seguridad se integran naturalmente en cada *Sprint*:

- Durante el *Sprint Planning* se realiza modelado de amenazas incremental para las funcionalidades planificadas.
- Durante el desarrollo se ejecutan análisis SAST y SCA automáticos en cada *commit*.
- Durante el *testing* se realizan pruebas de seguridad automatizadas.
- En la *Sprint Review* se demuestra no solo funcionalidad sino también que los controles de seguridad operan correctamente.
- La *Sprint Retrospective* incluye revisión de la postura de seguridad del *Sprint*, identificando vulnerabilidades descubiertas, analizando causas raíz, y definiendo mejoras específicas a prácticas de seguridad para el siguiente *Sprint*.

4.3.2 KANBAN

Kanban se enfoca en visualizar el flujo de trabajo y optimizar la entrega mediante principios de cambio evolutivo y mejora continua. Sus prácticas centrales incluyen visualizar el flujo, limitar trabajo en progreso (WIP), gestionar el flujo, hacer políticas explícitas e implementar ciclos de *feedback*.

En cuanto a las **implicaciones para SSDLC**, las actividades de seguridad se visualizan como parte del flujo de trabajo en el tablero *Kanban*. Los límites WIP previenen sobrecarga que podría llevar a atajos de seguridad. Las políticas de seguridad se hacen explícitas definiendo cuándo los elementos pueden avanzar entre etapas. Las métricas (como *lead time* o *cycle time*) incluyen el tiempo de actividades de seguridad.

4.3.3 EXTREME PROGRAMMING (XP)

XP enfatiza la excelencia técnica mediante prácticas como desarrollo guiado por pruebas (TDD), programación en parejas, integración continua, refactorización, diseño simple y estándares de codificación.

En cuanto a las **implicaciones para SSDLC**, las prácticas técnicas de *XP* son naturalmente compatibles con el SSDLC. TDD se extiende a pruebas de seguridad, la programación en parejas proporciona revisión continua de aspectos de seguridad, la integración continua permite detectar problemas tempranamente, y los estándares de codificación incluyen directrices de codificación segura.

4.3.4 LEAN DEVELOPMENT

Lean Development adapta los principios del Sistema de Producción de Toyota al desarrollo de software, formalizado por Mary y Tom Poppendieck en 2003. Se fundamenta en siete principios: eliminar desperdicios, amplificar el aprendizaje, decidir lo más tarde posible, entregar rápido, empoderar al equipo, construir calidad desde el

inicio y optimizar el conjunto. La metodología enfatiza la eliminación de cualquier actividad que no aporte valor al cliente, promoviendo equipos pequeños y autoorganizados, entrega incremental en ciclos cortos y validación continua con usuarios reales.

En cuanto a las **implicaciones para SSDLC**, las prácticas de seguridad se priorizan según el valor que aportan al cliente, eliminando verificaciones redundantes. El principio de "decidir lo más tarde posible" permite adaptar controles de seguridad basándose en amenazas reales identificadas en lugar de suposiciones. La entrega incremental facilita la detección temprana de vulnerabilidades mediante *feedback* continuo. Los equipos empoderados asumen responsabilidad compartida de seguridad, integrando controles de calidad y seguridad desde el inicio mediante pruebas automatizadas y refactorización continua.

4.4 INTEGRACIÓN DEL SSDLC EN EL CICLO ÁGIL

La integración del SSDLC en metodologías ágiles adapta las prácticas de seguridad a la cadencia iterativa:

- **Planificación de iteración**: Incorporar requisitos de seguridad como historias de usuario, realizar modelado de amenazas incremental, y definir criterios de seguridad en la *Definition of Done*.
- **Durante el desarrollo**: Aplicar desarrollo seguro, utilizar análisis estático (SAST) integrado en entornos de desarrollo, gestionar dependencias con análisis de composición (SCA), y prevenir secretos en código mediante detección automatizada.
- **Verificación**: Integrar pruebas de seguridad en suites automatizadas, realizar análisis dinámico (DAST), *fuzzing*, y revisiones de código con verificación de seguridad.
- **Retrospectiva**: Revisar postura de seguridad, identificar vulnerabilidades y causas raíz, y definir mejoras a prácticas de seguridad para futuras iteraciones.

La integración conceptual del SSDLC en metodologías ágiles establece el marco teórico necesario, pero la implementación práctica requiere un enfoque más profundo que automatice y operacionalice la seguridad en cada iteración. El modelo DevSecOps representa la evolución natural de esta integración, extendiendo los principios ágiles al dominio operacional mediante la automatización extensiva de controles de seguridad en pipelines de entrega continua.

5. MARCOS DE REFERENCIA PARA LA IMPLEMENTACION DEL SSDLC

La implementación efectiva de un Ciclo de Vida de Desarrollo de Software Seguro requiere el respaldo de marcos de referencia, estándares y metodologías reconocidas internacionalmente que proporcionen estructura, prácticas comprobadas y un lenguaje común para las organizaciones. Estos marcos no son prescriptivos en cuanto a herramientas o tecnologías específicas, sino que se centran en los resultados y objetivos de seguridad que deben alcanzarse, permitiendo así su aplicación independientemente del tamaño de la organización, sector, tecnología utilizada o modelo de desarrollo adoptado.

Existen diversos marcos que abordan el desarrollo seguro desde diferentes perspectivas. Algunos proporcionan prácticas fundamentales a implementar a lo largo del ciclo de vida (como NIST SSDF), otros ofrecen modelos de madurez para evaluar y mejorar progresivamente las capacidades de seguridad (como OWASP SAMM v2), algunos documentan prácticas observadas en la industria para benchmarking (como BSIMM), otros establecen procesos prescriptivos integrados con herramientas específicas (como Microsoft SDL), y otros proporcionan guías prácticas de codificación segura para desarrolladores (como OWASP *Secure Coding Practices*).

A lo largo de esta guía se presentan y desarrollan en profundidad dos marcos de referencia principales reconocidos internacionalmente: *OWASP Software Assurance Maturity Model* (SAMM v2) y *NIST Secure Software Development Framework* (SSDF v1.1). Además de estos, otros marcos reconocidos que están presentes en este documento y son ampliamente usados en el mercado son *Building Security In Maturity Model* (BSIMM), *Microsoft Security Development Lifecycle* (SDL) y OWASP *Secure Coding Practices*.

Estos marcos se complementan con recursos especializados que proporcionan orientación práctica para la implementación, incluyendo guías de desarrollo seguro (OWASP *Developer Guide*), estándares de verificación de seguridad (OWASP ASVS v5.0.0 para aplicaciones web y OWASP MASVS para aplicaciones móviles) y catálogos de vulnerabilidades comunes (OWASP Top 10 y CWE Top 25) que ayudan a los equipos de desarrollo a identificar y prevenir los problemas de seguridad más críticos.

La selección y combinación de estos marcos y recursos debe realizarse en función de las necesidades específicas, nivel de madurez, perfil de riesgo y contexto operativo de cada organización y proyecto, proporcionando así una aproximación práctica y adaptable para la implementación del desarrollo seguro de software.

Antes de abordar los principales marcos de referencia, conviene distinguir entre metodología de desarrollo y marco de referencia de seguridad. Las metodologías de desarrollo, como el modelo en cascada, el modelo en espiral o las metodologías ágiles, definen cómo se organiza el trabajo, cómo se estructura el ciclo de vida y cómo se entrega el software. Por su parte, los marcos de referencia de seguridad, como NIST SSDF, OWASP SAMM, Microsoft SDL o BSIMM, definen qué prácticas, capacidades y controles de seguridad conviene implantar dentro de ese ciclo de vida.

En términos generales, estos marcos pueden aplicarse tanto en metodologías tradicionales como en metodologías ágiles. La diferencia principal no radica en el marco a aplicar, sino en la forma de implantación: en entornos tradicionales las actividades de seguridad suelen concentrarse en hitos o fases más definidas, mientras que en entornos ágiles se ejecutan de manera más iterativa, continua y automatizada.

5.1 OWASP SAMM V2

OWASP SAMM (*Software Assurance Maturity Model*) [1] es un modelo prescriptivo y abierto para evaluar y mejorar la madurez de seguridad en el desarrollo de software, caracterizado por su carácter evolutivo y basado en riesgos, asegurando que se pueda implementar independientemente de la organización. Entre sus características principales destacan:

- **Medible:** define niveles de madurez claros para las diferentes prácticas de seguridad.
- **Accionable:** ofrece rutas e indicadores claros para la mejora de los ciclos de vida del desarrollo seguro.
- **Versátil:** agnóstico a las tecnologías, los lenguajes de programación, los procesos y la organización sobre la que se aplique.

El modelo OWASP SAMM está organizado en 5 funciones de negocio, cada una contiene 3 prácticas de seguridad, divididas en 2 flujos (Streams A y B) y 3 niveles de madurez.

Los niveles de madurez permiten identificar la implicación e implantación de la seguridad en el modelo SSDLC, donde los inferiores suelen ser más fáciles de ejecutar y requieren menos formalización, ideales para empresas de un menor tamaño o digitalización, siendo el mayor nivel de madurez el específico para empresas más tecnológicas.

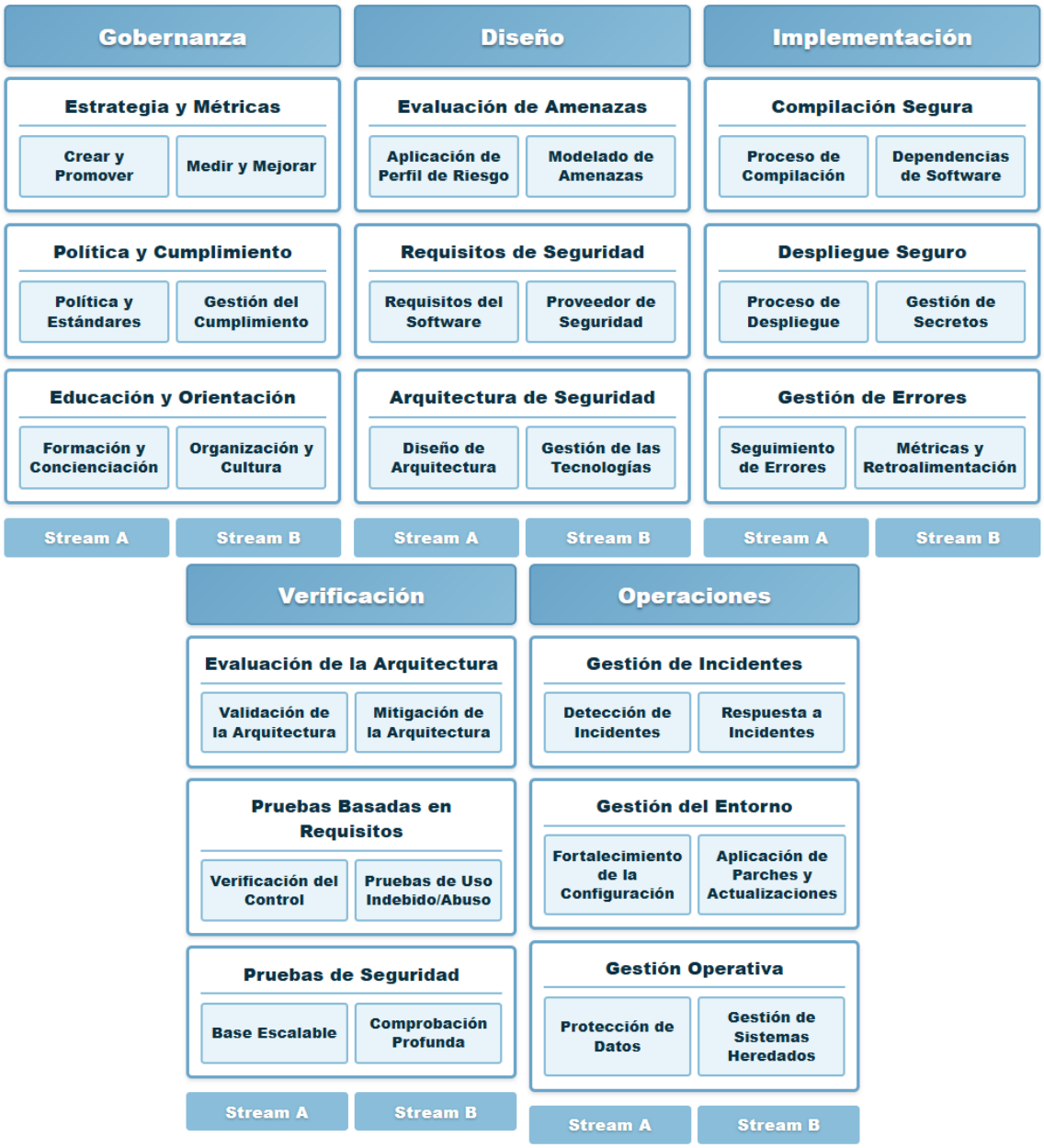


Figura 7. Funciones de Negocio de OWASP SAMM

5.1.1 GOBERNANZA

La gobernanza se centra en los procesos y actividades relacionados con la forma en que una organización gestiona las actividades generales de desarrollo de software. Esto incluye cuestiones que afectan a los grupos interfuncionales implicados en el desarrollo, así como a los procesos empresariales establecidos a nivel de la organización.

5.1.1.1 ESTRATEGIA Y MÉTRICAS

Constituye la base de las actividades de software seguro mediante la creación de un plan general eficiente y eficaz para alcanzar los objetivos de seguridad de software dentro de una organización. Se centra en elaborar el plan, mantenerlo y difundirlo.

a) **Crear y promover:** Este flujo trata sobre la creación y promoción de una hoja de ruta de seguridad de aplicaciones para establecer los objetivos de seguridad de la organización.

b) **Medir y mejorar:** Trata de definir métricas eficientes y eficaces que permitan identificar el cumplimiento de los KPIs (*Key Performance Indicators*) y objetivos definidos.

5.1.1.2 POLÍTICA Y CUMPLIMIENTO

Permite comprender y cumplir los requisitos legales y normativos externos, al tiempo que impulsa las normas de seguridad internas para garantizar el cumplimiento de una manera que esté alineada con el objetivo comercial de la organización.

a) **Política y estándares:** Cumplir con las políticas y estándares organizativos, desarrollando requisitos de seguridad mínimos.

b) **Gestión del cumplimiento:** Identificar y proporcionar requisitos de cumplimiento para las políticas existentes.

5.1.1.3 EDUCACIÓN Y ORIENTACIÓN

Se centra en aumentar los conocimientos de la organización en materia de software seguro. El objetivo es dotar al personal involucrado en el ciclo de vida del software con conocimientos y recursos para diseñar, desarrollar e implementar software seguro. Con un mejor acceso a la información, los equipos de proyecto pueden identificar y mitigar de forma proactiva los riesgos de seguridad específicos que se aplican a la organización.

a) **Formación y concienciación:** Aumentar los conocimientos generales sobre seguridad del software entre las diferentes partes interesadas de la organización.

b) **Organización y cultura:** Promover la cultura de la seguridad de las aplicaciones (con roles como el *Security Champion*) dentro de la organización como un factor importante para el éxito de un proyecto SDLC.

5.1.2 DISEÑO

El diseño se refiere a los procesos y actividades relacionados con la forma en que una organización define los objetivos y crea software dentro de los proyectos de desarrollo. En general, esto incluirá la recopilación de requisitos, la especificación de la arquitectura de alto nivel y el diseño detallado.

5.1.2.1 EVALUACIÓN DE AMENAZAS

Esta práctica permite identificar y comprender los riesgos a nivel de proyecto basándose en la funcionalidad del software que se está desarrollando y las características del entorno de ejecución. A partir de los detalles sobre las amenazas y los posibles ataques contra cada proyecto, la organización en su conjunto opera de manera más eficaz gracias a la toma de mejores decisiones sobre la priorización de las iniciativas de seguridad.

Además, las decisiones sobre la aceptación de riesgos están más fundamentadas y, por lo tanto, mejor alineadas con el negocio.

a) **Perfil de riesgo de aplicación:** Un perfil de riesgo de las aplicaciones ayuda a identificar qué aplicaciones pueden suponer una amenaza grave para la organización si fueran atacadas o sufrieran una brecha de seguridad.

b) **Modelado de amenazas:** El modelado de amenazas tiene por objetivo ayudar a los equipos de desarrollo de software a comprender qué riesgos existen en lo que se está creando, qué podría salir mal y cómo podemos mitigar o remediar los riesgos. Existen múltiples tipos de modelados de amenazas, como STRIDE o DREAD, aplicables según el objetivo de la organización.

5.1.2.2 REQUISITOS DE SEGURIDAD

La finalidad de esta práctica es definir los requisitos de seguridad adecuados para el software y sus proveedores.

a) **Requisitos de software:** Los requisitos de software especifican los objetivos y expectativas para proteger el servicio y los datos que constituyen el núcleo de la aplicación.

b) **Seguridad del proveedor:** La seguridad de los proveedores se ocupa de los requisitos relativos a las organizaciones proveedoras en el contexto del desarrollo de la aplicación, en particular en el caso del desarrollo externalizado.

5.1.2.3 ARQUITECTURA DE SEGURIDAD

Esta práctica se centra en la seguridad relacionada con los componentes y la tecnología con los que se trabaja durante el diseño arquitectónico del software. El diseño de arquitectura segura analiza la selección y composición de los componentes que forman la base de su solución, centrándose en sus propiedades de seguridad. La gestión tecnológica analiza la seguridad de las tecnologías de apoyo utilizadas durante el desarrollo, la implementación y las operaciones, como las herramientas de implementación y los sistemas operativos.

a) **Diseño de arquitectura:** El diseño de la arquitectura de un software puede afectar significativamente a la seguridad de este, por lo que se establecen los principios de seguridad por diseño (o "*security by design*"), estableciendo patrones seguros y comunes para su adopción en la compañía.

b) **Gestión de las tecnologías:** Las tecnologías y los marcos son los pilares de cualquier solución de software. Es necesario examinar sus propiedades de seguridad para garantizar un nivel adecuado de seguridad y anticiparse a cualquier posible problema en este ámbito, buscando una homogenización y un estándar seguro.

5.1.3 IMPLEMENTACIÓN

La implementación se centra en los procesos y actividades relacionados con la forma en que una organización crea e implementa componentes de software y sus defectos relacionados. Las actividades dentro de la función de implementación son las que más impacto tienen en el día a día de los desarrolladores. El objetivo común es entregar software que funcione de forma fiable y con un mínimo de defectos de seguridad.

5.1.3.1 CONSTRUCCIÓN SEGURA

El objetivo es crear un proceso de compilación repetible de forma coherente y tener en cuenta la seguridad de las dependencias de terceros usadas por las aplicaciones. La práctica hace hincapié en la importancia de crear software de forma estandarizada y repetible, y de hacerlo utilizando componentes seguros, incluidas las dependencias de software de terceros.

El primer flujo se centra en eliminar cualquier subjetividad del proceso de compilación mediante la búsqueda de la automatización total. Un pipeline de construcción automatizada (comúnmente conocido como CI) puede incluir comprobaciones de seguridad automatizadas adicionales, como SAST, SCA o DAST, para obtener una mayor garantía y señalar los fallos de seguridad de forma temprana, por ejemplo, parando la compilación (proceso conocido como *Security Gate*).

El segundo flujo reconoce la prevalencia de las dependencias de software en las aplicaciones modernas. Su objetivo es identificarlas y realizar un seguimiento de su estado de seguridad para contener el impacto de su inseguridad en una aplicación que, por lo demás, es segura.

a) **Proceso de construcción:** Un proceso de construcción coherente garantiza que el software que se está implementando sea previsible y esté directamente vinculado al código fuente, auditándolo a través de diferentes tipos de pruebas de seguridad.

b) **Dependencias de software:** Las bibliotecas externas son una parte muy importante del software moderno. Las actividades de este flujo ayudan a crear una visión de las bibliotecas externas y garantizan que su nivel de seguridad sea adecuado.

5.1.3.2 DESPLIEGUE SEGURO

El despliegue seguro está enfocado a garantizar que la seguridad y la integridad de las aplicaciones desarrolladas no se vean comprometidas durante la implementación.

El primer tipo de actuación se centra en eliminar los errores manuales mediante la automatización del proceso de despliegue en la medida de lo posible, y en hacer que su éxito dependa de los resultados de las comprobaciones de verificación de seguridad integradas.

El segundo tipo se enfoca en proteger la privacidad y la integridad de los datos confidenciales, como contraseñas, tokens y otros secretos, necesarios para que las aplicaciones funcionen en entornos de producción.

a) **Proceso de despliegue:** Debe de garantizar ser repetible y coherente, lo cual garantiza que solo se entreguen los artefactos de software correctos en la producción.

b) **Gestión de secretos:** Dado que la ejecución segura de cualquier sistema de software requiere credenciales, este flujo garantiza un manejo adecuado de estos elementos confidenciales dentro del entorno de la organización.

5.1.3.3 GESTIÓN DE DEFECTOS

Esta práctica se centra en la gestión de los fallos de seguridad en el software y sus métricas asociadas. Con este fin, se recopilan, registran y analizan los defectos de seguridad del software para impulsar decisiones basadas en métricas.

a) **Seguimiento de vulnerabilidades:** Gestiona la recopilación y el seguimiento de todos los posibles fallos de seguridad de un software, desde fallos en su arquitectura hasta defectos de desarrollo y vulnerabilidades en tiempo de ejecución.

b) **Métricas y retroalimentación:** Impulsar la mejora de las actividades de seguridad dentro de la organización a través de métricas y obtención de *feedback* continuo.

5.1.4 VERIFICACIÓN

La verificación se centra en los procesos y actividades relacionados con la forma en que una organización comprueba y prueba los artefactos producidos a lo largo del desarrollo de software. Por lo general, esto incluye tareas de control de calidad, como las pruebas de seguridad, pero también puede incluir otras actividades de revisión y evaluación.

5.1.4.1 EVALUACIÓN DE LA ARQUITECTURA

La evaluación de la arquitectura tiene como fin garantizar que la aplicación y la arquitectura de la infraestructura donde ha sido desplegada cumplan adecuadamente todos los requisitos de seguridad y cumplimiento pertinentes, y mitiguen suficientemente las amenazas de seguridad identificadas.

a) **Validación de la arquitectura:** Permite la identificación de los componentes de la arquitectura e infraestructura de la aplicación y la verificación de que cumplen los objetivos y requisitos de seguridad.

b) **Mitigación de la arquitectura:** La mitigación de la arquitectura se centra en garantizar que todas las amenazas identificadas durante la evaluación de amenazas se mitiguen adecuadamente y que las arquitecturas de referencia existentes se actualicen para abordar cualquier amenaza no tratada.

5.1.4.2 PRUEBAS BASADAS EN REQUISITOS

Esta práctica se basa en el uso de pruebas de seguridad tanto positivas (verificación de controles) como negativas (pruebas de uso indebido/abuso) basadas en los requisitos. El objetivo es garantizar que los controles de seguridad implementados funcionen según lo previsto y satisfagan los requisitos de seguridad establecidos para el proyecto.

a) **Verificación del control:** Se busca una validación donde se defina que se cumplen los controles y requisitos de seguridad mediante pruebas derivadas de los requisitos, y evita la introducción de nuevos defectos en versiones posteriores mediante pruebas de regresión.

b) **Pruebas de uso indebido/abuso:** Las pruebas de uso indebido/abuso aprovechan el *fuzzing* con el fin de identificar debilidades en las características para atacar una aplicación, así como pruebas de fallos de lógica o simulaciones de pruebas de denegación de servicio.

5.1.4.3 PRUEBAS DE SEGURIDAD

El propósito de las pruebas de seguridad es la detección y resolución de problemas básicos de seguridad mediante la automatización, lo que permite que las pruebas manuales se centren en vectores de ataque más complejos. Se aprovecha el hecho de que, si bien las pruebas de seguridad automatizadas son rápidas y se adaptan bien a numerosas aplicaciones, las pruebas en profundidad basadas en un buen conocimiento de una aplicación y su lógica empresarial a menudo solo son posibles mediante pruebas de seguridad manuales más lentas realizadas por expertos. Por lo tanto, cada flujo tiene un enfoque central.

a) **Línea base escalable:** Se centra en el uso de herramientas de pruebas automatizadas específicas para cada aplicación que integran la validación de la seguridad en el proceso de compilación e implementación (conocido como análisis DAST). El objetivo de esta corriente es favorecer la amplitud (un amplio espectro de aplicaciones) sobre la profundidad de las pruebas.

b) **Comprensión profunda:** Se centra en la realización de pruebas de seguridad manuales de componentes de alto riesgo, utilizando vectores de ataque complejos con el objetivo de convertir las pruebas de seguridad avanzadas en una parte integral del proceso de desarrollo (conocidos como pruebas de intrusión).

5.1.5 OPERACIONES

La función empresarial de operaciones abarca aquellas actividades necesarias para garantizar que se mantengan la confidencialidad, la integridad y la disponibilidad a lo largo de toda la vida útil operativa de una aplicación y sus datos asociados. Una mayor madurez con respecto a esta función empresarial proporciona una mayor garantía de que la organización es resistente ante las interrupciones operativas y es capaz de responder a los cambios en el panorama operativo.

5.1.5.1 GESTIÓN DE INCIDENTES

Aborda las actividades llevadas a cabo para mejorar la detección y la respuesta de la organización ante incidentes de seguridad. Una vez que la organización tiene aplicaciones en funcionamiento, es probable que se enfrente a incidentes de seguridad, como ataques de denegación de servicio o intento de intrusiones. En este modelo se define un incidente de seguridad como una violación, o la amenaza de una violación inminente, de al menos uno de los objetivos de seguridad de un activo, ya sea debido a un comportamiento malicioso o negligente.

a) **Detección de incidentes:** La detección de incidentes se refiere al proceso de determinar si un evento identificado como relevante para la seguridad es, de hecho, un incidente de seguridad. Las actividades de este flujo se centran en la capacidad de la organización para identificar los incidentes de seguridad cuando se producen y para iniciar las actividades de respuesta a incidentes adecuadas.

b) **Respuesta a incidentes:** La respuesta a incidentes comienza en el momento en que se reconoce y verifica la existencia de un incidente de seguridad. El objetivo es actuar de forma coordinada y eficaz para limitar al máximo los daños adicionales. Las actividades de este flujo se centran en la capacidad de la organización para responder de forma adecuada y eficaz a los incidentes de seguridad notificados.

5.1.5.2 GESTIÓN DEL ENTORNO

La gestión del entorno describe las actividades proactivas que se llevan a cabo para mejorar y mantener la seguridad de los entornos en los que operan las aplicaciones de la organización.

El trabajo de la organización en materia de seguridad de las aplicaciones no termina una vez que la aplicación entra en funcionamiento. Se lanzan regularmente nuevas funciones de seguridad y parches para los distintos elementos de las tecnologías que se utilizan, hasta que quedan obsoletos o dejan de ser compatibles.

a) **Fortalecimiento de la configuración:** Gestión por parte de la organización de las configuraciones relacionadas con la seguridad en todos los elementos de la pila tecnológica. Se hace hincapié en aquellos elementos (por ejemplo, sistemas operativos, contenedores, marcos, servicios, dispositivos y bibliotecas) obtenidos de terceros, ya que su arquitectura y diseño no están bajo el control de la organización.

b) **Aplicación de parches y actualizaciones:** Las actividades de este flujo se centran en la gestión de los parches y actualizaciones para todos los elementos de la pila tecnológica. En el caso del software desarrollado por la organización, estas actividades se refieren al suministro de parches y actualizaciones a los clientes, así como a su aplicación a las soluciones gestionadas por la organización (por ejemplo, el software como servicio). En el caso de los elementos de terceros, estas actividades se refieren a la

aplicación oportuna por parte de la organización de las actualizaciones y parches recibidos.

5.1.5.3 GESTIÓN OPERATIVA

La gestión operativa gira en torno a actividades de apoyo operativo necesarias para mantener la seguridad a lo largo del ciclo de vida del producto. Aunque estas funciones no las realiza directamente una aplicación, la seguridad general de la aplicación y sus datos depende de su correcto funcionamiento. Implementar una aplicación en un sistema operativo no compatible con vulnerabilidades sin parchear o no almacenar los soportes de copia de seguridad de forma segura puede hacer que las protecciones integradas en esa aplicación sean irrelevantes.

a) **Protección de datos:** Garantizar que la organización proteja adecuadamente los datos en todos los aspectos de su creación, manejo, almacenamiento y procesamiento, usando algoritmos de cifrado robustos y protección de información.

b) **Gestión de sistemas heredados:** Desde la perspectiva de la organización como consumidora de recursos, las actividades de este flujo se centran en la identificación, gestión y seguimiento de sistemas, aplicaciones, dependencias de aplicaciones y servicios que no se utilizan, al final de su ciclo de vida o que no tienen soporte. La eliminación de sistemas o servicios en desuso es una mejora a la capacidad de gestión del entorno, reduce la superficie de ataque y permite un ahorro de costes directos e indirectos.

5.2 NIST SECURE SOFTWARE DEVELOPMENT FRAMEWORK (SSDF)

5.2.1 INTRODUCCIÓN AL MARCO NIST SSDF

El *NIST Special Publication 800-218* define el **Secure Software Development *Framework* (SSDF)** [2] como un conjunto fundamental de prácticas de desarrollo seguro de software de alto nivel que pueden integrarse en cada implementación del ciclo de vida de desarrollo. Este documento recomienda prácticas que deben ayudar a los proveedores de software a reducir el número de vulnerabilidades en el software liberado, reducir el impacto potencial de la explotación de vulnerabilidades no detectadas o no abordadas, y abordar las causas raíz de los fallos de seguridad para prevenir recurrencias futuras.

El SSDF no prescribe cómo implementar cada práctica, centrándose en los resultados de las prácticas en lugar de en las herramientas, técnicas y mecanismos para conseguirlos. Esto significa que el SSDF puede ser utilizado por organizaciones en cualquier sector o comunidad, independientemente del tamaño o sofisticación en ciberseguridad.

El marco NIST SSDF presenta las siguientes características distintivas, muy similares a aquellas destacadas para el marco OWASP SAMM v2:

- **Enfoque orientado a resultados:** Las prácticas se definen por los objetivos de seguridad que deben alcanzarse, no por las tecnologías específicas a utilizar.
- **Aplicabilidad universal:** Válido para cualquier tipo de desarrollo de software, independientemente de la tecnología, plataforma, lenguaje de programación o entorno operativo.
- **Lenguaje común:** Proporciona vocabulario compartido para facilitar la comunicación entre productores y adquirentes de software.
- **Flexibilidad:** Las organizaciones pueden adaptar las prácticas según sus necesidades, riesgos y recursos disponibles.

El documento proporciona un lenguaje común para describir las prácticas fundamentales de desarrollo seguro de software. No se requiere experiencia en desarrollo seguro de software para entender las prácticas. El lenguaje común ayuda a facilitar las comunicaciones sobre prácticas de software seguro entre *stakeholders* organizacionales internos y externos.

5.2.2 AUDIENCIA Y APLICABILIDAD DEL SSDF

Dentro de los perfiles a los cuales se dirige este marco destacan dos perfiles: productores de software y los adquirientes de software.

En cuanto a los productores de software destacan los proveedores de productos comerciales listos para usar o *COTS*, desarrolladores de software gubernamental o *GOTS*, desarrolladores de software personalizado o equipos de desarrollo interno entre otros.

Sobre los adquirentes de software, aparecen tanto agencias federales como otras organizaciones que utilizan software externo.

El SSDF proporciona ventajas específicas para cada audiencia:

- Para **productores de software**:
  - Marco estructurado para **implementar o mejorar prácticas de desarrollo seguro** existentes.
  - Base para **comunicar capacidades de seguridad** a clientes, reguladores y auditores.
  - Criterios para **definir requisitos de seguridad a proveedores** de componentes de terceros.
  - Orientación independiente de la metodología de desarrollo utilizada (cascada, ágil, DevOps).
- Para **adquirentes de software**:
  - Criterios objetivos para **evaluar las prácticas de seguridad** de proveedores potenciales.

- Base para **establecer cláusulas de seguridad** en pliegos de contratación y contratos.
- Marco para **verificar el cumplimiento de estándares** de desarrollo seguro durante la ejecución contractual.
- **Lenguaje común** para especificar requisitos y evaluar el cumplimiento de estos.

5.2.3 PRINCIPIOS FUNDAMENTALES DEL SSDF

5.2.3.1 ENFOQUE BASADO EN RIESGOS

Una de las grandes cualidades de este marco es la influencia directa de ciertos factores usados al decidir qué prácticas implantar y cuánto tiempo y recursos dedicar a cada práctica actores como el riesgo, así como el costo, la viabilidad y la aplicabilidad.

- **Nivel de riesgo del software:** Perfil obtenido de la criticidad del software, sensibilidad de los datos que maneja, exposición a amenazas e impacto potencial de una brecha de seguridad.
- **Recursos disponibles:** Compendio entre presupuesto, personal cualificado, tiempo disponible y herramientas existentes en la organización.
- **Viabilidad técnica:** Capacidad técnica actual de la organización, así como la madurez de los procesos existentes y complejidad de implementación.
- **Aplicabilidad contextual:** Relevancia de cada práctica para el tipo específico de software, metodología de desarrollo y entorno operativo.

La intención del estándar SSDF no es crear una lista de verificación a seguir, sino proporcionar una base para planificar e implementar un enfoque basado en riesgos para adoptar prácticas de desarrollo seguro de software.

5.2.3.1.1 RESPONSABILIDAD COMPARTIDA EN MODELOS CLOUD

La responsabilidad de implementar las prácticas puede estar distribuida entre diferentes organizaciones, basándose en la entrega del software y servicios (por ejemplo, infraestructura como servicio, software como servicio, plataforma como servicio, contenedor como servicio, *serverless*…).

En estas situaciones, la pauta general dentro del marco es usar un **modelo de responsabilidad compartida** que involucra a los proveedores de plataforma/servicio y la organización inquilina que consume esas plataformas/servicios. En estos escenarios, se debe:

- Establecer **acuerdos claros con los proveedores** especificando qué parte es responsable de cada práctica y tarea del SSDF.
- Definir mecanismos mediante los cuales cada proveedor **atestiguará su conformidad** con las prácticas acordadas.

- **Documentar las responsabilidades** en contratos, acuerdos de nivel de servicio (SLA) y políticas de seguridad.
- Realizar **revisiones periódicas del cumplimiento** de las responsabilidades asignadas.

5.2.3.1.2 AUTOMATIZACIÓN Y ARTEFACTOS

En el marco del NIST se desarrolla ampliamente el uso de la automatización con el fin de mejorar y escalar los procesos de seguridad de forma flexible, sencilla y beneficiosa para la organización.

- **El SSDF enfatiza la importancia de utilizar automatización para:**
  - **Reducir el esfuerzo humano** requerido para implementar prácticas de seguridad.
  - Mejorar la **precisión, reproducibilidad y comprensión** de las verificaciones de seguridad.
  - Proporcionar **trazabilidad** mediante la generación automática de artefactos y evidencias.
  - Facilitar la **integración de seguridad en flujos de trabajo** de desarrollo existentes.
  - Permitir la **implementación de prácticas a escala** en organizaciones grandes o con múltiples proyectos.
- **Los artefactos generados deben de cumplir según el SSDF:**
  - **Documentar las actividades de seguridad** realizadas durante el desarrollo.
  - Proporcionar **evidencias para auditorías** y evaluaciones de cumplimiento.
  - **Facilitar el análisis** de causa raíz cuando se descubren vulnerabilidades.
  - Servir como **base para la mejora continua** de los procesos de desarrollo seguro.

5.2.4 ESTRUCTURA DEL SSDF: LAS CUATRO CATEGORÍAS DE PRACTICAS

El **SSDF versión 1.1** define prácticas fundamentales, sólidas y seguras recomendadas basadas en documentos de prácticas de desarrollo seguro establecidos. Estas prácticas se dividen en **4 categorías principales**, a consultar en la siguiente figura y que posteriormente se desglosarán en la presente guía.

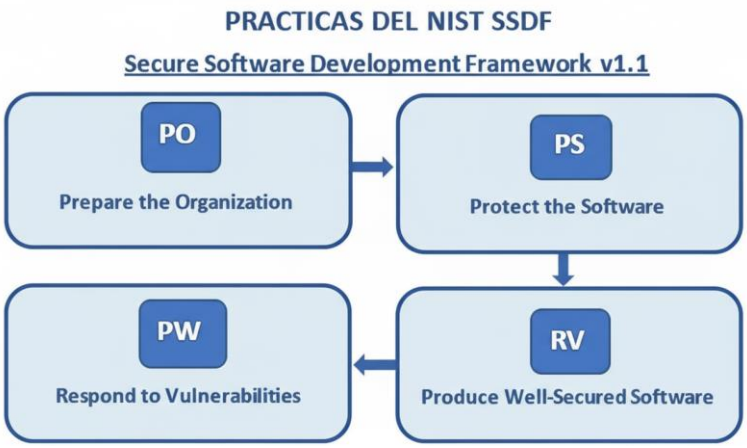


Figura 8. Prácticas del NIST SSDF v1.1

Cada práctica del NIST SSDF se estructura mediante **seis elementos fundamentales** que facilitan su comprensión, implementación y adaptación a diferentes contextos organizacionales (a consultar en la siguiente figura). Estos elementos proporcionan un marco consistente que permite a las organizaciones identificar, justificar e implementar cada práctica de manera efectiva.

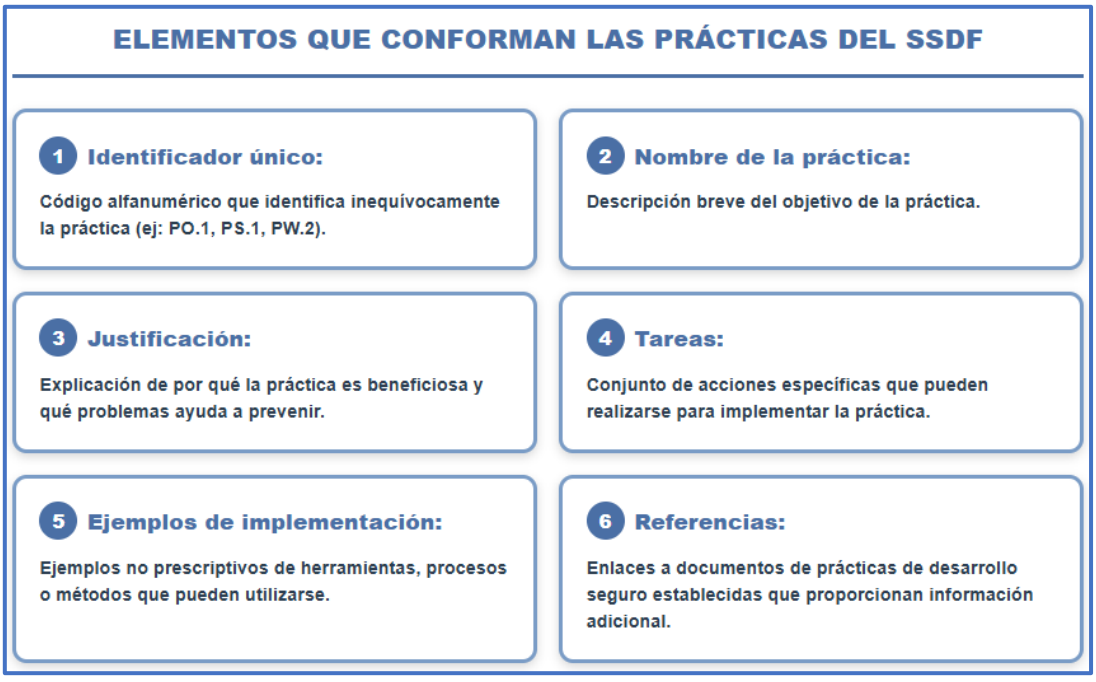


Figura 9. Elementos de las Prácticas del NIST SSDF v1.1

5.2.4.1 CATEGORÍA PO - PREPARAR LA ORGANIZACIÓN

Las organizaciones deben asegurar que sus personas, procesos y tecnología estén preparados para realizar desarrollo seguro de software a nivel organizacional. Muchas organizaciones encontrarán que algunas **prácticas PO** también son aplicables a subconjuntos de su desarrollo de software, como grupos de desarrollo individuales o proyectos.

Esta categoría establece los **fundamentos organizacionales necesarios** antes de iniciar el desarrollo de software. Las prácticas *PO* aseguran que la organización cuente con:

- **Requisitos de seguridad** claramente definidos y documentados.
- **Roles y responsabilidades** bien establecidos para todos los participantes en el SDLC.
- Cadenas de **herramientas (toolchains) implementadas** y mantenidas adecuadamente.
- **Criterios objetivos** para verificar la seguridad del software durante el desarrollo.
- **Entornos de desarrollo protegidos** contra amenazas internas y externas.

Dentro de la propia categoría se definen los fundamentos que componen la categoría *PO*, que permiten establecer objetivos de seguridad.

5.2.4.1.1 PO.1 - DEFINIR REQUISITOS DE SEGURIDAD PARA EL DESARROLLO DE SOFTWARE

La finalidad es asegurar que los requisitos de seguridad para el desarrollo de software sean conocidos en todo momento para que puedan tenerse en cuenta a lo largo del SDLC y se minimice la duplicación de esfuerzos. Esto incluye requisitos de fuentes internas (políticas organizacionales, objetivos de negocio, estrategia de gestión de riesgos…) y externas (leyes y regulaciones aplicables). Entre las tareas que tiene esta sección se encuentran:

- ✓ PO.1.1 - Identificar y documentar requisitos de seguridad para infraestructuras y procesos de desarrollo.
- ✓ PO.1.2 - Identificar y documentar requisitos de seguridad para el software desarrollado.
- ✓ PO.1.3 - Comunicar requisitos a terceros proveedores de componentes

5.2.4.1.2 PO.2 - IMPLEMENTAR ROLES Y RESPONSABILIDADES

El propósito es asegurar que todas las personas dentro y fuera de las organizaciones involucradas en el SDLC estén preparadas para realizar sus roles y responsabilidades relacionados con el SDLC a lo largo de todo el ciclo de vida. En este punto se incluye:

- ✓ PO.2.1 - Crear y mantener roles y responsabilidades
- ✓ PO.2.2 - Proporcionar formación basada en roles
- ✓ PO.2.3 - Obtener compromiso de la alta dirección

5.2.4.1.3 PO.3 - IMPLEMENTAR CADENAS DE HERRAMIENTAS DE SOPORTE (TOOLCHAINS)

La finalidad de esta fase es el uso de la automatización para reducir el esfuerzo humano y mejorar la precisión, reproducibilidad, usabilidad y comprensión de las prácticas de

seguridad a lo largo del SDLC, así como proporcionar una forma de documentar y demostrar el uso de estas prácticas. Las tareas de esta sección se componen de:

- ✓ PO.3.1 - Especificar herramientas para cada práctica
- ✓ PO.3.2 - Seguir prácticas de seguridad para las herramientas
- ✓ PO.3.3 - Configurar herramientas para generar artefactos

5.2.4.1.4 PO.4 - DEFINIR Y USAR CRITERIOS PARA VERIFICACIONES DE SEGURIDAD DEL SOFTWARE

Su propósito es ayudar a asegurar que el software resultante del SDLC cumpla con las expectativas de la organización mediante la definición y uso de criterios para verificar la seguridad del software durante el desarrollo. Entre las tareas que tiene esta sección se encuentran:

- ✓ PO.4.1 - Definir criterios para verificaciones de seguridad
- ✓ PO.4.2 - Implementar mecanismos para recopilar información

5.2.4.1.5 PO.5 - IMPLEMENTAR Y MANTENER ENTORNOS SEGUROS PARA EL DESARROLLO DE SOFTWARE

Su finalidad es asegurar que todos los componentes de los entornos para desarrollo de software estén fuertemente protegidos de amenazas internas y externas para prevenir compromisos de los entornos o del software que se está desarrollando o manteniendo en ellos. Las tareas que incluyen esta sección son:

- ✓ PO.5.1 - Separar y proteger cada entorno de desarrollo
- ✓ PO.5.2 - Asegurar y endurecer *endpoints* de desarrollo

5.2.4.2 CATEGORÍA PS - PROTEGER EL SOFTWARE (PROTECT THE SOFTWARE)

Las organizaciones deben proteger todos los componentes de su software de manipulación y acceso no autorizado.

Esta categoría se centra en salvaguardar el software y todos sus componentes durante todo el ciclo de vida, desde el código fuente hasta los artefactos distribuidos. Las **prácticas PS** aseguran:

- Protección del código fuente contra accesos no autorizados y modificaciones maliciosas.
- Disponibilidad de mecanismos para que los adquirentes verifiquen la integridad del software.
- Archivado seguro de todas las versiones de software con sus datos de procedencia asociados.

5.2.4.2.1 PS.1 - PROTEGER TODAS LAS FORMAS DE CÓDIGO DE ACCESO NO AUTORIZADO Y MANIPULACIÓN

El propósito es ayudar a prevenir cambios no autorizados al código, tanto inadvertidos como intencionales, que podrían eludir o negar las características de seguridad previstas del software. Para código que no está destinado a ser públicamente accesible, esto ayuda a prevenir el robo del software y puede dificultar o hacer más costoso para los atacantes encontrar vulnerabilidades. La tarea que incluye esta categoría es:

- ✓ PS.1.1 - Almacenar código basándose en el principio de mínimo privilegio

5.2.4.2.2 PS.2 - PROPORCIONAR UN MECANISMO PARA VERIFICAR LA INTEGRIDAD DE LAS VERSIONES DE SOFTWARE

La finalidad es ayudar a los adquirentes de software a asegurar que el software que adquieren es legítimo y no ha sido manipulado. La tarea que compone esta categoría es:

- ✓ PS.2.1 - Hacer disponible información de verificación de integridad

5.2.4.2.3 PS.3 - ARCHIVAR Y PROTEGER CADA VERSIÓN DE SOFTWARE

El propósito es preservar las versiones de software para ayudar a identificar, analizar y eliminar vulnerabilidades descubiertas en el software después de su lanzamiento. Las tareas que conforman esta categoría son:

- ✓ PS.3.1 - Archivar de forma segura archivos y datos necesarios
- ✓ PS.3.2 - Recopilar, salvaguardar y compartir datos de procedencia

5.2.4.3 CATEGORÍA PW - PRODUCIR SOFTWARE BIEN ASEGURADO (PRODUCE WELL-SECURED SOFTWARE)

Las organizaciones deben producir software bien asegurado con vulnerabilidades de seguridad mínimas en sus liberaciones. Esta categoría abarca las prácticas directamente relacionadas con el desarrollo del software que determinan su calidad de seguridad. Las **prácticas PW** aseguran que:

- El diseño del software incorpore requisitos de seguridad y mitigue riesgos identificados.
- Se reutilicen componentes de software existentes y bien asegurados cuando sea apropiado.
- El código fuente se desarrolle siguiendo prácticas de desarrollo seguro.
- Se realicen revisiones y análisis exhaustivos del código.
- Se ejecuten pruebas de seguridad tanto automatizadas como manuales.
- El software se configure con ajustes seguros por defecto.

5.2.4.3.1 PW.1 - DISEÑAR SOFTWARE PARA CUMPLIR REQUISITOS DE SEGURIDAD Y MITIGAR RIESGOS

El objetivo de esta tarea es identificar y evaluar los requisitos de seguridad para el software; determinar qué riesgos de seguridad probablemente enfrentará el software

durante la operación y cómo el diseño y arquitectura del software deben mitigar esos riesgos. Abordar seguridad y riesgos durante el diseño del software (seguro por diseño) es clave para mejorar la seguridad del software y también ayuda a mejorar la eficiencia del desarrollo. Las tareas que componen este punto son:

- ✓ PW.1.1 - Usar modelado de riesgos
- ✓ PW.1.2 - Rastrear y mantener requisitos, riesgos y decisiones de diseño
- ✓ PW.1.3 - Construir soporte para características de seguridad estandarizadas

5.2.4.3.2 PW.2 - REVISAR EL DISEÑO DEL SOFTWARE PARA VERIFICAR CUMPLIMIENTO

Su propósito es ayudar a asegurar que el software cumplirá con los requisitos de seguridad y abordará satisfactoriamente la información de riesgo identificada. La tarea que incluye este punto es:

- ✓ PW.2.1 - Revisar el diseño para confirmar cumplimiento

5.2.4.3.3 PW.3 - REUTILIZAR SOFTWARE EXISTENTE Y BIEN ASEGURADO CUANDO SEA FACTIBLE

La finalidad de la fase es reducir los costes de la implementación de software, acelerar el desarrollo y disminuir la probabilidad de introducir vulnerabilidades adicionales reutilizando módulos y servicios de software que ya han tenido su postura de seguridad verificada. Las tareas que componen esta sección son:

- ✓ PW.4.1 - Adquirir y mantener componentes bien asegurados de terceros
- ✓ PW.4.2 - Crear y mantener componentes internos bien asegurados
- ✓ PW.4.4 - Verificar que componentes de terceros cumplan requisitos

5.2.4.3.4 PW.4 - CREAR CÓDIGO FUENTE ADHIRIÉNDOSE A PRÁCTICAS DE DESARROLLO SEGURO

Busca disminuir el número de vulnerabilidades de seguridad en el software y reducir costes minimizando las vulnerabilidades introducidas durante la creación del código fuente que cumplen o exceden los criterios de severidad de vulnerabilidades definidos por la organización. La tarea que compone esta sección es:

- ✓ PW.5.1 - Seguir prácticas de desarrollo seguro apropiadas

5.2.4.3.5 PW.5 - CONFIGURAR PROCESOS DE COMPILACIÓN PARA MEJORAR LA SEGURIDAD

La meta de esta fase es disminuir el número de vulnerabilidades de seguridad en el software y reducir costes eliminando vulnerabilidades antes de que ocurran las pruebas. Las tareas que incluye esta sección son:

- ✓ PW.6.1 - Usar herramientas actualizadas con características de seguridad

- ✓ PW.6.2 - Determinar y usar configuraciones

5.2.4.3.6 PW.6 - REVISAR Y/O ANALIZAR CÓDIGO LEGIBLE POR HUMANOS PARA IDENTIFICAR VULNERABILIDADES

El propósito es ayudar a identificar vulnerabilidades para que puedan corregirse antes de que el software se despliegue, previniendo así su explotación. Usar métodos automatizados reduce el esfuerzo y recursos necesarios para detectar vulnerabilidades. Las tareas que incluye esta sección son:

- ✓ PW.7.1 - Determinar métodos de revisión/análisis de código
- ✓ PW.7.2 - Realizar revisión/análisis según estándares de desarrollo seguro

5.2.4.3.7 PW.8 - PROBAR CÓDIGO EJECUTABLE PARA IDENTIFICAR VULNERABILIDADES

Esta fase permite ayudar a identificar vulnerabilidades para que puedan corregirse después de que el software se lance. Las tareas que incluye esta sección son:

- ✓ PW.8.1 - Determinar si realizar pruebas de código ejecutable
- ✓ PW.8.2 - Definir alcance, diseñar, realizar y documentar pruebas

5.2.4.3.8 PW.9 - CONFIGURAR SOFTWARE CON CONFIGURACIONES SEGURAS POR DEFECTO

El objetivo es ayudar a mejorar la seguridad del software en el momento de la instalación para reducir la probabilidad de que se despliegue con configuraciones de seguridad débiles, poniéndolo en mayor riesgo de compromiso. Las tareas que incluyen esta sección son:

- ✓ PW.9.1 - Definir línea base segura
- ✓ PW.9.2 - Implementar y documentar configuraciones predeterminadas

5.2.4.4 CATEGORÍA RV - RESPONDER A VULNERABILIDADES (RESPOND TO VULNERABILITIES)

Las organizaciones deben identificar vulnerabilidades residuales en sus liberaciones de software y responder apropiadamente para abordar esos fallos de seguridad, previniendo que ocurran otras similares en el futuro.

Esta categoría se centra en la gestión continua de vulnerabilidades una vez que el software está en producción. Las **prácticas RV** aseguran:

- Identificación proactiva y confirmación de vulnerabilidades de forma continua.
- Evaluación, priorización y remediación apropiada basada en riesgo.
- Análisis de causas raíz para prevenir recurrencias futuras.
- Mejora continua del proceso SDLC basándose en lecciones aprendidas.

5.2.4.4.1 RV.1 - IDENTIFICAR Y CONFIRMAR VULNERABILIDADES DE FORMA CONTINUA

El objetivo es ayudar a asegurar que las vulnerabilidades se identifiquen y remedien rápidamente de acuerdo con el riesgo, reduciendo la ventana de oportunidad para los atacantes. Las tareas que incluye esta sección son:

- ✓ RV.1.1 - Recopilar información sobre vulnerabilidades potenciales
- ✓ RV.1.2 - Revisar, analizar y/o probar código para identificar/confirmar vulnerabilidades
- ✓ RV.1.3 - Tener política de divulgación y remediación de vulnerabilidades

5.2.4.4.2 RV.2 - EVALUAR, PRIORIZAR Y REMEDIAR VULNERABILIDADES

La finalidad es ayudar a asegurar que las vulnerabilidades se remedien de acuerdo con el riesgo para reducir la ventana de oportunidad para los atacantes. Las tareas que incluye esta sección son:

- ✓ RV.2.1 - Analizar vulnerabilidades para planificar remediación
- ✓ RV.2.2 - Planificar e implementar respuestas de riesgo

5.2.4.4.3 RV.3 - ANALIZAR VULNERABILIDADES PARA IDENTIFICAR SUS CAUSAS RAÍZ

La finalidad es ayudar a reducir la frecuencia de vulnerabilidades en el futuro mediante el aprendizaje de las causas que las produjeron. Las tareas que componen esta sección son:

- ✓ RV.3.1 - Analizar vulnerabilidades para determinar causas raíz
- ✓ RV.3.2 - Analizar causas raíz a lo largo del tiempo para identificar patrones
- ✓ RV.3.3 - Revisar software en busca de vulnerabilidades similares
- ✓ RV.3.4 - Revisar y actualizar el proceso SDLC si es apropiado

5.2.5 IMPLEMENTACIÓN DEL MARCO SSDF: CONSIDERACIONES PRÁCTICAS

La implementación efectiva del *framework* SSDF requiere un enfoque personalizado y basado en riesgos que considere el contexto específico de cada organización:

- **Adopción basada en riesgos**: Evaluar la criticidad del software, considerar recursos disponibles (como presupuesto, personal o herramientas), analizar la viabilidad técnica y organizacional, priorizar prácticas según riesgo y adoptar un enfoque incremental.
- **Personalización de términos**: Definir "datos sensibles", "persona cualificada" y "bien asegurado" según criterios organizacionales; establecer nomenclatura de entornos y determinar frecuencias o umbrales de actividades críticas.

- **Gestión de artefactos**: Generar evidencias (registros de actividades, resultados de análisis, informes, SBOM…), almacenarlas de forma segura con controles de acceso y protección de integridad, y utilizarlas para demostración de cumplimiento normativo y auditorías.

5.2.6 RELACIÓN DEL SSDF CON EL ESQUEMA NACIONAL DE SEGURIDAD

El NIST SSDF proporciona prácticas que ayudan a cumplir con diversos controles del Esquema Nacional de Seguridad (Real Decreto 311/2022) [19]. La adopción del NIST SSDF facilita significativamente la conformidad con el ENS al proporcionar:

- Prácticas concretas y verificables para implementar los controles del ENS.
- Artefactos documentados que sirven como evidencia de cumplimiento.
- Un enfoque sistemático para la gestión de riesgos en el desarrollo de software.
- Métricas e indicadores para demostrar la eficacia de las medidas implementadas.

5.2.7 BENEFICIOS DE ADOPTAR EL NIST SSDF

5.2.7.1 PARA PRODUCTORES DE SOFTWARE

- **Reducción de vulnerabilidades**: Disminución significativa del número de vulnerabilidades en el software liberado mediante la aplicación sistemática de prácticas de seguridad.
- **Reducción de costes**: Detección y corrección temprana de problemas de seguridad, evitando los costes elevados de remediación en producción.
- **Mejora de eficiencia**: Automatización de verificaciones de seguridad que reduce el esfuerzo manual y acelera el desarrollo.
- **Lenguaje común**: Vocabulario estandarizado para comunicar capacidades y requisitos de seguridad con clientes, auditores y reguladores.
- **Mejora continua**: Marco estructurado para evaluar prácticas actuales, identificar brechas y planificar mejoras incrementales.
- **Ventaja competitiva**: Capacidad demostrable de producir software seguro, diferenciándose de competidores.

5.2.7.2 PARA ADQUIRENTES DE SOFTWARE

- **Criterios objetivos**: Base clara y verificable para evaluar las capacidades de seguridad de proveedores potenciales.
- **Reducción de riesgos**: Mayor confianza en la seguridad del software adquirido mediante la verificación de prácticas implementadas.

- **Requisitos contractuales**: Marco para establecer cláusulas de seguridad específicas y verificables en contratos de adquisición.
- **Verificación de cumplimiento**: Mecanismos claros para auditar y verificar que los proveedores cumplen con los requisitos acordados.
- **Gestión de la cadena de suministro**: Mayor visibilidad y control sobre los riesgos de seguridad en la cadena de suministro de software.

5.2.7.3 PARA LA ORGANIZACIÓN

- **Cumplimiento normativo**: Facilitación del cumplimiento con ENS, RGPD, NIS2, Cyber Resilience Act y otras regulaciones aplicables.
- **Cultura de seguridad**: Fomento de una cultura organizacional donde la seguridad es responsabilidad compartida de todos los involucrados en el desarrollo.
- **Alineación estratégica**: Integración de la seguridad del software con los objetivos estratégicos y la gestión de riesgos de la organización.
- **Mejora de la reputación**: Demostración pública del compromiso con la seguridad, fortaleciendo la confianza de clientes.
- **Resiliencia**: Mayor capacidad para identificar, responder y recuperarse de incidentes de seguridad relacionados con software.

5.3 OTROS FRAMEWORKS RELEVANTES

Además de los marcos de referencia OWASP SAMM y NIST SSDF desarrollados extensamente en las secciones anteriores, existen otros *frameworks* relevantes que aportan perspectivas complementarias al desarrollo seguro de software. Aunque estos marcos no se desarrollan a continuación con el mismo nivel de profundidad, resultan útiles para contextos organizacionales específicos y proporcionan enfoques alternativos que enriquecen el ecosistema del SSDLC.

5.3.1 BUILDING SECURITY IN MATURITY MODEL (BSIMM)

El *Building Security In Maturity Model* (BSIMM) [3] es un modelo descriptivo y empírico desarrollado por Synopsys que documenta las prácticas de seguridad de software observadas en organizaciones reales. A diferencia de los modelos prescriptivos como OWASP SAMM o NIST SSDF, BSIMM no proporciona recomendaciones de qué hacer, sino que describe qué están haciendo actualmente las organizaciones líderes en seguridad de software. El modelo se actualiza anualmente mediante el análisis de datos de más de 100 empresas participantes de diversos sectores, proporcionando un *benchmark* único para que las organizaciones comparen sus prácticas con las de la industria.

BSIMM está estructurado en cuatro dominios (Gobernanza, Inteligencia, *Touchpoints* de SSDLC y Despliegue) que abarcan 12 prácticas de seguridad y más de 120 actividades

específicas. Cada actividad se clasifica según su frecuencia de adopción en la industria, permitiendo a las organizaciones identificar prácticas comunes, emergentes o avanzadas. El modelo es especialmente valioso para organizaciones maduras que buscan *benchmarking* con sus pares industriales y para ejecutivos que necesitan justificar inversiones en seguridad mediante datos empíricos de adopción en el mercado.

5.3.2 MICROSOFT SECURITY DEVELOPMENT LIFECYCLE (SDL)

El *Microsoft Security Development Lifecycle* (SDL) [4] es un *framework* práctico y prescriptivo desarrollado por Microsoft en 2004 y refinado continuamente basándose en décadas de experiencia en desarrollo de software empresarial. SDL proporciona un conjunto de prácticas de seguridad organizadas en siete fases del ciclo de vida (Formación, Requisitos, Diseño, Implementación, Verificación, Lanzamiento y Respuesta) con énfasis particular en la integración con metodologías DevOps y CI/CD. El *framework* está diseñado específicamente para organizaciones que desarrollan software a escala y requieren un alto nivel de automatización y velocidad de entrega sin comprometer la seguridad.

SDL se distingue por su fuerte énfasis en la automatización de controles de seguridad, la integración nativa con herramientas de desarrollo de Microsoft y su enfoque pragmático para equipos que operan bajo metodologías ágiles. El *framework* incluye requisitos obligatorios y opcionales, permitiendo a las organizaciones adaptar su adopción según el nivel de riesgo del software. Microsoft proporciona documentación detallada, herramientas de soporte y programas de formación que facilitan la implementación práctica del SDL. Este marco es particularmente relevante para organizaciones que utilizan el ecosistema de desarrollo de Microsoft (Azure DevOps, GitHub, Visual Studio…) y buscan integración profunda entre seguridad y desarrollo.

5.3.3 OWASP SECURE CODING PRACTICES

OWASP *Secure Coding Practices* [14] es una guía de referencia rápida que proporciona un conjunto conciso de prácticas de codificación segura aplicables durante la fase de desarrollo del SDLC. Originalmente publicada como documento independiente, su contenido ha sido integrado y expandido en la OWASP *Developer Guide*, que ofrece orientación completa para desarrolladores sobre cómo construir aplicaciones web y móviles seguras. La guía cubre diez áreas fundamentales de seguridad: validación de entrada, codificación de salida, gestión de autenticación y contraseñas, gestión de sesiones, control de acceso, prácticas criptográficas, gestión de errores y auditoría, protección de datos, seguridad de comunicaciones y configuración de sistemas.

A diferencia de los *framework* más amplios como SAMM o SSDF, OWASP *Secure Coding Practices* se centra específicamente en proporcionar a los desarrolladores directrices prácticas y accionables que pueden aplicarse inmediatamente durante la escritura de código. La guía es independiente del lenguaje de programación y se complementa con otros recursos OWASP como el Top 10, ASVS (*Application Security Verification Standard*) y MASVS (*Mobile Application Security Verification Standard*), proporcionando así una

suite completa de recursos para desarrollo seguro. Es particularmente valiosa para equipos de desarrollo que necesitan una referencia rápida y práctica durante sus actividades diarias de implementación, y sirve como base para establecer estándares de desarrollo seguro organizacionales.

5.4 COMPARATIVA DE FRAMEWORKS Y MODELOS DE SEGURIDAD EN EL DESARROLLO DE SOFTWARE

En el contexto del ciclo de vida del desarrollo de software seguro (SSDLC), existen diversos *frameworks* y modelos que proporcionan orientación estructurada para integrar la seguridad en cada fase del desarrollo. Esta comparativa analiza los cinco *frameworks* detallados previamente y más relevantes del sector: OWASP SAMM, BSIMM, Microsoft SDL, NIST SSDF y OWASP Secure Coding Practices.

5.4.1 TABLA COMPARATIVA DE FRAMEWORKS

Criterio	OWASP SAMM v2	BSIMM	Microsoft SDL	NIST SSDF	OWASP Secure Coding
Año de Creación	2009	2008	2004	2020	2010
Última Versión	v2.0.3 (2022)	v14 (2024)	2024	v1.1 (2022)	Integrado en Developer Guide (2024)
Naturaleza	Modelo de Madurez Prescriptivo	Modelo Descriptivo Empírico	*Framework* Práctico Prescriptivo	*Framework* de Alto Nivel	Guía de Mejores Prácticas
Organización	OWASP Foundation	Synopsys	Microsoft	NIST	OWASP Foundation
Licencia/Coste	Open Source / Gratuito	Propietario / Pago	Gratuito	Público Gratuito	Open Source / Gratuito
Estructura	5 funciones, 15 prácticas, 2 streams, 3 niveles de madurez	4 dominios, 12 prácticas, más de 120 actividades	10 prácticas, 7 fases	4 categorías, 19 prácticas, 42 tareas	10 áreas de seguridad
Enfoque Principal	Mejora continua y madurez	Benchmarking de industria	Integración DevSecOps	Baseline de alto nivel	Desarrollo seguro
Público Objetivo	Cualquier organización	Organizaciones maduras	Empresas con modelos DevSecOps	Productores y adquirientes	Desarrolladores
Evaluación de Madurez	✓✓✓ (3 niveles por práctica)	✓✓✓ (basado en frecuencia)	✗	✗	✗
Benchmarking Industrial	✓ Limitado	✓✓✓ Excelente (100+ empresas)	✗	✗	✗
Guías de Implementación	✓✓✓ Muy detalladas	✓✓ Moderadas	✓✓✓ Muy detalladas	✓✓ Alto nivel	✓✓ Prácticas específicas

Documentación	✓✓✓ Excelente	✓✓ Buena	✓✓✓ Excelente	✓✓✓ Excelente	✓✓ Buena
Comunidad Activa	✓✓✓ Muy activa	✓ Limitada (comercial)	✓✓ Activa	✓✓ Activa	✓ Moderada
Complejidad de Adopción	Media	Alta	Alta	Baja-Media	Baja
Cobertura SDLC	Completa	Completa	Completa	Completa	Parcial (desarrollo)
Integración DevSecOps	✓✓ Buena	✓ Moderada	✓✓✓ Excelente	✓✓ Buena	✓✓ Buena
Mapeos a Otros *Frameworks*	✓✓✓ Múltiples	✓✓ Disponibles	✓ Algunos	✓✓✓ Múltiples	✓ Limitados
Requisito Regulatorio	No (voluntario)	No (voluntario)	No (voluntario)	Sí (OMB M-22-18 para US Gov)	No (voluntario)
Flexibilidad/ Adaptabilidad	✓✓✓ Muy alta	✓✓ Media	✓✓ Alta	✓✓✓ Muy alta	✓✓✓ Muy alta
ROI Demostrable	✓✓ Bueno	✓✓✓ Excelente	✓✓ Bueno	✓✓ Inicial	✓ Táctico

LEYENDA			
Soportado/Disponible	**Bien soportado**	**Excelente soporte**	**No disponible**
✓	✓✓	✓✓✓	✗

6. PRUEBAS DE SEGURIDAD Y DESARROLLO SEGURO

Uno de los grandes objetivos de la creación de modelos SSDLC suele estar vinculado a la creación de software resiliente y seguro, el cual proporcione unas altas capacidades de resistir potenciales ataques.

Para ello, en esta sección profundizaremos en las diferentes prácticas de seguridad y pruebas que se realizan a lo largo del ciclo de vida del software en relación con la implementación de código fuente, así como a las pautas más relevantes para el desarrollo de código seguro.

6.1 HERRAMIENTAS DE PRUEBAS DE SEGURIDAD EN APLICACIONES (AST)

Las pruebas de seguridad de aplicaciones (AST) son un proceso sistemático de identificación, análisis y reporte de vulnerabilidades de seguridad en las aplicaciones para protegerlas contra posibles ataques.

Evalúan el software a lo largo de su ciclo de vida, desde las primeras fases de desarrollo hasta la implementación y el mantenimiento, centrándose en el código, las configuraciones y el comportamiento en tiempo de ejecución para detectar amenazas como ataques de inyección, autenticación insegura, configuraciones erróneas o datos confidenciales expuestos, entre otros.

Las AST son esenciales para detectar vulnerabilidades en diferentes etapas del SDLC, prevenir brechas de seguridad, garantizar el cumplimiento normativo y reforzar la postura de seguridad general de una organización.

A continuación, se muestra un breve análisis de las principales categorías.

6.1.1 PRUEBAS ESTÁTICAS DE SEGURIDAD DE APLICACIONES (SAST)

Las herramientas SAST utilizan un enfoque de pruebas de caja blanca, en el que los evaluadores inspeccionan el funcionamiento interno de una aplicación. En un análisis SAST se inspecciona el código fuente estático e informa sobre las debilidades de seguridad, las cuales pueden aplicar al código en reposo (no desplegado) para encontrar problemas como mala validación de entradas, malas configuraciones de seguridad o uso de patrones de desarrollo inseguros.

Mediante métodos como el análisis del árbol AST (representación jerárquica de la sintaxis del código en forma de árbol cuyos nodos representan una construcción del lenguaje analizado y que obvia detalles redundantes), patrones de código, datos en el código o flujos de llamadas, una herramienta SAST examina una aplicación a nivel de función, archivo/clase y aplicación, y la profundidad de su análisis afecta directamente a la precisión de los resultados.

Existen distintos puntos en el ciclo de vida donde incluir los análisis SAST, desde la etapa más temprana dentro de los IDE de los desarrolladores hasta puntos más centralizados

como en los controles de seguridad integrados sobre las *pull requests*, en los procesos de integración continua.

a) **Ventajas:**

- No requiere una aplicación en ejecución.
- Cobertura de gran parte del código fuente.
- Fomenta hábitos de codificación seguros entre los desarrolladores.
- Permite integrar un análisis en una fase más reciente del ciclo de vida que otros análisis AST.

b) **Inconvenientes:**

- Puede generar una cantidad excesiva de falsos positivos.
- No prueba bibliotecas ni *frameworks* externos.
- No es un método de prueba en tiempo real.
- Dependiente de la tecnología usada

c) **Herramientas:**

- **SonarQube:** Plataforma *open source* ampliamente utilizada para la inspección continua de la calidad del código, pero con reglas específicas de seguridad que la permiten actuar como SAST.
- **Fortify Static Code Analyzer:** Herramienta comercial propiedad de OpenText que analiza el código fuente en busca de vulnerabilidades.
- **Checkmarx:** Herramienta comercial popular para el análisis estático de código.
- **Veracode Static Analysis:** Solución SAST comercial basada en la nube.
- **Kiuwan:** Otra herramienta SAST SaaS comercial.
- **Semgrep:** Una herramienta SAST rápida y *open source*, con licenciamiento comercial de pago.
- **Cppcheck:** Herramienta open source de análisis estático orientada principalmente a C y C++, útil para detectar defectos, malas prácticas y determinados problemas de seguridad en etapas tempranas del desarrollo.
- **Clang-Tidy:** Herramienta basada en LLVM/Clang para C y C++ que combina comprobaciones de calidad, modernización y seguridad, especialmente útil en integración continua y revisiones automáticas.

- **GitHubCodeQL:** Motor de análisis semántico de GitHub que trata el código como una base de datos. Permite realizar consultas (queries) para encontrar vulnerabilidades y errores lógicos complejos mediante el análisis del flujo de datos. Es gratuito para *Open Source* y de pago para empresas.
- **OWASP – SpotBugs + FindSecBugs:** SpotBugs es una herramienta de análisis estático para proyectos Java, y el complemento FindSecBugs añade reglas específicas de seguridad para la detección de vulnerabilidades en el código.

6.1.2 ANÁLISIS DE LA COMPOSICIÓN DEL SOFTWARE (SCA)

Las herramientas SCA ayudan a las organizaciones a analizar los componentes de terceros que se utilizan en su software, habitualmente aquellos que son *open source*. Estos análisis son realmente interesantes ya que las aplicaciones empresariales pueden utilizar miles de componentes de terceros, y que en la mayoría de casos pueden contener vulnerabilidades de seguridad.

Un análisis SCA ayuda a comprender qué componentes y versiones se están utilizando realmente, a identificar las vulnerabilidades de seguridad más graves que afectan a esos componentes y a comprender la forma más fácil de remediarlas. Esto permite a los equipos de desarrollo identificar y corregir rápidamente todos los problemas de seguridad y cumplimiento detectados que introducen los componentes utilizados para crear la aplicación.

Otra de sus grandes cualidades suele ser generar un SBOM (Software Bill of Materials), un documento que describe los componentes de software utilizados en un producto o sistema. Con esta práctica los equipos pueden detectar y catalogar las bibliotecas usadas en toda la organización, de tal forma que permita verificar que componentes están afectados al descubrirse una nueva vulnerabilidad crítica y poder actuar en consecuencia sobre esas aplicaciones.

a) Ventajas:

- Análisis exhaustivo de todos los activos de software de terceros.
- Advertencia de vulnerabilidades 0-day nada más ser descubiertas.
- Creación de un catálogo de componentes de software.
- Detección de problemas con las licencias de bibliotecas utilizadas.

b) Inconvenientes:

- La tasa de éxito depende de lo actualizadas que estén las fuentes y bases de datos en cuanto a las últimas vulnerabilidades.
- Dependiente de la tecnología usada

c) **Herramientas:**

- **Snyk:** Plataforma comercial utilizada para encontrar y corregir vulnerabilidades en dependencias de código abierto.
- **Black Duck:** Herramienta comercial especializada en la gestión de riesgos de código abierto y el cumplimiento de licencias.
- **WhiteSource:** Herramienta comercial de SCA que ayuda a las organizaciones a gestionar el riesgo de componentes de código abierto.
- **Dependency Track:** Herramienta de análisis de software que permite analizar dependencias de software y gestionar el catálogo de activos de terceros.
- **GitHubDependabot:** Funcionalidad integrada en GitHub que detecta automáticamente dependencias vulnerables en los proyectos y genera alertas o pull requests para actualizarlas a versiones seguras.
- **OWASP – Dependency-Check:** Herramienta open source que analiza las dependencias del proyecto y las compara con bases de datos públicas de vulnerabilidades (como NVD) para identificar componentes afectados.
- **JFrog – Xray:** Herramienta comercial que analiza dependencias, contenedores y artefactos en repositorios, identificando vulnerabilidades conocidas y problemas de licenciamiento en los componentes utilizados.

6.1.3 PRUEBAS DINÁMICAS DE SEGURIDAD DE APLICACIONES (DAST)

Las herramientas DAST adoptan un enfoque de pruebas de caja negra. Ejecutan pruebas sobre una aplicación desplegada y la inspeccionan en tiempo de ejecución, detectando problemas que pueden representar vulnerabilidades de seguridad. Esto puede incluir problemas de inyección, tales como SQL injection o Cross-site Scripting, debilidades en la autenticación y autorización, mala configuración en cabeceras, cookies o sesiones inseguras o fallos de memoria, entre otros.

a) **Ventajas:**

- Identifica vulnerabilidades en la capa externa de la aplicación, que pueden permitir la entrada de ataques maliciosos.
- Independiente del lenguaje y tecnología.
- Muchos menos falsos positivos que SAST.

b) Inconvenientes:

- Necesidad de un entorno en ejecución de prueba para garantizar que la aplicación original y sus datos estén protegidos contra ataques simulados.
- No es fácilmente escalable.
- Pruebas más lentas.
- La remediación es más costosa que para los análisis SAST, ya que se implanta en una fase del ciclo de vida posterior.

c) Herramientas:

- **OWASP ZAP (Zed Attack Proxy):** Herramienta de código abierto para pruebas de penetración y escaneo de vulnerabilidades.
- **BurpSuite:** Plataforma comercial para pruebas de seguridad de aplicaciones web.
- **Acunetix:** Herramienta comercial de escaneo de vulnerabilidades web.
- **Netsparker:** Herramienta comercial de DAST que automatiza el proceso de descubrimiento y verificación de vulnerabilidades.
- **Rapid7 – InsightAppSec:** Herramienta comercial de análisis dinámico de seguridad (DAST) que permite escanear aplicaciones web en ejecución para detectar vulnerabilidades como inyecciones, fallos de autenticación, configuraciones inseguras y otros riesgos explotables desde el exterior.

6.1.4 PRUEBAS INTERACTIVAS DE SEGURIDAD DE APLICACIONES (IAST)

Las herramientas IAST siguen generalmente un enfoque de instrumentación, incorporando agentes y sensores en una aplicación en ejecución. Mediante el uso de estos instrumentos, la herramienta IAST realiza una búsqueda continua y en tiempo real de vulnerabilidades mediante el examen de las interacciones de la aplicación con pruebas manuales, automatizadas, o una combinación de ambas.

Los análisis IAST proporcionan información valiosa sobre la causa raíz de las vulnerabilidades y las líneas específicas de código afectadas, lo que facilita mucho su corrección, a la vez que analizan el código fuente, el flujo de datos, la configuración y las bibliotecas de terceros, y son adecuadas para las pruebas de API. Algunas herramientas IAST también incorporan SCA para ampliar las capacidades de análisis de la herramienta a componentes y marcos de código abierto.

a) Ventajas:

- Prueba toda la aplicación, incluidas las bibliotecas y los *frameworks*.

- Muchos menos falsos positivos que SAST o DAST.
- Análisis en tiempo real.

b) Inconvenientes:

- Ralentiza el rendimiento de la aplicación.
- Menor soporte ante tecnologías y lenguajes.
- Mayor dificultad de implementación.

c) Herramientas:

- **Contrast Security – Contrast Assess**: Instrumentación en tiempo real y análisis detallado de vulnerabilidades.
- **HCL Technologies – AppScan IAST**: Combina pruebas dinámicas con visibilidad del código fuente.
- **Synopsys – Seeker (IAST)**: Detecta vulnerabilidades en aplicaciones web y móviles en ejecución.
- **Veracode – IAST**: Integración con pruebas manuales y automatizadas, con detección precisa de fallos.
- **Rapid7 – InsightAppSec (IAST features)**: Funcionalidades IAST añadidas a su plataforma de análisis dinámico, identificando vulnerabilidades mientras la aplicación se ejecuta.

6.1.5 AUTOPROTECCIÓN DE APLICACIONES EN TIEMPO DE EJECUCIÓN (RASP)

Las herramientas RASP evolucionaron a partir de SAST, DAST e IAST. Son capaces de analizar el tráfico de las aplicaciones y el comportamiento de los usuarios en tiempo de ejecución, para detectar y prevenir amenazas.

La protección RASP tiene visibilidad del código fuente de las aplicaciones y puede analizar sus debilidades y vulnerabilidades, identificando los fallos de seguridad que han sido explotados y proporcionar una protección activa mediante el bloqueo de la sesión o la emisión de una alerta.

Las herramientas RASP se integran con las aplicaciones y analizan el tráfico en tiempo de ejecución, y no solo pueden detectar y advertir sobre vulnerabilidades, sino que también previenen activamente los ataques.

a) Ventajas:

- Protección en tiempo real.
- Generación de alertas con el fin de detectar comportamientos inseguros.

b) Inconvenientes:

- Ralentiza el rendimiento de la aplicación.

- Menor soporte ante tecnologías y lenguajes.

c) **Herramientas:**

- **Contrast Security – Contrast Protect**: Protección activa en tiempo real con análisis profundo del código.
- **Imperva – RASP**: Prevención de ataques y monitorización continua del tráfico de la aplicación.
- **Micro Focus – Fortify Application Defender**: Integración con aplicaciones para bloquear amenazas en ejecución.
- **Prevoty (adquirida por Imperva)**: Detecta y previene ataques aprovechando el conocimiento del comportamiento de la aplicación.
- **F5 – Advanced WAF + RASP capabilities**: Combinación de firewall de aplicaciones con capacidades RASP para protección activa.

6.1.6 PRUEBAS DE SEGURIDAD DE APLICACIONES MÓVILES (MAST)

Las herramientas MAST (Mobile Application Security Testing) combinan análisis estáticos, dinámicos y forenses de datos generados por aplicaciones móviles. Permiten detectar vulnerabilidades de seguridad mediante técnicas como SAST, DAST e IAST, y además abordan problemas específicos de dispositivos móviles, como *rooting* o *jailbreaking*, conexiones a redes Wi-Fi maliciosas y fugas de datos desde dispositivos móviles.

a) **Ventajas:**

- **Cobertura completa** de seguridad móvil.
- Detecta **vulnerabilidades específicas** de plataformas móviles.
- **Integrable en DevSecOps** para procesos de desarrollo seguro.
- Facilita el **cumplimiento normativo**.
- Permite **simulación de ataques reales** en entornos controlados.
- Combinas múltiples **técnicas de análisis** para mayor efectividad.

b) **Inconvenientes:**

- **Coste elevado** de herramientas y licencias.
- **Curva de aprendizaje elevada** para su correcta implementación.
- Dependencia de **dispositivos físicos o emuladores** para pruebas precisas.
- Posibles **falsos positivos** que requieren análisis adicional.
- **Complejidad en integración** con entornos de desarrollo existentes.

c) **Herramientas:**

- **F5 – Advanced WAF + RASP capabilities**: Combina protección activa con análisis móvil.
- **Guardsquare – iXGuard / DexGuard**: Protección y ofuscación de aplicaciones móviles.
- **Synopsys – Coverity (Mobile)**: Análisis estático para detectar vulnerabilidades en apps móviles.
- **HCL Technologies – AppScan Mobile**: Pruebas completas de seguridad móvil, incluyendo DAST e IAST.
- **OWASP – MobSF (Mobile Security *Framework*)**: Herramienta de código abierto para análisis estático, dinámico y forense.
- **ProGuard**: Herramienta gratuita y de código abierto para Java y Android que ofusca y protege el código y tamaño de las aplicaciones.

6.2 PRÁCTICAS FUNDAMENTALES DE DESARROLLO SEGURO

El desarrollo seguro de software es un proceso que permite implementar código resiliente, lo cual requiere la aplicación sistemática de principios y prácticas que minimicen la introducción de vulnerabilidades durante todas las fases del ciclo de vida del desarrollo.

6.2.1 PRINCIPIOS DE DESARROLLO SEGURO

El desarrollo seguro constituye la primera línea de defensa contra las vulnerabilidades de software. Las buenas prácticas de implementación de código fuente pueden traducirse en requisitos de desarrollo sin necesidad de que el creador del software tenga un conocimiento profundo de vulnerabilidades y *exploits*. Los principios fundamentales incluyen las siguientes pautas:

- **Validación de entradas**: Validar todas las entradas mediante listas permitidas de caracteres, verificando tipo de dato, longitud y formato.
- **Codificación de salidas**: Codificar la salida de datos según contexto de destino para prevenir inyecciones.
- **Autenticación y sesiones**: Almacenamiento seguro de credenciales y implementación y gestión correcta de sesiones.
- **Control de acceso**: Verificar autorización basada en roles con principio de denegación por defecto.
- **Manejo de errores y *logging***: Evitar revelar información sensible y registrar y monitorizar eventos de seguridad.
- **Criptografía**: Usar algoritmos robustos con gestión adecuada de claves.
- **Protección de datos**: Cifrar datos en almacenamiento y en tránsito.

- **Seguridad de comunicaciones**: Implementar TLS estándar y filtrar parámetros confidenciales.
- **Protección contra vulnerabilidades comunes**: Prevenir amenazas de OWASP Top 10 y CWE Top 25.

6.2.2 RECURSOS Y ESTÁNDARES DE DESARROLLO SEGURO

Las organizaciones deben aprovechar recursos consolidados y estándares establecidos por la comunidad internacional de seguridad para implementar prácticas efectivas de desarrollo seguro.

6.2.2.1 OWASP DEVELOPER GUIDE

Guía de referencia fundamental que proporciona orientación comprensiva sobre desarrollo seguro de software [13], cubriendo desde arquitectura y diseño hasta implementación, pruebas y despliegue. Incluye mejores prácticas para tecnologías específicas, *frameworks* y lenguajes de programación, así como recomendaciones sobre herramientas de seguridad y su integración en el ciclo de vida del desarrollo. Esta guía sirve como recurso educativo principal para equipos de desarrollo que buscan incorporar seguridad desde las primeras fases del proyecto.

6.2.2.2 OWASP APPLICATION SECURITY VERIFICATION STANDARD (ASVS V5.0.0)

Estándar de verificación de seguridad de aplicaciones que define requisitos de seguridad organizados en categorías funcionales (autenticación, gestión de sesiones, control de acceso, validación de datos, criptografía, manejo de errores, entre otros) [15]. El estándar establece tres niveles de verificación:

- Nivel 1 (oportunista) para aplicaciones de bajo riesgo con verificaciones automatizadas básicas.
- Nivel 2 (estándar) para aplicaciones que manejan datos sensibles requiriendo controles de seguridad sólidos.
- Nivel 3 (avanzado) para aplicaciones críticas que requieren el máximo nivel de verificación incluyendo revisión manual exhaustiva.

ASVS proporciona criterios objetivos para evaluar la seguridad de aplicaciones web y puede utilizarse como base para definir requisitos de seguridad en contratos de desarrollo, establecer criterios de aceptación en proyectos, y guiar pruebas de penetración y auditorías de seguridad.

6.2.2.3 OWASP MOBILE APPLICATION SECURITY VERIFICATION STANDARD (MASVS)

Estándar especializado que define requisitos de seguridad específicos para aplicaciones móviles en plataformas iOS y Android [16]. Aborda aspectos únicos de seguridad móvil como almacenamiento seguro en el dispositivo, comunicación segura en redes móviles, autenticación y autorización en contextos móviles, criptografía apropiada para recursos limitados, protección contra ingeniería inversa y manipulación de aplicaciones, y gestión

segura de datos sensibles en entornos potencialmente comprometidos. MASVS complementa ASVS proporcionando guía específica para el ecosistema móvil.

6.2.2.4 OWASP TOP 10

Lista actualizada periódicamente que identifica los diez riesgos de seguridad más críticos [17] y usuales en aplicaciones web según datos empíricos de la industria. La versión actual, perteneciente a 2021, incluye riesgos como control de acceso deficiente, fallos criptográficos, inyección, diseño inseguro, configuración de seguridad incorrecta, componentes vulnerables y obsoletos, fallas de identificación y autenticación, fallas de integridad del software y datos, fallas de registro y monitorización, y falsificación de solicitudes del lado del servidor (SSRF). Cada riesgo incluye descripción detallada, ejemplos de escenarios de explotación, impacto potencial y recomendaciones específicas de prevención.

El OWASP Top 10 debe utilizarse para priorizar esfuerzos de seguridad, definir casos de prueba críticos, establecer formación de concienciación enfocada, y validar que las herramientas de análisis automatizado (SAST/DAST) detecten efectivamente estos riesgos prioritarios.

6.2.2.5 CWE TOP 25

Lista de las 25 debilidades de software más peligrosas publicada anualmente por MITRE, basada en datos reales de vulnerabilidades explotadas [18]. Incluye debilidades como desbordamiento de búfer, validación inadecuada de entrada, inyección SQL, *cross-site scripting* (XSS), deserialización insegura, uso de funciones criptográficas débiles, o control inadecuado de recursos. Para cada debilidad se proporcionan descripciones técnicas detalladas, ejemplos de código vulnerable, consecuencias potenciales, técnicas de detección y estrategias de mitigación específicas. CWE Top 25 complementa OWASP Top 10 proporcionando una perspectiva más técnica centrada en debilidades de implementación específicas.

Estos recursos deben integrarse en programas de formación continua del personal, estándares de codificación organizacionales, revisiones de código estructuradas, y verificaciones automatizadas del pipeline de CI/CD mediante configuración de herramientas SAST/DAST para detectar específicamente estos riesgos prioritarios. Su adopción sistemática reduce significativamente la introducción de vulnerabilidades comunes, mejora la calidad del software desarrollado.

Los recursos y estándares presentados en este capítulo proporcionan las herramientas técnicas fundamentales para el desarrollo seguro de software. Sin embargo, la efectividad de estas herramientas y prácticas depende críticamente del marco metodológico en el que se implementen.

7. INTEGRACIÓN DE DEVSECOPS EN EL SSDLC

7.1 INTRODUCCIÓN A DEVSECOPS EN EL SSDLC

Las prácticas y herramientas de seguridad descritas en los capítulos anteriores constituyen la base técnica del desarrollo seguro. No obstante, su eficacia aumenta significativamente cuando se integran de forma sistemática en el flujo de desarrollo, validación y despliegue del software. En este contexto, DevSecOps actúa como el modelo operativo que permite incorporar dichos controles de manera continua, automatizada y trazable dentro del SSDLC.

7.1.1 DEVOPS

DevOps es una cultura, filosofía y conjunto de prácticas que combina el desarrollo de software (*Development*) y las operaciones de TI (*Operations*), con el objetivo de acortar el ciclo de vida del desarrollo y proporcionar entrega continua de software de alta calidad.

DevOps no es una metodología de desarrollo como tal, sino que reúne las bases del desarrollo de software y las operaciones para permitir que las organizaciones sean ágiles y mantengan el ritmo de innovación. Para conseguirlo se basa fundamentalmente en la automatización, la colaboración entre equipos tradicionalmente separados, y la integración y despliegue continuos (CI/CD).

a) Ventajas de DevOps:

- Despliegues más rápidos y frecuentes mediante automatización
- Reducción del tiempo entre desarrollo y producción
- Mayor estabilidad ya que la construcción y despliegue de software se realiza de forma automática, sin necesidad de incorporar el factor humano manual.
- Detección rápida de incidentes con monitorización continua
- Mejora la colaboración entre equipos
- Automatización de pruebas mejora calidad del software

b) Inconvenientes de DevOps:

- Requiere cambio cultural profundo en la organización
- Inversión inicial considerable en herramientas y formación
- Complejidad en implementación de pipelines CI/CD
- Necesidad de equipos con habilidades multidisciplinares
- La velocidad de cambios puede introducir riesgos

c) Aplicabilidad:

- DevOps es una práctica apropiada para organizaciones que desarrollan y operan software continuamente, equipos que necesitan desplegar cambios frecuentemente, aplicaciones nativas de la nube, y organizaciones dispuestas a invertir en automatización y transformación cultural.

7.1.2 DEVSECOPS

DevSecOps es la evolución natural de DevOps que permite integrar la seguridad (*Security*) como componente fundamental y de primera clase del proceso desde el inicio, en lugar de tratarla como una fase separada al final del desarrollo. La metodología DevSecOps se refiere específicamente a integrar prácticas de seguridad desarrolladas por el equipo de seguridad dentro de la fase de integración y entrega continuas (CI/CD) y en las cadenas de herramientas utilizadas por desarrolladores y gestionadas por equipos de operaciones. Este enfoque implementa el principio de *Shift Left* en seguridad, incorporando prácticas, herramientas y controles de seguridad automatizados a lo largo de todas las fases del ciclo de vida del desarrollo de software [9].

a) Ventajas de DevSecOps:

- Integración de seguridad desde fases tempranas
- Detección temprana de vulnerabilidades mediante herramientas automatizadas
- Reducción de costes al resolver problemas antes de producción
- Responsabilidad compartida de seguridad entre todos los equipos
- Mayor velocidad sin comprometer la seguridad
- Cumplimiento continuo de requisitos regulatorios

b) Inconvenientes de DevSecOps:

- Requiere madurez previa en prácticas DevOps
- Necesidad de herramientas especializadas costosas (SAST, DAST, SCA)
- Curva de aprendizaje pronunciada en seguridad para desarrolladores
- Puede ralentizar inicialmente los despliegues
- Requiere cambio cultural hacia responsabilidad de seguridad
- Inversión continua en formación del personal
- Complejidad en configuración de múltiples herramientas de seguridad

c) Aplicabilidad:

- DevSecOps es más apropiado para organizaciones con prácticas DevOps maduras, desarrollo de software crítico o que maneja datos sensibles, organizaciones sujetas a regulaciones estrictas de seguridad, y equipos que despliegan cambios frecuentemente y necesitan asegurar cada despliegue.

7.1.3 CI/CD COMO HABILITADOR DE DEVSECOPS

En el contexto de DevSecOps, CI/CD constituye el mecanismo técnico que permite integrar controles de seguridad en el ciclo de vida del software de forma automatizada, continua y trazable. Más que como una definición general del concepto, en este apartado CI/CD se aborda desde su función como habilitador operativo para incorporar validaciones de seguridad, pruebas automáticas y mecanismos de aseguramiento dentro del SSDLC.[9][10]

La **integración continua (CI)** consiste en integrar los cambios del código en un repositorio de código fuente compartido de forma automática y frecuente, seguido de un proceso automatizado de construcción, pruebas y empaquetado. Este proceso incluye:

- Incorporación del código al repositorio de forma frecuente.
- Construcción automática del artefacto: descarga de dependencias y bibliotecas, compilación del código fuente (si aplica según la tecnología), y empaquetado del software.
- Ejecución de pruebas automatizadas: pruebas unitarias, pruebas de integración, pruebas funcionales y controles de calidad (QA).
- Generación y almacenamiento del artefacto versionado en un *registry* desde el cual puede distribuirse posteriormente.

La **distribución continua (CD)** abarca dos enfoques que toman el artefacto construido en CI y gestionan su despliegue:

- **Entrega continua (*Continuous Delivery*):** Prepara automáticamente los artefactos para el despliegue, realizando validaciones de seguridad y configuración en entornos de preproducción, pero requiere aprobación manual para el paso a producción.
- **Despliegue continuo (*Continuous Deployment*):** Automatiza completamente el proceso de despliegue sin intervención manual, liberando el artefacto directamente a producción una vez superadas todas las validaciones automatizadas.

En ambos enfoques, el CD prepara el artefacto según el entorno destino (configuraciones, secretos, variables de entorno…) y lo despliega en la infraestructura correspondiente (servidores, contenedores, clústeres de Kubernetes, servicios cloud, etc.).

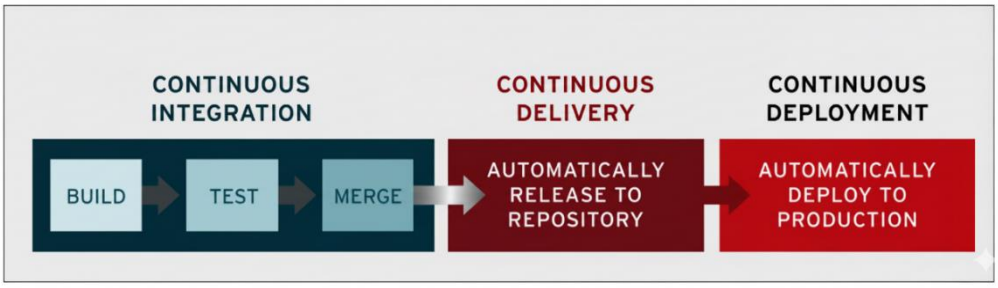


Figura 10. Flujo CI/CD

Las diferentes automatizaciones dentro del CI/CD se denominan *pipelines*, las cuales permiten integrar no tan solo las tareas propias de integración y despliegue continuos, si no controles de seguridad de forma automática y continua en cada etapa del flujo de trabajo. Esta integración se materializa mediante la incorporación de herramientas especializadas de seguridad como análisis estático de código (SAST), análisis dinámico (DAST), análisis de composición de software (SCA), y pruebas de seguridad automatizadas que se ejecutan en las diferentes fases del *pipeline*.

De esta manera, la metodología DevOps integrada con seguridad (conocido como DevSecOps, concepto que abordaremos en mayor profundidad más adelante) transforma los *pipelines* tradicionales de CI/CD en *pipelines* seguros donde la seguridad no es una fase final, sino un control transversal que detecta vulnerabilidades tempranamente aplica el principio de *shift left*.

7.1.3.1 HERRAMIENTAS DE CI/CD

La implementación y ejecución de los pipelines de CI/CD requiere el uso de herramientas especializadas que automatizan y orquestan las diferentes etapas del ciclo de vida. Entre las herramientas más utilizadas en la industria se encuentran:

- **Jenkins:** Servidor de automatización *open-source* ampliamente adoptado que permite construir, probar y desplegar software mediante *pipelines* configurables.
- **GitLab CI/CD:** Plataforma integrada que proporciona capacidades completas de CI/CD dentro del mismo entorno de gestión de código fuente.
- **Azure DevOps:** Suite de servicios de Microsoft que ofrece pipelines de CI/CD nativos para entornos cloud y *on-premise*.
- **ArgoCD:** Herramienta declarativa de entrega continua específicamente diseñada para entornos Kubernetes, siguiendo el paradigma de GitOps.

La selección de herramientas debe basarse en factores como la infraestructura existente, requisitos de seguridad, nivel de automatización deseado y compatibilidad con el *stack* tecnológico de la organización.

7.1.4 INTEGRACIÓN CONTINUA (CI)

La integración continua (CI) permite que los desarrolladores fusionen cambios de código de manera frecuente mediante validación automática a través de compilación y pruebas automatizadas. Desde la perspectiva de seguridad, el CI debería garantizar que cada cambio de código:

- Permita la detección temprana de vulnerabilidades mediante análisis estático de código (SAST) integrado en el proceso de construcción, identificando fallos de seguridad antes de que el código avance en el pipeline.
- No introduzca nuevas vulnerabilidades inadvertidas mediante la ejecución automática de pruebas de seguridad que validen el cumplimiento de estándares de desarrollo seguro y políticas organizacionales.
- Incluya verificación automática de dependencias y componentes de terceros mediante análisis de composición de software (SCA) para detectar vulnerabilidades conocidas en bibliotecas externas.
- Genere artefactos de construcción verificables con información de procedencia (SBOM) que permitan trazar las fuentes del código integrado.

La implementación de controles de seguridad en CI reduce significativamente el riesgo de introducir vulnerabilidades, disminuye los costes de remediación al detectar problemas en fases tempranas, y permite mantener una postura de seguridad consistente a lo largo del ciclo de vida del desarrollo.

7.1.5 ENTREGA CONTINUA (CD)

La entrega continua automatiza el proceso de preparación del código validado para su despliegue en entornos de preproducción y producción. En esta etapa, el artefacto generado durante la integración continua (CI) se somete a validaciones adicionales de seguridad y calidad antes de quedar listo para su liberación, requiriendo una aprobación manual final para el paso a producción. Las actividades de seguridad en la entrega continua incluyen:

- Pruebas de seguridad en entornos similares a producción mediante análisis dinámico de aplicaciones (DAST) y pruebas de penetración automatizadas.
- Validación de configuraciones de seguridad en la infraestructura para garantizar que el entorno de despliegue cumple con los requisitos de seguridad establecidos.
- Verificación de cumplimiento normativo mediante controles automatizados que validan políticas de seguridad y privacidad.

- Generación de artefactos de seguridad (SBOM, reportes de análisis de vulnerabilidades, evidencias de pruebas…) que documenten el estado de seguridad del software.

El objetivo es mantener una base de código siempre preparada para implementarse en producción de manera segura, garantizando que cada versión candidata ha superado todas las validaciones de seguridad requeridas antes de su liberación.

7.1.6 DESPLIEGUE CONTINUO (CONTINUOUS DEPLOYMENT)

El despliegue continuo automatiza completamente el proceso de liberación a producción, eliminando la necesidad de aprobación manual. Una vez que el código supera todas las pruebas y validaciones de seguridad en las etapas anteriores del pipeline, se despliega automáticamente en el entorno de producción. Esta automatización total requiere implementar controles de seguridad robustos y mecanismos de protección que garanticen la estabilidad y seguridad del software en producción:

- Dependencia total de la automatización de pruebas de seguridad efectivas que validen exhaustivamente el código antes del despliegue.
- Mecanismos de *rollback* automatizados y rápidos que permitan revertir cambios problemáticos de forma inmediata ante la detección de fallos o vulnerabilidades.
- Monitorización continua en tiempo real con capacidades de detección de anomalías, actividades sospechosas y comportamientos inesperados del software desplegado.
- Gestión automatizada, segura y auditable de secretos, credenciales y configuraciones sensibles, evitando la exposición de información crítica durante el despliegue.

El despliegue continuo representa el nivel más maduro de automatización en CI/CD, permitiendo lanzamientos frecuentes y rápidos, pero requiere una confianza total en los controles de seguridad automatizados del pipeline.

7.1.7 CI/CD Y DEVSECOPS

El proceso CI/CD constituye un pilar fundamental de DevSecOps [9][10][11], ya que integra la seguridad como práctica continua en cada etapa del desarrollo. El pipeline de CI/CD en DevSecOps debe:

- Hacer cumplir políticas de seguridad automáticamente mediante "*security gates*".
- Proporcionar visibilidad continua del estado de seguridad en cada etapa.
- Facilitar la colaboración entre equipos con visibilidad compartida de riesgos.

- Automatizar la generación de evidencias para cumplimiento normativo.

7.1.7.1 SECURITY GATES

Las *security gates* son puntos de control automatizados que bloquean el avance del CI o CD del código si no cumple los criterios de seguridad establecidos. Estos *gates* validan requisitos específicos en cada fase:

- **Análisis estático (SAST)**
- **Análisis de composición (SCA)**
- **Análisis dinámico (DAST)**
- **Validación de políticas de infraestructura**

Si alguna verificación falla según los umbrales definidos, el pipeline se detiene automáticamente, evitando que código inseguro avance hacia producción. Estos umbrales suelen estar relacionados con la severidad o posibilidad de explotación de las vulnerabilidades detectadas.

7.1.7.2 VISIBILIDAD Y AUTOMATIZACIÓN DE EVIDENCIAS

Los pipelines de DevSecOps permiten proporcionar visibilidad continua mediante *dashboards* o métricas que muestren las vulnerabilidades detectadas, su severidad y estado de remediación. Además, automatizan la generación de evidencias para cumplimiento normativo:

- *Software Bill of Materials* (SBOM) con inventario completo de componentes y dependencias.
- Reportes de análisis de vulnerabilidades (SAST, DAST, SCA…).
- Resultados de pruebas de seguridad y registros de aprobaciones.

Estas evidencias facilitan auditorías y cumplimiento con marcos regulatorios como ENS, RGPD y NIS2.

7.1.8 SEGURIDAD DE LA INFRAESTRUCTURA CI/CD

La seguridad del propio pipeline de CI/CD es fundamental [12], ya que su compromiso permitiría inyectar código malicioso en los agentes que ejecutan los *scripts*, afectando a todos los usuarios finales. Las medidas esenciales incluyen:

- Control de acceso estricto con autenticación multifactor y principio de mínimo privilegio.
- Protección de secretos mediante almacenamiento en sistemas especializados (nunca en código fuente).
- Auditoría completa con registro exhaustivo de actividades y protección contra manipulación.

- Validación de integridad mediante firma digital de artefactos y verificación de herramientas.
- Gestión de vulnerabilidades con actualización continua del ecosistema CI/CD.

7.2 DEVSECOPS EN SSDLC EN METODOLOGÍAS TRADICIONALES

En metodologías tradicionales, la adopción de prácticas DevSecOps no modifica necesariamente la estructura secuencial del ciclo de vida, pero sí permite reforzar cada fase mediante automatización, trazabilidad y controles de seguridad continuos. En este contexto, el valor de DevSecOps reside en adaptar sus principios a entornos con fases más delimitadas, incorporando mecanismos de verificación y aseguramiento desde la planificación hasta la operación. [9]

- **Principio Shift Left**: Operacionaliza la incorporación de seguridad desde fases tempranas en lugar de verificación tardía.
- **Automatización como habilitador**: La automatización de verificaciones de seguridad es esencial para que no se convierta en cuello de botella. Las herramientas automatizadas se integran en *pipelines* CI/CD ejecutándose con cada cambio de código.
- **Responsabilidad compartida**: En lugar de que la seguridad sea responsabilidad exclusiva de un equipo separado, DevSecOps distribuye esta responsabilidad a todos los miembros del equipo. Los especialistas en seguridad evolucionan hacia roles de habilitación proporcionando herramientas, formación y consultoría.

7.3 INTEGRACIÓN DE DEVSECOPS EN METODOLOGÍAS ÁGILES

DevSecOps integra sistemáticamente prácticas, herramientas y controles de seguridad en metodologías ágiles y pipelines DevOps, transformando la seguridad de fase separada a actividad continúa integrada en el flujo de trabajo normal [9][10]. Entre las prácticas de DevSecOps por fase del ciclo tenemos:



Figura 11. Prácticas de DevSecOps

7.4 AUTOMATIZACIÓN EN DEVSECOPS

La automatización constituye el habilitador fundamental que permite la integración efectiva del SSDLC en metodologías ágiles y DevSecOps. Sin automatización extensiva, las verificaciones de seguridad se convierten en cuellos de botella que impiden la entrega continua característica de los entornos de desarrollo modernos.

7.4.1 FUNDAMENTOS DE LA AUTOMATIZACIÓN EN ENTORNOS ÁGILES

- **Velocidad versus seguridad manual**: En metodologías tradicionales con ciclos de varios meses, las revisiones manuales de seguridad al final del ciclo resultan viables, aunque costosas. En entornos ágiles con *sprints* de 1-4 semanas y entregas continuas, las revisiones manuales crean retrasos inaceptables. Un equipo que despliega múltiples veces al día no puede esperar semanas por aprobaciones manuales de seguridad. La automatización resuelve esta tensión ejecutando verificaciones en minutos o segundos en lugar de días o semanas.
- **Consistencia y reproducibilidad**: Los humanos son inconsistentes ejecutando tareas repetitivas y cometen errores bajo presión de plazos ajustados. Las herramientas automatizadas ejecutan las mismas verificaciones exactamente de la misma manera cada vez, sin fatiga, sin sesgos, y sin atajos. Esta consistencia asegura que ningún *commit* elude verificaciones de seguridad, independientemente de la urgencia percibida o la carga de trabajo del equipo.

- **Escalabilidad organizacional**: Una organización con decenas o cientos de equipos desarrollando software simultáneamente no puede escalar equipos de seguridad centralizados proporcionalmente. La automatización permite que pequeños equipos de seguridad soporten grandes organizaciones de desarrollo mediante herramientas y plataformas que multiplican su efectividad.
- **Feedback inmediato**: Los desarrolladores son más productivos cuando reciben *feedback* sobre problemas inmediatamente mientras el contexto está fresco en su memoria. Las herramientas automatizadas proporcionan resultados en minutos, permitiendo corrección inmediata. El *feedback* diferido (horas o días después) interrumpe el flujo de trabajo y requiere costo cognitivo de recargar contexto.

7.4.2 PIPELINE DE SEGURIDAD AUTOMATIZADO

Un pipeline DevSecOps completamente automatizado integra verificaciones de seguridad en cada etapa del ciclo de vida del software [10][11], tales como:

- *Pre-commit* y *commit* local
- *Push* al repositorio y análisis continuo
- *Build* y compilación segura
- Construcción de artefactos y firma digital
- Análisis de contenedores e imágenes
- Despliegue a *staging* y análisis dinámico
- *Fuzzing* continuo
- Infraestructura como Código (IaC) y validación
- Verificación antes de producción
- Monitorización y respuesta automatizada en producción
- Gestión automatizada de vulnerabilidades

7.4.3 AUTOMATIZACIÓN DE RESPUESTA A INCIDENTES

La automatización no termina con prevención; también acelera la respuesta cuando se detectan incidentes de seguridad en producción.

- **Detección automatizada**: Correlación de eventos de múltiples fuentes (logs de aplicación, WAF (*Web Application Firewall*), red, sistema operativo) identificando patrones de ataque. *Machine learning* detecta anomalías que podrían indicar compromiso.
- **Respuesta automática**: Cuando se detecta actividad maliciosa confirmada, acciones automatizadas pueden ejecutarse inmediatamente: bloquear direcciones IP atacantes en WAF/*firewall*,

revocar *tokens* de acceso comprometidos, aislar recursos afectados en segmentos de red separados, escalar contenedores sospechosos para análisis forense, iniciar procedimientos de *backup* y recuperación.

- **Orquestación con SOAR**: Plataformas de *Security Orchestration, Automation and Response* (SOAR) como Splunk SOAR, Palo Alto Cortex XSOAR, o IBM Resilient automatizan *workflows* de respuesta a incidentes: recopilación automática de evidencia, enriquecimiento de alertas con *threat intelligence*, ejecución de *playbooks* de respuesta predefinidos, notificación a *stakeholders* apropiados, documentación automática para auditoría.

7.4.4 BENEFICIOS CUANTIFICABLES DE LA AUTOMATIZACIÓN

- **Reducción de tiempo de verificación**: Verificaciones manuales que tomarían días o semanas se completan en minutos. Esto permite múltiples despliegues diarios sin comprometer seguridad.
- **Aumento de cobertura**: Herramientas automatizadas pueden verificar 100% del código contra cientos o miles de reglas de seguridad, cobertura imposible de lograr manualmente.
- **Detección temprana**: Problemas detectados en fase de desarrollo cuestan 10-100 veces menos remediar que en producción. La automatización maximiza la detección temprana.
- **Consistencia de aplicación**: Políticas de seguridad se aplican consistentemente sin excepciones basadas en presiones de tiempo o decisiones subjetivas.
- **Liberación de recursos humanos**: Equipos de seguridad se enfocan en actividades de alto valor (arquitectura de seguridad, modelado de amenazas, respuesta a incidentes complejos) en lugar de verificaciones rutinarias.
- **Mejora continua mediante datos**: La automatización genera métricas detalladas sobre tipos de vulnerabilidades, tiempo de remediación, efectividad de controles, permitiendo decisiones basadas en datos para optimizar el programa de seguridad.

7.5 CULTURA Y CAPACITACIÓN

- **Formación continua**: Capacitación en principios de seguridad, vulnerabilidades específicas de tecnologías utilizadas, sesiones *hands-on*, formación *just-in-time* al detectar vulnerabilidades, y certificaciones como CSSLP.
- **Security Champions**: Programa establecido por OWASP que identifica y capacita a individuos en cada equipo de desarrollo que actúan como embajadores de seguridad dentro de sus equipos. Los *Security Champions*

reciben formación adicional especializada en seguridad de aplicaciones, participan activamente en comunidades de práctica de seguridad, sirven como punto de contacto entre el equipo de desarrollo y el equipo de seguridad central, promueven la adopción de mejores prácticas de seguridad en su equipo, participan en revisiones de código con enfoque de seguridad, representan a su equipo en foros de seguridad organizacionales, y ayudan a traducir requisitos de seguridad en implementaciones prácticas. Este modelo distribuido permite escalar las capacidades de seguridad en organizaciones grandes, reducir la brecha entre seguridad y desarrollo, y crear una red de expertos en seguridad distribuidos que elevan el nivel de seguridad de toda la organización sin depender exclusivamente del equipo de seguridad central.

- **Cultura de responsabilidad compartida**: Transformación donde la seguridad es prioridad de todos, requiriendo liderazgo que articule su importancia, métricas que incluyan resultados de seguridad, celebración de contribuciones, *blameless postmortems*, y tiempo dedicado para deuda técnica de seguridad.

7.6 DESAFÍOS Y MITIGACIÓN

- **Fatiga de alertas:** Ajustar configuraciones para reducir falsos positivos, establecer umbrales solo para severidad alta/crítica e implementar triaje automático.
- **Resistencia cultural:** Involucrar desarrolladores en selección de herramientas, demostrar valor con ejemplos concretos y proporcionar formación práctica.
- **Complejidad de herramientas:** Estandarizar en número limitado de herramientas, priorizar integraciones robustas y revisar regularmente el porfolio tecnológico.
- **Balance velocidad-seguridad:** Aplicar enfoque basado en riesgo, automatizar procesos críticos e implementar controles incrementalmente.
- **Gestión de deuda técnica:** Hacer visible la deuda mediante *dashboards*, establecer políticas de priorización y asignar capacidad fija a remediación.

7.7 BENEFICIOS DE LA INTEGRACIÓN

- Identificación temprana de vulnerabilidades mediante iteración rápida y retroalimentación continua, siendo mucho más económico corregirlas en fases iniciales que al final del desarrollo.
- Automatización de pruebas y procesos de seguridad que reduce la carga manual, mejora la eficiencia y mantiene la velocidad de desarrollo mediante la eliminación de esperas por revisiones manuales.

- Integración de la seguridad en el flujo de trabajo diario, promoviendo la colaboración entre desarrolladores y equipos de seguridad en lugar de tratarla como proceso aislado.
- Adaptabilidad a cambios que permite responder rápidamente a nuevas amenazas y vulnerabilidades, incorporando las últimas mejores prácticas de seguridad en cada iteración.
- Mejora continua de prácticas y herramientas de seguridad, elevando la conciencia organizacional y reduciendo significativamente la probabilidad de incidentes en producción.

7.8 INTEGRACIÓN DE DEVSECOPS EN OTRAS METODOLOGÍAS

Si bien Scrum, Kanban y XP son las metodologías ágiles más extendidas, los principios del SSDLC y DevSecOps pueden integrarse efectivamente en cualquier modelo de desarrollo. La selección de la metodología debe basarse en las características del proyecto, madurez del equipo, criticidad del sistema y requisitos organizacionales, pero independientemente del modelo elegido, las prácticas de seguridad deben incorporarse sistemáticamente.

- **Modelo en Cascada (Waterfall):** El modelo en cascada, caracterizado por su naturaleza secuencial y lineal, integra prácticas del SSDLC en cada fase: requisitos de seguridad y análisis de riesgos inicial en la fase de requisitos; modelado de amenazas y arquitectura de seguridad en el diseño; codificación segura y análisis SAST/SCA en la implementación; pruebas de seguridad exhaustivas mediante DAST y penetración en la fase de pruebas; y gestión de vulnerabilidades en el mantenimiento.

    - **Aplicabilidad de DevSecOps:** Aunque DevSecOps está optimizado para entregas frecuentes, sus principios de automatización de verificaciones de seguridad, responsabilidad compartida, y *shift left* son aplicables en cascada. Las herramientas automatizadas se ejecutan en cada fase, y los equipos de desarrollo, operaciones y seguridad colaboran desde el inicio del proyecto.

- **Modelo en Espiral:** El modelo en espiral combina desarrollo iterativo con análisis de riesgos explícito, siendo naturalmente compatible con el SSDLC dado que el análisis de riesgos constituye actividad central en cada ciclo de determinación de objetivos, análisis y evaluación de riesgos, desarrollo y pruebas, y planificación de la siguiente iteración.

    - **Aplicabilidad de DevSecOps:** La naturaleza iterativa del modelo espiral facilita la integración de prácticas DevSecOps. Las verificaciones automatizadas de seguridad se ejecutan en cada iteración, proporcionando *feedback* continuo. El énfasis en análisis de riesgos se complementa con herramientas automatizadas que identifican y priorizan vulnerabilidades. Los

artefactos de seguridad se generan en cada ciclo, proporcionando visibilidad continua de la postura de seguridad.

7.8.1 SELECCIÓN DE METODOLOGÍA Y ADAPTACIÓN DEL SSDLC

La selección de la metodología de desarrollo debe considerar múltiples factores: la naturaleza y complejidad del proyecto, la estabilidad o volatilidad de los requisitos, el tamaño y distribución del equipo, la criticidad del sistema desde perspectiva de seguridad, la madurez de la organización en prácticas ágiles y de seguridad, las restricciones regulatorias o contractuales, y el entorno tecnológico y operativo. El principio fundamental independientemente de la metodología seleccionada, los principios del SSDLC permanecen constantes:

- La seguridad debe integrarse desde las fases más tempranas (*shift left*).
- Las verificaciones de seguridad deben automatizarse en la máxima medida posible.
- La responsabilidad de seguridad debe distribuirse entre todos los miembros del equipo.
- Las prácticas deben basarse en análisis de riesgos priorizando controles según criticidad.
- Debe existir mejora continua mediante retrospectivas y análisis de incidentes. Lo que varía es la forma específica de implementar estos principios adaptándose a las características de cada metodología.

Los principios DevSecOps son aplicables universalmente porque se fundamentan en automatización, colaboración y seguridad continua, conceptos relevantes en cualquier modelo de desarrollo:

- Las herramientas de análisis automatizado (SAST, DAST, SCA) funcionan independientemente de la metodología.
- Los *pipelines* CI/CD pueden configurarse para ciclos largos o cortos.
- La cultura de responsabilidad compartida beneficia cualquier estructura organizacional. La diferencia radica en la frecuencia de ejecución de prácticas y el nivel de automatización requerido, siendo mayor en metodologías con entregas más frecuentes.

8. MÉTRICAS DEL CICLO DE VIDA DEL SSDLC

Las métricas en el Ciclo de Vida del Desarrollo de Software Seguro (SSDLC) permiten evaluar de forma objetiva la eficacia de las prácticas de seguridad integradas a lo largo del ciclo de vida, independientemente del modelo de desarrollo adoptado (metodologías tradicionales o ágiles).

Estas métricas tienen como objetivo principal proporcionar visibilidad continua sobre la postura de seguridad del software, facilitar la toma de decisiones basada en datos, priorizar esfuerzos de mejora y verificar la efectividad del principio de *shift left*, sin estar asociadas a modelos contractuales o acuerdos de nivel de servicio.

a) Métricas de proceso

- **Porcentaje de *commits* que superan las verificaciones de seguridad en el primer intento**, como indicador de madurez en desarrollo seguro.
- ***Mean Time to Remediate* (MTTR) de vulnerabilidades**, desde su detección hasta su corrección.
- **Distribución de vulnerabilidades por fase del ciclo de vida**, para evaluar la eficacia del enfoque *shift left*.
- **Cobertura de análisis de seguridad** (SAST, DAST, SCA, análisis de infraestructura como código), en relación con el código y los artefactos generados.

b) Métricas de resultado

- **Número de vulnerabilidades clasificadas por severidad**, y su evolución en el tiempo.
- **Tendencia de vulnerabilidades abiertas frente a cerradas**, como indicador de capacidad de remediación.
- **Incidentes de seguridad en entornos de producción**, asociados al software desarrollado.
- **Tiempo de exposición de vulnerabilidades**, desde su introducción hasta su mitigación.
- **Porcentaje de aplicaciones o servicios que cumplen las políticas de seguridad definidas**.

c) Métricas operativas y de eficiencia

- **Frecuencia de despliegues**, como indicador de madurez del ciclo de entrega.
- **Tiempo de ciclo desde cambio de código hasta despliegue**, incluyendo verificaciones de seguridad.
- **Tasa de cambios fallidos relacionados con aspectos de seguridad**.

- **Disponibilidad operativa de los servicios**, como métrica indirecta del impacto de la seguridad en la estabilidad del sistema.

d) Visualización y seguimiento

- ***Dashboards* de seguridad integrados**, accesibles para los distintos perfiles implicados (desarrollo, seguridad, operaciones y gestión).
- **Actualización continua de métricas**, permitiendo análisis en tiempo real y capacidad de profundizar en el detalle cuando sea necesario.
- **Uso de métricas como herramienta de mejora continua**, apoyando retrospectivas, revisiones periódicas y toma de decisiones basada en evidencia.

9. ABREVIACIONES

AST	Application Security Testing
BSIMM	Building Security In Maturity Model
CD	Continuous Delivery / Continuous Deployment
CI	Continuous Integration
CI/CD	Continuous Integration / Continuous Delivery
CSPM	Cloud Security Posture Management
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
ENS	Esquema Nacional de Seguridad
IAST	Interactive Application Security Testing
MAST	Mobile Application Security Testing
NIST	National Institute of Standards and Technology
OWASP	Open Worldwide Application Security Project
PCI-DSS	Payment Card Industry Data Security Standard
RASP	Runtime Application Self-Protection
RGPD	Reglamento General de Protección de Datos
SAMM	Software Assurance Maturity Model
SAST	Static Application Security Testing
SBOM	Software Bill of Materials
SCA	Software Composition Analysis
SDLC	Software Development Lifecycle
SDL	Security Development Lifecycle
SIEM	Security Information and Event Management
SSDF	Secure Software Development Framework
SSDLC	Secure Software Development Lifecycle
XP	Extreme Programming

ANEXO A. FUENTES CONSULTADAS

1. MARCOS DE REFERENCIA Y ESTÁNDARES FUNDAMENTALES

[1] OWASP Foundation, "OWASP Software Assurance Maturity Model (SAMM) v2," [Online]. Disponible: https://owaspsamm.org/model/

[2] M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development *Framework* (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities," NIST Special Publication 800-218, National Institute of Standards and Technology, 2022. [Online]. Disponible: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf

[3] Synopsys, "Building Security In Maturity Model (BSIMM14)," 2024. [Online]. Disponible: https://www.bsimm.com/

[4] Microsoft Security Response Center, "Microsoft Security Development Lifecycle," 2024. [Online]. Disponible: https://www.microsoft.com/en-us/securityengineering/sdl

[5] ISO/IEC, "ISO/IEC 12207 Software Life Cycle Processes," 1995. [Online]. Disponible: https://globalgbc.org/wp-content/uploads/2022/08/ISO-IEC12207.pdf

2. METODOLOGÍAS ÁGILES

[6] K. Beck et al., "Manifesto for Agile Software Development," Agile Alliance, 2001. [Online]. Disponible: https://agilemanifesto.org/

[7] K. Schwaber and J. Sutherland, "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game," Scrum.org, Nov. 2020. [Online]. Disponible: https://scrumguides.org/

[8] Scaled Agile, Inc., "SAFe 6.0 *Framework*," 2023. [Online]. Disponible: https://scaledagile*framework*.com/

3. DEVSECOPS Y CI/CD

[9] NIST National Cybersecurity Center of Excellence, "Secure Software Development, Security, and Operations (DevSecOps) Practices," NIST Special Publication 1800-44A (Initial Public Draft), 2025. [Online]. Disponible: https://www.nccoe.nist.gov/sites/default/files/2025-07/nist-sp-1800-44a-ipd.pdf

[10] R. Chandramouli, F. Kautz, and S. Torres-Arias, "Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines," NIST Special Publication 800-204D, National Institute of Standards and Technology, 2024. [Online]. Disponible: https://csrc.nist.gov/pubs/sp/800/204/d/final

[11] R. Chandramouli, "Implementation of DevSecOps for a Microservices-based Application with Service Mesh," NIST Special Publication 800-204C, National Institute of Standards and Technology, 2022. [Online]. Disponible: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204C.pdf

[12] National Security Agency and Cybersecurity and Infrastructure Security Agency, "Defending Continuous Integration/Continuous Delivery (CI/CD) Environments," Cybersecurity Information Sheet, Jun. 2023. [Online]. Disponible: https://media.defense.gov/2023/Jun/28/2003249466/-1/-1/0/CSI_DEFENDING_CI_CD_ENVIRONMENTS.PDF

4. RECURSOS OWASP DE DESARROLLO SEGURO

[13] OWASP Foundation, "OWASP Developer Guide," 2024. [Online]. Disponible: https://owasp.org/www-project-developer-guide/

[14] OWASP Foundation, "OWASP Secure Coding Practices-Quick Reference Guide," 2024. [Online]. Disponible: https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

[15] OWASP Foundation, "OWASP Application Security Verification Standard (ASVS) v5.0.0," 2025. [Online]. Disponible: https://owasp.org/www-project-application-security-verification-standard/

[16] OWASP Foundation, "OWASP Mobile Application Security Verification Standard (MASVS)," 2024. [Online]. Disponible: https://mas.owasp.org/MASVS/

[17] OWASP Foundation, "OWASP Top 10 - 2021," 2021. [Online]. Disponible: https://owasp.org/Top10/

[18] MITRE Corporation, "CWE Top 25 Most Dangerous Software Weaknesses," 2024. [Online]. Disponible: https://cwe.mitre.org/top25/

5. NORMATIVA Y REGULACIÓN

[19] "Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad," Boletín Oficial del Estado, 2022. Disponible: https://www.boe.es/buscar/doc.php?id=BOE-A-2023-8109

PUBLICACIONES DE DEFENSA

GOBIERNO DE ESPAÑA
MINISTERIO DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA
SUBDIRECCIÓN GENERAL DE PUBLICACIONES Y PATRIMONIO CULTURAL