CAIGG
CONSEJO DE AUDITORÍA INTERNA GENERAL DE GOBIERNO
Ministerio Secretaría General de la Presidencia

Guía de Auditoría, Seguridad de la Información y Ciberseguridad N°8

DESARROLLO SEGURO DE SOFTWARE

ÍNDICE

Nota

PRESENTACIÓN

En cumplimiento con las instrucciones del Presidente de la República, Gabriel Boric Font, sobre fortalecimiento de la Política de Auditoría Interna de Gobierno; el Consejo de Auditoría Interna General de Gobierno, entidad asesora en materias de auditoría interna, control interno, probidad, gestión de riesgos y gobernanza del Supremo Gobierno, presenta a la Red de Auditoría Gubernamental, la GASIC N°8: Desarrollo Seguro de Software.

Esta guía es parte de una iniciativa del Consejo de Auditoría Interna General de Gobierno (CAIGG) que busca fortalecer la posición del sector público en materias de Seguridad de la Información y Ciberseguridad, dotando de instrumentos a los Auditores Internos y Servicios Públicos de instrumentos y herramientas que permitan desarrollar un levantamiento de información en base a las mejores prácticas y la legislación vigente.

Santiago, Mayo 2024.

Daniela Caldana Fulss
Auditora General de Gobierno

Capítulo 1

INTRODUCCIÓN AL DESARROLLO SEGURO DE SOFTWARE

En el contexto actual, donde la ciberseguridad se ha convertido en una preocupación primordial, el desarrollo seguro de software emerge como una disciplina crítica para proteger los sistemas y datos sensibles de las organizaciones gubernamentales. La creciente sofisticación de las amenazas cibernéticas y la dependencia de sistemas digitales para operaciones críticas requieren un enfoque riguroso y sistemático para asegurar que el software desarrollado y adquirido sea intrínsecamente seguro.

El desarrollo seguro de software no es un proceso aislado, sino que debe integrarse a lo largo de todo el ciclo de vida del desarrollo, desde la concepción y planificación hasta el despliegue y mantenimiento. Este enfoque integral garantiza que cada etapa del proceso de desarrollo considere y mitigue los riesgos de seguridad, promoviendo la resiliencia y la confianza en los sistemas de información

Nota Importante

Estrictamente hablando, **Seguridad de la Información y Ciberseguridad** son dos conceptos diferentes.

La **"Seguridad de la Información"** es la preservación de la Confidencialidad, Integridad y Disponibilidad de la Información en los activos de información, en cualquier medio (incluso, las personas); por otro lado **"Ciberseguridad"** hace referencia exclusiva al ciberespacio y activos digitales.

En esta guía adoptamos el concepto de "Seguridad de la Información y Ciberseguridad", pero para evitar la redundancia y el exceso de texto, utilizaremos los conceptos de "Seguridad de la Información", "Ciberseguridad" o el acrónimo "SIC" como sinónimos para mejorar la comprensión lectora.

El Ciclo de Desarrollo de Software: Un Análisis Detallado

El ciclo de desarrollo de software, también conocido como ciclo de vida del desarrollo de software (SDLC, por sus siglas en inglés), es un marco que define las fases que un proyecto de software sigue desde su inicio hasta su finalización. Este ciclo es esencial para la creación de software de alta calidad que cumpla con los requisitos del cliente y sea seguro, eficiente y mantenible. A continuación, se presenta un análisis detallado de las fases del SDLC, sus metodologías, y la importancia de cada una en el contexto del desarrollo seguro de software.

Fases del Ciclo de Desarrollo de Software

El SDLC se compone de varias fases clave, cada una de las cuales desempeña un papel crucial en el desarrollo del software. Aunque los detalles pueden variar dependiendo de la metodología específica utilizada, las fases comunes incluyen:

1. **Planificación y Análisis de Requisitos**
2. **Diseño del Sistema**
3. **Desarrollo y Codificación**
4. **Pruebas**
5. **Despliegue**
6. **Mantenimiento y Soporte**

1. Planificación y Análisis de Requisitos

La fase de planificación es la etapa inicial del SDLC, donde se define el alcance del proyecto y se identifican los objetivos principales. Durante esta fase, se lleva a cabo un análisis detallado de los requisitos del cliente y del sistema, incluyendo requisitos funcionales y no funcionales. Es fundamental involucrar a todas las partes interesadas para asegurar que se entienden claramente las expectativas y restricciones del proyecto.

En el contexto del desarrollo seguro de software, esta fase debe incluir la identificación de requisitos de seguridad específicos, tales como la protección de datos, la gestión de identidades y accesos, y la conformidad con normativas y estándares de seguridad.

2. Diseño del Sistema

En la fase de diseño, se crea una arquitectura detallada del sistema que cumple con los requisitos definidos. Esta arquitectura abarca tanto el diseño del software como el hardware necesario para soportarlo. Se desarrollan diagramas de flujo, modelos de datos y otros documentos técnicos que guían la implementación del sistema.

El diseño seguro del software debe considerar principios como la defensa en profundidad, el principio del mínimo privilegio y la separación de funciones. También es crucial definir controles de seguridad que se integren en la arquitectura del sistema.

3. Desarrollo y Codificación

La fase de desarrollo es donde se escribe el código del software siguiendo el diseño establecido. Los desarrolladores traducen los diagramas y especificaciones técnicas en un lenguaje de programación específico. Es durante esta fase que se construye el núcleo funcional del software.

Para garantizar la seguridad del software, es esencial seguir prácticas de codificación segura, como la validación de entradas, la gestión adecuada de errores y excepciones, y la utilización de bibliotecas y componentes seguros. La revisión de código y la aplicación de análisis estáticos y dinámicos son prácticas recomendadas para identificar y mitigar vulnerabilidades durante esta fase.

4. Pruebas

La fase de pruebas es crítica para asegurar que el software funciona según lo previsto y que cumple con los requisitos especificados. Las pruebas pueden incluir pruebas unitarias, pruebas de integración, pruebas del sistema y pruebas de aceptación del usuario. Cada tipo de prueba tiene como objetivo detectar errores y garantizar que el software es robusto y fiable.

Las pruebas de seguridad, como las pruebas de penetración y los análisis de vulnerabilidades, son componentes esenciales de esta fase. Estas pruebas ayudan a identificar posibles fallos de seguridad antes de que el software sea desplegado en un entorno de producción.

5. Despliegue

Una vez que el software ha pasado todas las pruebas y ha sido aprobado, se despliega en el entorno de producción. La fase de despliegue puede implicar la instalación del software en los servidores, la configuración del entorno y la capacitación de los usuarios finales.

El despliegue seguro del software debe incluir la implementación de controles de acceso, la configuración segura del entorno y la preparación de planes de respuesta ante incidentes. Es fundamental asegurarse de que todas las dependencias y componentes del sistema estén actualizados y configurados adecuadamente.

6. Mantenimiento y Soporte

El mantenimiento es la fase final del SDLC, pero no por ello menos importante. Incluye la corrección de errores que no fueron detectados durante las pruebas, la implementación de nuevas funcionalidades y la actualización del software para adaptarse a cambios en el entorno o en los requisitos del usuario.

El mantenimiento seguro del software requiere una gestión continua de vulnerabilidades, la aplicación regular de parches de seguridad y la realización de auditorías de seguridad periódicas. También es esencial monitorear el software en busca de comportamientos anómalos y responder rápidamente a cualquier incidente de seguridad.

Capítulo 2

METODOLOGÍAS DEL CICLO DE DESARROLLO DE SOFTWARE

2. METODOLOGÍAS DEL CICLO DE DESARROLLO DE SOFTWARE

Existen varias metodologías que guían el SDLC, cada una con sus propias ventajas y desventajas. Las más comunes incluyen:

Cascada (Waterfall)

La metodología en cascada es un enfoque secuencial donde cada fase del SDLC debe completarse antes de pasar a la siguiente. Aunque es fácil de entender y gestionar, puede ser rígida y no permite cambios fáciles una vez que una fase ha concluido.

Agile

Agile es una metodología iterativa y flexible que promueve el desarrollo incremental y la colaboración constante con el cliente. Las fases del SDLC se repiten en ciclos cortos llamados "sprints", lo que permite adaptarse rápidamente a los cambios en los requisitos.

DevOps

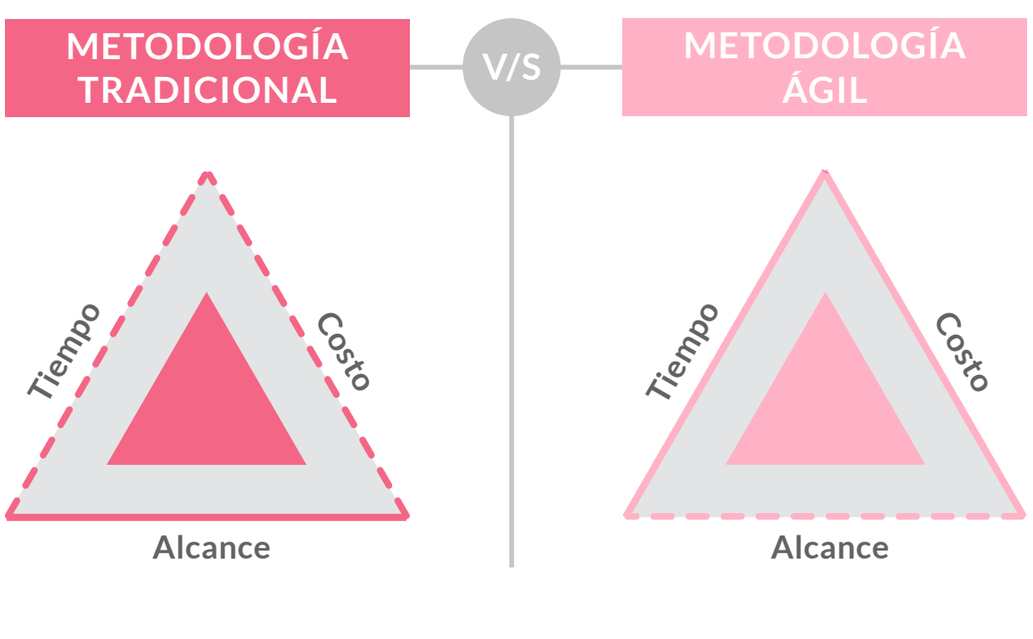
DevOps es una práctica que combina el desarrollo de software (Dev) y las operaciones de TI (Ops) para mejorar la colaboración y la eficiencia. DevOps enfatiza la automatización y la integración continua, lo que permite una entrega de software más rápida y fiable.

Metodología	*Descripción*	*Ventajas*	*Desventajas*
Cascada (Waterfall)	Enfoque secuencial donde cada fase debe completarse antes de pasar a la siguiente.	Claridad y estructura. Documentación extensa; Facilidad de gestión	Rigidez; Riesgo de problemas tardíos; Poca flexibilidad
Agile	Metodología iterativa y flexible con ciclos cortos llamados ‘Sprints’.	Flexibilidad; Colaboración y comunicación; Entrega continua.	Falta de documentación formal; Requiere disciplina; Incertidumbre.
DevOps	Práctica que combina desarrollo y operaciones para mejorar la colaboración y eficiencia.	Entrega rápida y continua; Mejora la colaboración; Alta fiabilidad.	Complejidad inicial; Cambio cultural; Requiere alta automatización.

Nota Importante | Existen otras metodologías de desarrollo de software, pero no es el alcance de esta guía su análisis.

Características

- Rigidez
- Negociación de Contrato
- Procesos y Herramientas
- Documentación Comprensiva
- Seguimiento al Plan
- Procesos muy Controlados
- Grupos de Gran Tamaño



Características

- Flexibilidad
- Colaboración del Cliente
- Personas e Interacción
- Software Funcionando
- Adaptarse a los Cambios
- Procesos menos Controlados
- Grupos Pequeños In-situ

Capítulo 3

DEVOPS: UN ENFOQUE DETALLADO EN EL CICLO DE DESARROLLO DE SOFTWARE

3. DEVOPS: ENFOQUE DETALLADO DEL CICLO DE DESARROLLO DE SOFTWARE

En el dinámico panorama tecnológico actual, DevOps ha emergido como una práctica revolucionaria que integra el desarrollo de software (Dev) y las operaciones de TI (Ops). Este enfoque tiene como objetivo mejorar la colaboración, la eficiencia y la velocidad de entrega mediante la automatización y la integración continua.

Principios Fundamentales de DevOps

DevOps se basa en varios principios clave que transforman la manera en que se desarrollan y gestionan las aplicaciones:

01 **Colaboración y Comunicación**
La esencia de DevOps radica en la estrecha colaboración entre los equipos de desarrollo y operaciones. Esta metodología rompe los silos tradicionales, promoviendo una comunicación fluida y alineación de objetivos.

02 **Automatización**
La automatización es el pilar central de DevOps, permitiendo acelerar los procesos de desarrollo, pruebas y despliegue. Las herramientas de integración y entrega continua (CI/CD) automatizan la construcción, prueba y despliegue del software, garantizando una entrega rápida y eficiente.

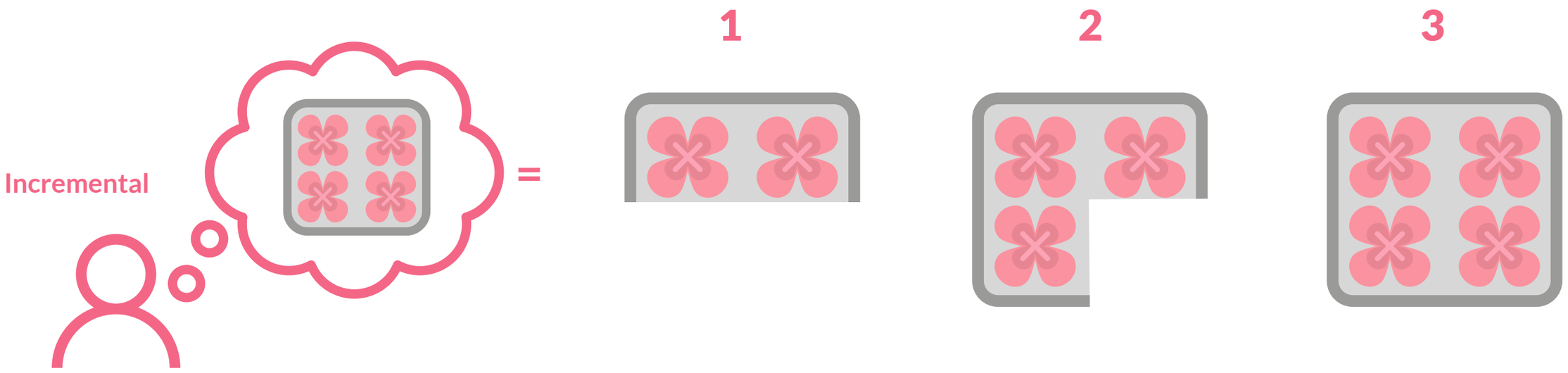
03 **Integración Continua (CI)**
La integración continua implica la integración frecuente del código de todos los desarrolladores en un repositorio compartido. Cada integración se verifica automáticamente mediante pruebas unitarias y de integración, asegurando que el nuevo código no rompa la funcionalidad existente.

04 **Entrega Continua (CD)**
La entrega continua extiende la integración continua al despliegue automático de cada cambio que pasa las pruebas en un entorno de producción. Esto permite que el software sea entregado a los usuarios finales de manera rápida y frecuente, con mínimas interrupciones.

05 **Monitoreo y Registro Continuo**
El monitoreo continuo de aplicaciones e infraestructura permite la detección proactiva de problemas. Los registros detallados ayudan a identificar y resolver problemas rápidamente, mejorando la estabilidad y el rendimiento del software.

06 **Infraestructura como Código (IaC)**
IaC permite gestionar y aprovisionar la infraestructura mediante scripts de configuración automatizados, en lugar de procesos manuales.

Desarrollo Iterativo vs Incremental



Beneficios de DevOps

La adopción de DevOps ofrece múltiples beneficios que transforman positivamente el ciclo de desarrollo de software:

1. Velocidad y Agilidad

DevOps permite ciclos de desarrollo más cortos y tiempos de entrega más rápidos, facilitando la respuesta ágil a las necesidades del mercado y del cliente. La capacidad de implementar cambios rápidamente reduce el tiempo de comercialización y mejora la competitividad.

2. Calidad y Fiabilidad

Las prácticas de pruebas automatizadas y el monitoreo continuo aseguran que el software sea robusto y fiable. La integración continua detecta errores en las primeras etapas del desarrollo, reduciendo los defectos en la producción.

3. Escalabilidad y Consistencia

IaC y la automatización aseguran que la infraestructura sea escalable y consistente, facilitando la gestión de entornos grandes y complejos. Los entornos reproducibles eliminan los problemas relacionados con configuraciones inconsistentes.

4. Colaboración y Responsabilidad

DevOps mejora la colaboración entre equipos, promoviendo una cultura de responsabilidad compartida por la calidad y el éxito del producto. Las prácticas colaborativas y la transparencia en los procesos fomentan un sentido de propiedad y compromiso entre los miembros del equipo.

Herramientas y Tecnologías DevOps

Diversas herramientas facilitan la implementación de DevOps, cada una con un propósito específico:

1. Control de Versiones

- Git: Sistema de control de versiones distribuido que facilita la gestión del código fuente y la colaboración entre desarrolladores.
- Subversion (SVN): Alternativa a Git, más tradicional y centralizado.

2. Integración y Entrega Continua

- Jenkins: Herramienta de automatización de código abierto que soporta CI/CD.
- Travis CI: Servicio de CI basado en la nube para proyectos alojados en GitHub.
- CircleCI: Plataforma de CI/CD que automatiza la construcción, prueba y despliegue de software.

3. Gestión de Configuración e Infraestructura como Código

- Terraform: Herramienta de IaC que permite la provisión de infraestructura en múltiples proveedores de servicios en la nube.
- Ansible: Plataforma de automatización de TI que gestiona la configuración y el despliegue de aplicaciones.
- Chef/Puppet: Herramientas de gestión de configuración que automatizan la configuración y administración de servidores.

4. Monitoreo y Registro

- Prometheus: Sistema de monitoreo y alerta diseñado para la fiabilidad y la escalabilidad.
- ELK Stack (Elasticsearch, Logstash, Kibana): Conjunto de herramientas para búsqueda, análisis y visualización de datos generados por máquinas en tiempo real.
- Grafana: Plataforma de código abierto para la visualización y análisis de datos.

Capítulo 4

4. MARCO NORMATIVO Y ESTÁNDARES DE REFERENCIA

4. MARCO NORMATIVO Y ESTÁNDARES DE REFERENCIA

Para guiar este proceso, existen múltiples estándares y marcos de referencia que proporcionan directrices y mejores prácticas. Entre los más reconocidos se encuentran ISO/IEC 27034, OWASP Software Assurance Maturity Model (SAMM), y el NIST Special Publication 800-218, también conocido como Secure Software Development Framework (SSDF).

ISO/IEC 27034

La norma ISO/IEC 27034 proporciona un marco comprensivo para integrar la seguridad en el proceso de desarrollo de software. Este estándar define un conjunto de prácticas organizativas, documentales y técnicas que ayudan a las organizaciones a asegurar que sus aplicaciones de software son desarrolladas con la seguridad en mente. La ISO/IEC 27034 enfatiza la importancia de considerar la seguridad desde las etapas iniciales del desarrollo, estableciendo controles y medidas que deben ser implementadas de manera continua.

OWASP SAMM

El OWASP Software Assurance Maturity Model (SAMM) es un modelo de madurez que permite a las organizaciones evaluar y mejorar sus prácticas de desarrollo de software seguro. SAMM proporciona un enfoque estructurado para medir el estado actual de las prácticas de seguridad del software y planificar mejoras. El modelo se organiza en varias categorías, incluyendo gobernanza, diseño, implementación, verificación y despliegue, cada una con actividades específicas que promueven la seguridad en el desarrollo.

NIST SP 800-218 (SSDF)

El NIST Special Publication 800-218, también conocido como Secure Software Development Framework (SSDF), ofrece un conjunto de prácticas de desarrollo seguro que las organizaciones pueden integrar en sus procesos de desarrollo de software. El SSDF proporciona directrices detalladas sobre cómo gestionar la seguridad a lo largo del ciclo de vida del desarrollo de software, incluyendo la planificación, implementación, verificación y respuesta a incidentes. Este marco es particularmente relevante para las organizaciones gubernamentales, ya que se alinea con las políticas y requisitos de seguridad federales.

Relevancia para los Auditores Gubernamentales

Para los auditores gubernamentales, la comprensión y evaluación de las prácticas de desarrollo seguro de software es esencial para garantizar la protección de los sistemas y datos críticos. La aplicación de estos estándares y marcos de referencia permite a los auditores verificar que las organizaciones están adoptando medidas adecuadas para mitigar los riesgos de seguridad. Además, proporciona un lenguaje común y un conjunto de expectativas que facilitan la comunicación y la colaboración entre equipos de desarrollo, seguridad y auditoría.

Capítulo 5

IMPORTANCIA DE LA SEGURIDAD EN EL SDLC

5. IMPORTANCIA DE LA SEGURIDAD EN EL SDLC

La integración de la seguridad en cada fase del SDLC es esencial para desarrollar software robusto y resistente a ataques. Para esta guía, considerando las fuentes seleccionadas, se han definido un conjunto de temas que no son excluyentes de nuevas tendencias o futuras actualizaciones. Estos temas son:

Planeación y Definición de Requisitos
Definición del Proceso, Definición de Requisitos de Seguridad, Funciones y Responsabilidades.

Herramientas y Entornos de Desarrollo
Cadenas de Soporte, Implementación de Entornos Seguros, Arquitectura de Software: Construcción, Despliegue y Seguimiento.

Pruebas, Verificación y Mantenimiento
Pruebas y Verificación, Revisión y Evaluación Continua de Vulnerabilidades, Funcionalidad de Software Existente.

Codificación y Protección de Software
Codificación Segura, Protección del Código y Software, Seguridad del Software.

Cada uno de estos temas está desarrollado en la Matriz de Controles de esta Guía, y su principal contenido se describe a continuación:

Fases SDLC	*Requrimiento*	*Diseño*	*Programación*	*Pruebas e Integración*	*Lanzamiento*
Tareas Hechas	Comienzo	Identificar los Activos de Diseño	Elegir un Lenguaje de Programación	Iniciar Pruebas	Mantenimiento Correctivo
	Obtención	Abstraer Especificaciones	Clasificaciones de Módulo	Pruebas del Sistema	Mantenimiento Adaptativo
	Elaboración	Diseño de Componentes	Elegir Herramientas de Programación	Pruebas de Aceptación	Mantenimiento de Perfeccionamiento
	Negociación	Diseño de Interface	Considerar las Opciones de Reutilización	Pruebas de Integración	Mantenimiento Preventivo
	Especificaciones	Diseño de Base de Datos			
	Validación				
	Gestión				
Posibles Problemas Seguridad	• Comprensión Compartida de los Requisitos. • Obtención de Requisitos de Seguridad. • Falta de Defensa en Profundidad. • Falta de Conciencia de Seguridad.	• Establecer Requisitos Seguridad del Diseño • Evaluar Riesgos de los Componentes de Terceros • Trazabilidad • Control de Accesos • Falta de Defensa en Profundidad. • Falta de Conciencia de Seguridad. • Defectos de Diseño	• Desbordamiento del Buffer • Falla de Inyección de Código • Falta de Uso • Prácticas de Codificación Segura • Falta de Seguridad • Conciencia • Evaluar Riesgos de los Componentes de Terceros	• Selección de Herramientas • Utilizar Múltiples Enfoques • Aceptación y Resistencia • Problemas de Cumplimiento • Presupuesto y Tiempo • Restricciones • Riesgos Técnicos	• Posibilidad de Mala Configuración • Posibilidad de Defectos

	Requrimiento	Diseño	Programación	Pruebas e Integración	Lanzamiento
Prácticas Mitigación	• Todas las Partes Interesadas Deben Estar de Acuerdo con las Definiciones de Requisitos • Identificar Activos Vulnerables y Críticos • Identificar las Dependencias de Requisitos • Identificar Amenazas • Desarrollar los Artefactos Correspondientes • Obtener Requisitos de Seguridad • Realizar Requisitos Priorización y Clasificación • Realizar Requisitos de Inspección • Actualizar el Repositorio de Requerimientos	• Economía de Mecanismo • Falso - Seguro por Defecto • Control de Accesos • Mínimos Privilegios • Mecanismos Menos Comunes • Aceptabilidad Psicológica • Defensa en Profundidad • Revisión de Diseño	• Codificación Segura • OSWAP Prácticas de Codificación Segura y Lista de Verificación • OSWAP Prácticas de Codificación General • Programación de Pares	• Pruebas e Integración Seguras • Los Casos de Prueba Deben ser Generados en Base a la Salida de la Fase RE • Pruebas Funcionales • Pruebas No Funcionales • Pruebas de Integración	• Proceso de Gestión de Cambios de Documentos • Seguir el Proceso de Gestión de Cambios • Planificar Recursos de Apoyo

Ilustración 1 Esquema General de Identificacion de Requisitos

5.1 PLANEACIÓN Y DEFINICIÓN DE REQUISITOS

Definición del Proceso

El proceso de desarrollo de software comienza con una planificación meticulosa. Esta fase inicial es crucial, ya que establece el rumbo del proyecto. Aquí, se delinean los objetivos, se identifican los recursos necesarios y se elaboran los cronogramas. Una planificación efectiva no solo se enfoca en los aspectos técnicos, sino que también considera las necesidades y expectativas del cliente. Se trata de una fase donde la comunicación abierta y clara con todas las partes interesadas es fundamental para asegurar que todos los involucrados comprendan el alcance y los objetivos del proyecto.

Definición de Requisitos de Seguridad

Definir los requisitos de seguridad desde el principio es esencial para proteger los datos y las operaciones de los usuarios finales. Estos requisitos deben ser específicos y detallados, abordando aspectos como la protección de datos sensibles, la autenticación de usuarios y la resistencia a ataques externos. La inclusión de expertos en seguridad durante esta fase es vital para identificar posibles vulnerabilidades y desarrollar estrategias para mitigarlas. Este enfoque proactivo ayuda a prevenir problemas de seguridad que podrían surgir más adelante en el ciclo de desarrollo.

Funciones y Responsabilidades

Asignar roles y responsabilidades claras dentro del equipo de desarrollo es esencial para el éxito del proyecto. Cada miembro del equipo debe conocer sus responsabilidades y entender cómo su trabajo contribuye al objetivo final. Esto incluye no solo a los desarrolladores, sino también a los gerentes de proyecto, analistas de negocios, ingenieros de seguridad y demás stakeholders. Un organigrama bien definido y una comunicación constante aseguran que todas las tareas críticas se manejen de manera eficiente y que cualquier problema se pueda abordar rápidamente.

5.2 HERRAMIENTAS Y ENTORNOS DE DESARROLLO

Herramientas y Cadenas de Soporte

El uso de herramientas adecuadas en el desarrollo de software puede marcar una gran diferencia en la eficiencia y calidad del producto final. Las herramientas de gestión de versiones, integración continua y automatización de pruebas son esenciales para mantener el código limpio y funcional. Además, las herramientas de seguimiento de errores y gestión de proyectos ayudan a los equipos a mantenerse organizados y enfocados. La elección correcta de estas herramientas debe basarse en las necesidades específicas del proyecto y en la experiencia del equipo con dichas herramientas.

Implementación de Entornos Seguros

Crear entornos seguros para el desarrollo y prueba es fundamental para proteger el código y los datos durante el ciclo de vida del proyecto. Esto implica no solo asegurar los servidores y redes, sino también implementar políticas de acceso estrictas y utilizar herramientas de cifrado. Los entornos de desarrollo deben ser lo más similares posible a los entornos de producción para evitar sorpresas al desplegar el software. Además, es importante realizar auditorías de seguridad periódicas para identificar y corregir posibles vulnerabilidades.

Arquitectura de Software: Construcción, Despliegue y Seguimiento

La arquitectura del software es la columna vertebral de cualquier aplicación exitosa. Una buena arquitectura debe ser escalable, flexible y fácil de mantener. Durante la fase de construcción, es crucial seguir principios de diseño sólidos y mejores prácticas de codificación. El despliegue debe ser automatizado para minimizar errores humanos y asegurar una entrega rápida y fiable. El seguimiento continuo de la aplicación, mediante herramientas de monitoreo, permite detectar y resolver problemas en tiempo real, garantizando así un rendimiento óptimo y una alta disponibilidad.

5.3 PRUEBAS, VERIFICACIÓN Y MANTENIMIENTO

Pruebas y Verificación

Las pruebas y la verificación son etapas críticas en el ciclo de desarrollo de software. Las pruebas unitarias, de integración y de sistema aseguran que cada componente del software funcione correctamente y que todos los componentes interactúen de manera adecuada. Las pruebas de aceptación por parte del usuario final validan que el software cumple con los requisitos especificados. Además, las pruebas de seguridad ayudan a identificar vulnerabilidades que podrían ser explotadas por atacantes. Un enfoque sistemático y exhaustivo en las pruebas es vital para garantizar un software de alta calidad y libre de errores.

Revisión y Evaluación Continua de Vulnerabilidades

La seguridad del software no es un evento único, sino un proceso continuo. Las revisiones periódicas y la evaluación constante de vulnerabilidades son esenciales para mantener la seguridad del software a lo largo del tiempo. Esto incluye realizar auditorías de código, pruebas de penetración y análisis de vulnerabilidades regularmente. Además, es importante mantenerse actualizado con las últimas amenazas y técnicas de ataque para poder adaptar las medidas de seguridad en consecuencia. La implementación de programas de divulgación de vulnerabilidades también puede ser beneficiosa para identificar y corregir problemas rápidamente.

Funcionalidad de Software Existente

El mantenimiento del software existente es tan importante como el desarrollo de nuevas funcionalidades. Esto implica no solo corregir errores y aplicar parches de seguridad, sino también mejorar y optimizar el rendimiento del software. La monitorización continua y la retroalimentación de los usuarios son esenciales para identificar áreas de mejora. Además, es crucial planificar y gestionar las actualizaciones de software de manera que minimicen la interrupción del servicio y aseguren la continuidad del negocio.

5.4 CODIFICACIÓN Y PROTECCIÓN DE SOFTWARE

Codificación Segura

La codificación segura es fundamental para prevenir vulnerabilidades que puedan ser explotadas por atacantes. Esto incluye seguir prácticas de codificación seguras, como la validación de entradas, la gestión adecuada de errores y excepciones, y la implementación de controles de acceso adecuados. Los desarrolladores deben ser conscientes de las amenazas comunes y cómo mitigarlas mediante técnicas de codificación. La educación continua y la formación en seguridad son esenciales para mantener un alto nivel de competencia en este campo.

Protección del Código y Software

Proteger el código fuente y el software contra el acceso no autorizado y la manipulación es crucial para mantener la integridad y la confidencialidad del producto. Esto incluye el uso de sistemas de control de versiones seguros, la implementación de medidas de cifrado y la gestión adecuada de las claves de cifrado. Además, es importante asegurarse de que el código fuente esté almacenado en repositorios seguros y que solo el personal autorizado tenga acceso a él. La implementación de políticas de control de acceso y la auditoría regular de los sistemas de almacenamiento y distribución del código son prácticas recomendadas.

Seguridad del Software

La seguridad del software abarca todas las medidas y prácticas implementadas para proteger el software de amenazas y vulnerabilidades. Esto incluye no solo la codificación segura y la protección del código, sino también la implementación de controles de seguridad en todas las fases del ciclo de desarrollo. La adopción de marcos de seguridad, como OWASP y NIST, proporciona una guía estructurada para asegurar el software. Además, la colaboración con expertos en seguridad y la participación en comunidades de seguridad ayudan a mantenerse actualizado con las mejores prácticas y las últimas amenazas.
Cada uno de estos ensayos ofrece una visión integral de los aspectos clave del desarrollo seguro de software, proporcionando a los profesionales de TI y a los auditores gubernamentales una comprensión profunda y detallada de las prácticas y principios que deben seguirse para garantizar la seguridad y calidad del software.

Capítulo 6

CÓMO UTILIZAR LA GUÍA PARA LA AUDITORÍA INTERNA

6. CÓMO UTILIZAR LA GUÍA PARA LA AUDITORÍA INTERNA

Para que el auditor interno pueda aprovechar al máximo esta publicación, es conveniente que se refiera a los instrumentos complementarios: Las preguntas de auditoría temáticas y el modelo de madurez general. Cada GASIC se compone de tres componentes:

Guía de Auditoría de la Seguridad de la Información y Ciberseguridad (GASIC):
Este es el cuerpo teórico y consiste en el marco contextual necesario para que el auditor interno comprenda el alcance y del dominio de seguridad que está evaluando. Es un instrumento con los conceptos fundamentales recopilados de mejores prácticas.

Modelo de Madurez:
Recopila controles desde las mejores prácticas asociadas al tema central de Guía de Auditoría, organiza los controles en una propuesta de madurez y permite al auditor conocer los requisitos que debería evaluar.

Ejemplos de Preguntas de Auditoría:
Complementa el modelo de madurez a través de una serie de preguntas organizadas en varios documentos. Cada documento representa un control que pertenece a uno de los ejes temáticos definidos al interior de la Guía de Auditoría.

La ilustración a continuación presenta esta estructura documental:

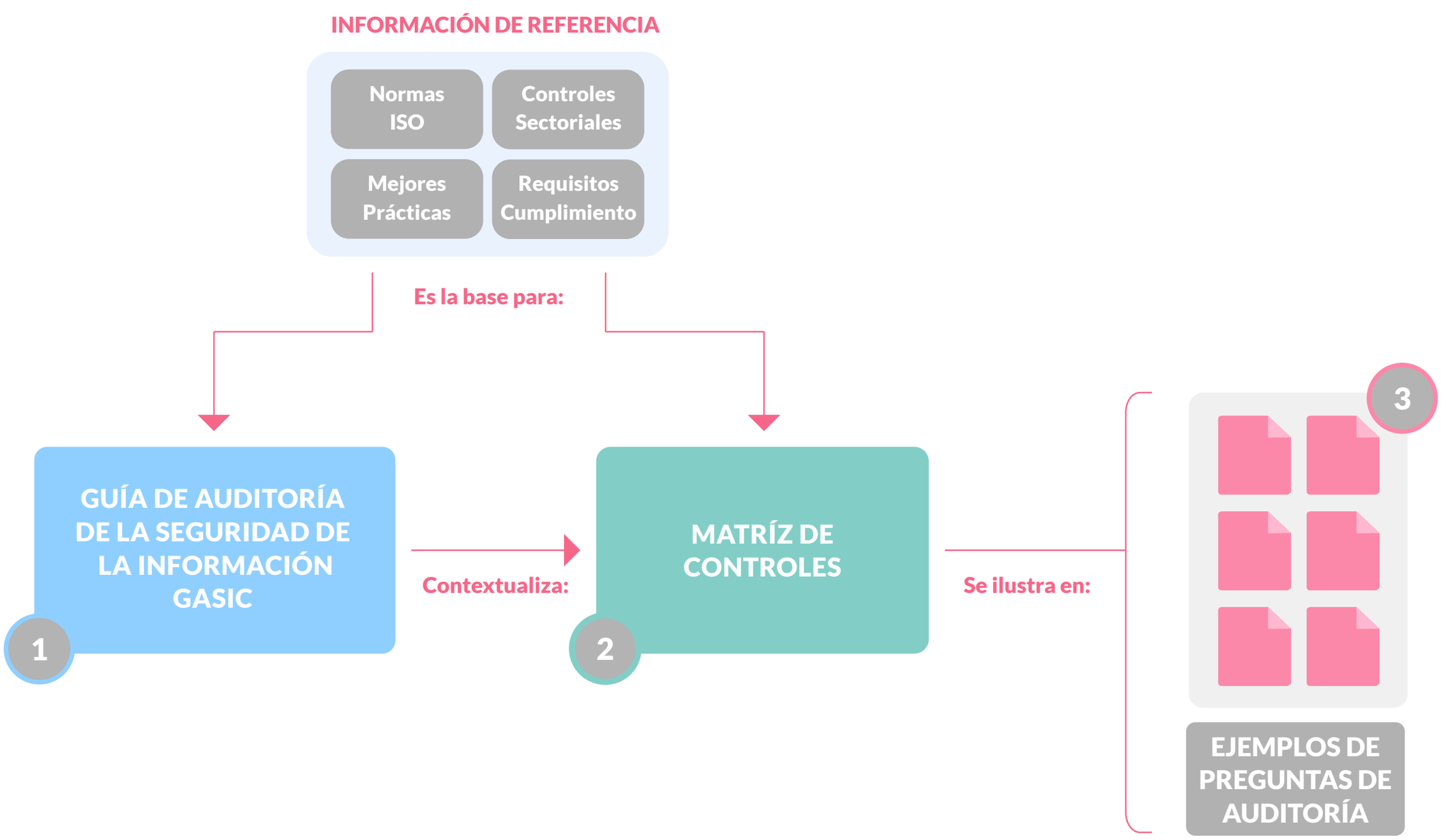


Ilustración n°5. Modo de uso y Estructura Documental GASIC. Fuente: Elaboración Propia

El método de trabajo sugerido es el siguiente:

01 **El auditor interno debe estudiar cada Guía de Auditoría y su contexto para tener plena comprensión del tema a trabajar.**

02 **A continuación, puede utilizar el Modelo de Madurez para seleccionar los controles que sean apropiados para la organización. La selección de controles debe estar alineados con:**

a. **La estrategia de la organización.**
b. **Los resultados de la evaluación de riesgos.**
c. **Los requisitos de cumplimiento.**
d. **La estrategia de auditoría interna, expresada en el plan.**

03 **Por último, puede utilizar los documentos de ejemplo para la planificación de las preguntas y pruebas que fuese a realizar. El formato del programa, plan, instrumentos, pruebas y reportería debe ser aquel solicitado en el contexto de cada auditoría, que está fuera del alcance de esta guía.**

NOTA

Los ejemplos de pruebas tienen como propósito ilustrar la forma en la que los requisitos de los marcos que se encuentran en el matriz de controles. El auditor puede elegir utilizar un conjunto de estos ejemplos o diseñar sus propias pruebas para evaluar el nivel de cumplimiento de cada control.

En ningun caso, los ejemplos pretenden ser una lista completa; recuerde, debe contextualizar el ejercicio a la realidad de su organización.

Ejes temáticos

1.Técnicas de Seguridad: Este eje tiene por objetivo la consideración de mecanismos de seguridad mínimos para establecer una línea base que permita a la organización desarrollar software mientras minimiza los riesgos.

	OBJETIVO ESPECÍFICO	CRITERIO DE AUDITORÍA
1	**Implementación de Entornos Seguros**	**La organización dispone de controles de seguridad generales para minimizar los riesgos de los entornos de producción.**
2	**Protección del Código y Software**	**La organización almacena de forma segura el código fuente y los elementos de configuración.**
3	**Codificación Segura**	**La organización define y opera un proceso de validación en la codificación que observe las mejores prácticas y ayude a evitar los riesgos de seguridad.**
4	**Revisión y Evaluación Continua de Vulnerabilidades y Configuraciones**	**La organización define y opera un proceso de revisión de vulnerabilidades y de configuración para minimizar la probabilidad de que ocurra un incidente de seguridad.**

2. Seguridad en el Proceso de SDLC: Este eje está enfocado en incorporar buenas prácticas de seguridad dentro del ciclo de desarrollo de software, sin importar el tipo de metodología que utilice. Observa las prácticas básicas de seguridad que buscan minimizar los riesgos de esta operación.

	OBJETIVO ESPECÍFICO	CRITERIO DE AUDITORÍA
1	**Definición del Proceso**	**La organización define un proceso de desarrollo de software seguro, que considera las necesidades de las partes interesadas, es evaluado, medido y mejorado consistentemente.**
2	**Definición de Requisitos de Seguridad**	**Identificar y documentar todos los requisitos de seguridad para las infraestructuras y procesos de desarrollo de software de la organización, y mantener los requisitos a lo largo del tiempo.**
3	**Funciones y Responsabilidades**	**La organización define los roles y responsabilidades que deberán dar cuenta de los resultados de la seguridad en SDLC.**
4	**Herramientas y Cadenas de Soporte**	**La organización define las cadenas de herramientas de operación y seguridad que serán utilizadas en el SDLC.**
5	**Pruebas y Verificación**	**La organización define, implementa, ejecuta, controla y mejora las pruebas y mecanismos de verificación, con foco en la seguridad SDLC.**
6	**Análisis Continuo de Software**	**La organización define procedimientos para la evaluación y análisis de los componentes de software una vez han completado el proceso de desarrollo.**