



Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Software

Desarrollo de Software Seguro

**Investigación y exposición sobre
metodologías y tipos de SSDLC
(metodología Rational Unified
Process-Secure RUPSec)**

Integrantes: Jonathan García, Axel Herrera, Bryan Quispe

Docente: MsC. Geovanny Cudco

4 de diciembre de 2025

Objetivos

1.1. Objetivo General

Fomentar la comprensión sobre la importancia del desarrollo de software seguro mediante la investigación de las metodologías y tipos de SSDLC, y desarrollar habilidades de síntesis y presentación a través de la implementación de un pipeline CI/CD automatizado.

1.2. Objetivos Específicos

- Según la metodología de Rational Unified Process-Secure (RUPSec), se debe describir y enumerar las etapas del ciclo de vida y herramientas recomendadas para cada etapa dentro del flujo de trabajo automatizado.
- Especificar ventajas y desventajas de su implementación con casos de uso o ejemplos reales, demostrando la reducción de deuda técnica de seguridad.
- Analizar cómo la integración de prácticas de seguridad en cada fase del RUPSec (y el pipeline CI/CD) contribuye a la prevención temprana de vulnerabilidades y al fortalecimiento del producto final (Shift-Left Security).

Etapas principales del ciclo de vida RUPSec

La metodología Process-Secure (RUPSec) adopta la estructura iterativa e incremental del Rational Unified Process (RUP). La principal diferencia teórica radica en la incorporación transversal de la ingeniería de seguridad, transformando la seguridad en un requisito funcional y arquitectónico desde el primer momento, en lugar de un añadido final.

A continuación, se describen los objetivos teóricos de cada fase:

2.1. Fase de Inicio (Inception)

En esta etapa, el objetivo primordial es definir el alcance de la seguridad y justificar el negocio para las inversiones en protección. Se busca establecer una visión común de los riesgos.

- **Definición de Requisitos de Seguridad:** Identificación de las necesidades de confidencialidad, integridad y disponibilidad (CIA) del sistema.
- **Evaluación de Riesgos Iniciales:** Análisis preliminar para determinar la criticidad de los datos que manejará el sistema.
- **Desarrollo de Casos de Abuso (Misuse Cases):** Definición teórica de escenarios negativos donde se modela el comportamiento de un atacante para entender cómo podría intentar vulnerar el sistema.
- **Cumplimiento Normativo:** Identificación de leyes, regulaciones y políticas internas que el software debe cumplir obligatoriamente.

2.2. Fase de Elaboración (Elaboration)

Durante la elaboración, el enfoque se centra en la arquitectura. La seguridad se diseña estructuralmente para mitigar los riesgos identificados en la fase anterior.

- **Diseño de Arquitectura de Seguridad:** Definición de los mecanismos de defensa estructurales, como patrones de autenticación, autorización y segmentación de redes.
- **Refinamiento de Amenazas:** Análisis detallado de los vectores de ataque potenciales sobre la arquitectura propuesta.
- **Validación de Diseño:** Verificación de que la arquitectura propuesta cumple con los requisitos de seguridad antes de comenzar la producción masiva de código.
- **Selección de Estándares:** Definición de las guías de estilo y normas de codificación segura que seguirá el equipo de desarrollo.

2.3. Fase de Construcción (Construction)

Esta fase se centra en la implementación física del software. La teoría dicta que la seguridad debe ser proactiva durante la escritura del código.

- **Implementación Segura:** Desarrollo de componentes de software siguiendo las prácticas de "Secure Coding" para evitar vulnerabilidades comunes (como inyecciones o desbordamientos).
- **Integración Continua de Seguridad:** Incorporación de validaciones de seguridad cada vez que se integra nuevo código al repositorio principal.
- **Gestión de Vulnerabilidades:** Identificación y corrección temprana de defectos de seguridad en el código fuente antes de que el sistema esté completo.

2.4. Fase de Transición (Transition)

La fase final busca validar que el producto es seguro para operar en el entorno real del cliente. Se pasa de la construcción a la validación operativa.

- **Validación de Seguridad Final:** Confirmación de que el sistema resiste ataques en un entorno similar al de producción.
- **Auditoría de Cumplimiento:** Verificación final de que se cumplen todos los requisitos normativos y legales definidos en la fase de Inicio.
- **Plan de Respuesta a Incidentes:** Preparación de los protocolos y documentación necesaria para que el equipo de operaciones pueda reaccionar ante posibles brechas de seguridad una vez el sistema esté en vivo.

Herramientas y Prácticas de Seguridad

La implementación efectiva de RUPSec requiere el uso de herramientas especializadas y la adopción de prácticas de industria en cada fase del ciclo de vida. A continuación, se detallan los recursos recomendados.

3.1. Herramientas y Prácticas en la Fase de Inicio

En esta fase, el enfoque es conceptual y de análisis de riesgos.

■ Prácticas Recomendadas:

- *Brainstorming de Seguridad*: Reuniones con stakeholders para identificar activos críticos.
- *Definición de Casos de Abuso (Misuse Cases)*: Diagramación de actores maliciosos interactuando con el sistema.

■ Herramientas de Apoyo:

- **Herramientas CASE/UML**: Enterprise Architect, Lucidchart o StarUML (para diagramar casos de uso y abuso).
- **Plantillas de Riesgos**: Hojas de cálculo o software de GRC (Governance, Risk and Compliance) para matrices de probabilidad e impacto.

3.2. Herramientas y Prácticas en la Fase de Elaboración

El enfoque aquí es el diseño seguro y la anticipación de ataques.

■ Prácticas Recomendadas:

- *Modelado de Amenazas (Threat Modeling)*: Análisis sistemático para identificar vulnerabilidades de diseño (ej. metodología STRIDE).
- *Diseño de Controles de Seguridad*: Definición de esquemas de autenticación y autorización.

■ Herramientas de Apoyo:

- **Microsoft Threat Modeling Tool**: Estándar para crear diagramas de flujo de datos y detectar amenazas automáticamente.
- **OWASP Threat Dragon**: Herramienta de código abierto para modelado de amenazas visual.

3.3. Herramientas y Prácticas en la Fase de Construcción

Durante la codificación, se utilizan herramientas de análisis automático.

■ Prácticas Recomendadas:

- *Estándares de Codificación Segura*: Seguir guías como OWASP Top 10 o CERT Secure Coding.
- *Revisión de Código por Pares (Peer Review)*: Auditoría manual de código entre desarrolladores.

■ **Herramientas de Apoyo:**

- **SAST (Static Application Security Testing)**: SonarQube, Fortify o Checkmarx (analizan el código fuente sin ejecutarlo).
- **SCA (Software Composition Analysis)**: OWASP Dependency-Check (detecta vulnerabilidades en librerías de terceros).
- **Plugins de IDE**: SonarLint (alerta de errores de seguridad en tiempo real mientras se escribe código).

3.4. Herramientas y Prácticas en la Fase de Transición

Antes del despliegue, se ataca el sistema para validarlo.

■ **Prácticas Recomendadas:**

- *Penetration Testing (Hacking Ético)*: Simulación controlada de ciberataques externos e internos.
- *Hardening del Servidor*: Aseguramiento de la configuración del entorno de despliegue.

■ **Herramientas de Apoyo:**

- **DAST (Dynamic Application Security Testing)**: OWASP ZAP o Burp Suite (escanean la aplicación en ejecución buscando fallos como XSS o SQL Injection).
- **Escáneres de Red**: Nmap o Nessus (para detectar puertos abiertos y servicios vulnerables en el servidor).
- **WAF (Web Application Firewall)**: ModSecurity o AWS WAF (para proteger la aplicación una vez desplegada).

Ventajas y desventajas de su implementación

Al momento de implementar esta metodología en la seguridad de un producto de software, se deben considerar tanto las ventajas como las desventajas que implica su adopción.

Ventajas

Seguridad integrada desde el inicio del ciclo de vida: A lo largo del ciclo de vida del producto de software, la seguridad se incorpora en todas las fases, lo cual permite prevenir fallos ante posibles vulnerabilidades.

Trazabilidad completa de requisitos funcionales y de seguridad: Cada requisito debe ser documentado adecuadamente, incluyendo casos de uso y casos de abuso, lo cual mejora la auditoría, el control y la claridad del proyecto.

Reducción de vulnerabilidades en el proyecto: Al implementar nuevas actividades en la extensión de RUP, como el modelado de amenazas, revisiones de código y pruebas de penetración, se obtiene un sistema con menos vulnerabilidades y mayor confiabilidad.

Desventajas

Aumento de tiempo y costo inicial del proyecto: Al implementar controles y pruebas de seguridad, se requiere mayor esfuerzo, personal capacitado y más tiempo durante las primeras fases del desarrollo.

Requiere experiencia especializada: RUPSec necesita profesionales con conocimientos en ingeniería de requisitos seguros, arquitectura segura, modelado de amenazas y pruebas avanzadas de seguridad.

Generación excesiva de documentación: Al ser un proceso iterativo y basado en artefactos, se genera una gran cantidad de documentación, tanto para los casos de uso como para los casos de abuso, por lo que es necesario gestionar adecuadamente todos los documentos producidos.

Casos de uso o ejemplos reales

5.1. Caso Real Gubernamental: Web Service Seguro

5.1.1. Incepción

El sistema real consistió en un *web service* utilizado para compartir información sensible entre varias instituciones públicas. Los riesgos asociados incluían:

- Fuga de datos personales,
- Manipulación de consultas,
- Accesos no autorizados,
- Falta de trazabilidad.

Los requisitos de seguridad definidos fueron:

- Autenticación fuerte mediante certificados,
- Cifrado extremo a extremo,
- Auditoría obligatoria,
- Validación estricta de mensajes,
- Registro detallado de eventos.

5.1.2. Elaboración

El equipo realizó un modelado de amenazas, identificando riesgos externos, internos y técnicos. La arquitectura se diseñó con múltiples capas:

- VPN interinstitucional,
- Firewalls segmentados e IDS,
- WS-Security,
- Validación de esquemas XSD,
- Cifrado AES-256 en base de datos,
- Controles RBAC por institución,
- Logs inmutables enviados a un SIEM.

5.1.3. Construcción

Se aplicaron revisiones formales de código según normas de seguridad del Estado, junto con análisis automatizado mediante:

- Fortify,
- Veracode,
- Checkmarx.

Las pruebas incluyeron:

- Pruebas de penetración internas,
- Evaluación contra inyección XML (XXE),
- Fuzzing de endpoints SOAP/REST,
- Simulación de ataques *replay*.

5.1.4. Transición

El sistema fue desplegado bajo un proceso de certificación de seguridad. Se implementó:

- Monitoreo 24/7,
- Rotación de certificados,
- Controles estrictos de acceso,
- Auditorías semestrales.

La trazabilidad se mantuvo en todas las etapas: desde requisitos hasta pruebas y cambios en producción, asegurando cumplimiento normativo.

5.2. Conclusión sobre el caso real gubernamental

El caso demuestra la aplicabilidad de RUPSec en un entorno crítico. Esta metodología permite un enfoque ordenado, iterativo y auditable, integrando seguridad desde las primeras etapas y manteniéndola durante todo el ciclo de vida. Esta plataforma gubernamental se beneficia de RUPSec al enfrentar riesgos elevados, cumplir regulaciones y asegurar la continuidad operativa.

Conclusiones

6.1. Conclusiones principales

- La implementación de RUPSec permite fortalecer la seguridad del software al integrarla desde las primeras fases del desarrollo, logrando detectar y mitigar vulnerabilidades antes de que se conviertan en riesgos críticos para el sistema.
- El enfoque basado en trazabilidad y control documental facilita la auditoría y el seguimiento de los requisitos, lo que mejora la calidad del producto final y permite una mejor comprensión del proceso por parte del equipo técnico y los stakeholders.
- Aunque RUPSec aporta beneficios significativos, también implica un aumento en el esfuerzo, tiempo y costo, por lo que su adopción requiere una planificación adecuada y la participación de personal capacitado para garantizar resultados efectivos.

6.2. Recomendaciones

- Se recomienda aplicar RUPSec especialmente en proyectos que manejan información sensible o crítica, asegurando que el equipo reciba capacitación adecuada para optimizar el proceso y evitar que la carga documental o la complejidad afecten al desarrollo.