GUÍA DE BUENAS PRÁCTICAS APLICABLE A LA METODOLOGÍA DE DESARROLLO ÁGIL SCRUM PARA FORTALECER LA SEGURIDAD DE LA INFORMACIÓN

TRABAJO DE GRADO

JUAN SEBASTIÁN RESTREPO ÁNGEL

1622010135

INSTITUCIÓN UNIVERSITARIA POLITÉCNICO GRANCOLOMBIANO

FACULTAD DE INGENIERÍA Y CIENCIAS BÁSICAS

ESPECIALIZACIÓN EN SEGURIDAD DE LA INFORMACIÓN

2017

GUÍA DE BUENAS PRÁCTICAS APLICABLE A LA METODOLOGÍA DE DESARROLLO ÁGIL SCRUM PARA FORTALECER LA SEGURIDAD DE LA INFORMACIÓN

TRABAJO DE GRADO

JUAN SEBASTIÁN RESTREPO ÁNGEL

1622010135

Asesor

ALEJANDRO CASTIBLANCO CARO

INSTITUCIÓN UNIVERSITARIA POLITÉCNICO GRANCOLOMBIANO

FACULTAD DE INGENIERÍA Y CIENCIAS BÁSICAS

ESPECIALIZACIÓN EN SEGURIDAD DE LA INFORMACIÓN

2017

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____

_____

_____

Firmas de los jurados

Bogotá D.C., 15 de Septiembre de 2017.

TABLA DE CONTENIDO

INTRODUCCIÓN

El software representa un conjunto complejo de aspectos de seguridad que deben ser cubiertos por los arquitectos, diseñadores y programadores. Las aplicaciones más seguras son aquellas en las cuales la seguridad se tuvo en cuenta durante todo su proceso de desarrollo. Las metodologías ágiles se fundamentan en el manifiesto ágil, que menciona que la prioridad más alta es la satisfacción del cliente a través de entregas tempranas y continúas de un producto con valor (Fowler & Highsmith, 2001), por lo cual el cumplimiento de requerimientos no funcionales como mantener los principios básicos de la seguridad de la información: Confidencialidad, Disponibilidad, Integridad; pasa a un segundo plano. La poca atención puesta en la protección de las aplicaciones, provoca que los usuarios comprometan sus datos y otros activos de información al emplearlas; y es esta necesidad la que motiva la propuesta de incorporar un enfoque de seguridad en la metodología de desarrollo ágil SCRUM.

En la primera parte de este documento se presenta el resumen ejecutivo en donde a partir de la identificación de la situación problema que se presenta en una compañía que utiliza la metodología de desarrollo ágil SCRUM en sus proyectos, se plantea realizar la comparación de algunas metodologías de desarrollo para definir las características de un software seguro, el análisis de los riesgos identificados al utilizar SCRUM, y finalmente se propone una guía de buenas prácticas que sea aplicable para mejorar la seguridad durante el ciclo de vida de desarrollo de la metodología SCRUM.

Posteriormente se presentan la justificación, el marco teórico y la metodología que soportan el desarrollo del proyecto y finalmente se describen los resultados alcanzados y las conclusiones obtenidas.

AGRADECIMIENTOS

A mis padres por su apoyo incondicional, a mi esposa por su amor y comprensión, a los docentes del Politécnico Grancolombiano por las experiencias compartidas y el conocimiento transmitido que me han permitido crecer como persona y como profesional.

1. RESUMEN EJECUTIVO

La situación problema descrita en este trabajo se presenta en una compañía de desarrollo de software que para el campo académico se llamará New Soft. New Soft es una compañía pequeña que cuenta con alrededor de 50 empleados, con 7 años de experiencia en el mercado y se especializa en el desarrollo de aplicaciones de comercio electrónico empleando para ello el modelo de desarrollo ágil Scrum. Dentro de la compañía se encuentran 5 equipos de desarrollo por lo que actualmente se maneja ese mismo número de proyectos para clientes al tiempo.

En el desarrollo de software, la seguridad de la información ha pasado de ser un requerimiento no funcional que podía implementarse como parte de la calidad del producto a un elemento primordial y obligatorio de cualquier aplicación. Las metodologías de desarrollo ágiles tienen como objetivo proporcionar un dinamismo al ciclo de vida del desarrollo que permita la interacción entre el equipo de desarrollo centrando la mayor parte del esfuerzo en la codificación y proporcionando una alta capacidad de adaptación en tiempo real para cumplir las expectativas y requerimientos funcionales de los clientes.

"La clave de un software seguro, es el proceso de desarrollo utilizado" (Brito, 2013). En el seguimiento de una metodología es donde se construye una aplicación que pueda resistir o sostenerse ante ataques, recuperarse rápidamente y mitigar el daño causado por la explotación de vulnerabilidades que no puedan ser eliminadas o resistidas. Muchos de los defectos relacionados con la seguridad en software se pueden evitar si los desarrolladores estuvieran mejor preparados para reconocer las implicaciones de su diseño y de las posibilidades de implementación. En New Soft se vienen presentando inconvenientes con algunos clientes debido a que vulnerabilidades de seguridad se ponen en manifiesto tiempo después de que las aplicaciones se encuentren en entornos de producción, lo que en ocasiones conlleva a que se deba suspender el funcionamiento de las aplicaciones mientras el equipo de desarrollo brinda el soporte necesario. Ante estos eventos, se hace necesario comprobar si, *¿La metodología de desarrollo de software empleada por New Soft garantiza que los productos entregados a los clientes preserven la seguridad de la información?*

Una aplicación segura nace en el seguimiento de una metodología de desarrollo que contemple buenas prácticas, métodos y herramientas para mejorar desde los requisitos funcionales como los aspectos no funcionales, y es allí donde se incluye la seguridad de la información. Un ciclo de vida del software seguro que abarque desde los clientes hasta la dirección de los proyectos y su desarrollo, pruebas y despliegue debe ser la guía para los equipos de desarrollo. En contraposición con los modelos de desarrollo clásicos, los denominados modelos de desarrollo ágiles proporcionan al ciclo de vida del desarrollo un dinamismo que permite la interacción entre diferentes equipos de forma ágil, centrando la mayor parte del esfuerzo en la codificación y proporcionando una alta capacidad de adaptación en tiempo real durante el desarrollo para cumplir las expectativas y requerimientos funcionales.

Con el desarrollo de este proyecto se pretende diseñar un modelo de buenas prácticas que sea aplicable a la metodología de desarrollo ágil SCRUM para fortalecer la seguridad de la información y reducir las vulnerabilidades en las aplicaciones desarrolladas siguiendo mencionada metodología. Para ello se propone definir las características y /o propiedades de un software seguro a partir de la comparación de tres metodologías para el desarrollo seguro de aplicaciones: *Microsfoft Security Development Lifecycle (MS SDL)*, OWASP CLASP (Comprehensive Lightweight Application Security Process) y NIST 800-64. Asimismo, se propone analizar los riesgos del modelo de desarrollo ágil SCRUM durante el ciclo de vida del desarrollo de software en la compañía New Soft, identificando las vulnerabilidades y amenazas que llevan a fallos de seguridad en las aplicaciones, empleando para ello el marco de trabajo MAGERIT hasta el punto de valorar el riesgo. Por último se plantea el diseño de una guía con un modelo de buenas prácticas aplicable a la metodología ágil de desarrollo de software SCRUM que incluye la seguridad de la información como parte fundamental de la arquitectura de las aplicaciones.

Para cumplir con los objetivos propuestos, inicialmente se llevará a cabo una etapa de indagación y consulta sobre software seguro que abarque definiciones y metodologías; y sobre marcos de trabajo para el análisis de riesgos específicamente MAGERIT. Posteriormente, se tendrá un periodo de análisis para identificar zonas de convergencia entre las metodologías de desarrollo seguro y establecer cuáles son las prácticas aplicables

a un ciclo de vida ágil, así como se establecerán las vulnerabilidades y amenazas en la compañía New Soft en el desarrollo de sus productos. Finalmente se tendrá una etapa de definición y resultados, en la cual se construirá el comparativo de las tres metodologías de desarrollo de software seguro en aras de identificar mejores prácticas de seguridad de la información que se puedan aplicar a metodologías de desarrollo ágil; el análisis de riesgos del modelo de desarrollo ágil SCRUM en la compañía New Soft con el fin de identificar las vulnerabilidades, amenazas y/o errores comunes que llevan a fallos de seguridad en las aplicaciones, empleando para ello el marco de trabajo MAGERIT; y la guía con un modelo de buenas prácticas de seguridad que sea aplicable a la metodología de desarrollo ágil SCRUM para fortalecer la seguridad de la información durante las diferentes etapas del ciclo de vida de desarrollo de software.

2. JUSTIFICACIÓN

Para el año 2016 las estadísticas resultantes de los estudios realizados por algunas de las principales compañías en el campo de las tecnologías de la información relacionadas con vulnerabilidades en aplicaciones de software reflejan lo siguiente:

- Microsoft: 41,8% de todas las divulgaciones de vulnerabilidades se califican como muy graves, siendo la cifra más alta en los últimos tres años
- Hewlett Packart: más de un tercio de las aplicaciones escaneadas (35%) exhibió al menos una vulnerabilidad crítica o de alta severidad, y presenta las principales vulnerabilidades en aplicaciones así: protocolos de transporte, configuración de servidores web, seguridad en cookies, fugas en sistemas de información y violación a la privacidad.

La poca preparación que reciben los gerentes de proyecto y los ingenieros de software en diseño y desarrollo de aplicaciones seguras, trae consigo la ejecución de malas prácticas y técnicas que no permiten un reconocimiento eficaz de posibles fallos y vulnerabilidades, lo que conlleva a que los requerimientos de seguridad en las aplicaciones sean insuficientes. Durante los últimos 12 meses, los equipos de desarrollo de New Soft recibieron un total de 4 capacitaciones, todas relacionadas en herramientas de desarrollo pero ninguna con temas de seguridad de la información.

La elaboración de malos requisitos, el diseño y la implementación del código normalmente representa del 40 al 50 por ciento del costo total del desarrollo de software (Jones, 1986). Corregir un problema de requisitos una vez que el software está funcionando, normalmente cuesta entre 50 y 200 veces más de lo que se necesitaría para corregir el mismo problema si se detecta durante la fase de requisito (Boehm & Papaccio, 1988). Si se pueden prevenir los fallos o detectarlos a tiempo, los beneficios de costos y cronogramas son significativos. Es más módico corregir un error en el momento que los requisitos se especifican de lo que es después del diseño, implementación, documentación y casos de prueba. El ahorro potencial de la detección temprana de errores es significativo y aproximadamente el 60 por ciento de todos los defectos generalmente se presentan en la fase de diseño (Gilb, 1988). Dentro del

proceso de pruebas de New Soft, se ha observado que para cada funcionalidad en una aplicación se ejecuta en promedio 10 casos de prueba, de los cuales 3 son pruebas no funcionales, por lo tanto en promedio para cada aplicación solo se realiza un porcentaje de pruebas mínimas en relación a la seguridad de la información. Asimismo, los clientes han reportado vulnerabilidades en promedio un mes después de estar las aplicaciones en entorno de producción, y los equipos de desarrollo de la compañía han tomado hasta 2 días en resolver estos inconvenientes; por lo que el tiempo que las aplicaciones están fuera de línea por vulnerabilidades detectadas en producción es insatisfactorio.

Cuando un producto software tiene demasiados defectos, incluyendo fallos de seguridad, vulnerabilidades y errores, los ingenieros de software pueden terminar gastando más tiempo en corregir estos problemas de los que gastan en el desarrollo del software en el primer momento. En New Soft, se maneja un tiempo promedio de 45 días por incremento funcional en los diferentes proyectos desarrollados. Por otra parte cuando se haya alguna vulnerabilidad en fase de pruebas, se tiene un registro de 3 días en promedio para corregir. Los gerentes de proyecto podrían lograr en un plazo más corto productos de mayor calidad solo con abordar la seguridad en todo el ciclo de vida del software, especialmente durante las primeras etapas, para aumentar la probabilidad de que el software sea más seguro desde el primer momento.

3. MARCO TEÓRICO Y REFERENTES

3.1 Metodologías de desarrollo de software

En el desarrollo de software, una metodología hace cierto énfasis al entorno en el cuál se plantea y estructura el desarrollo de un sistema. Una Metodología de desarrollo de software, consiste principalmente en hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo. “Una estructura general para la ingeniería de software define cinco actividades estructurales: comunicación, planeación, modelado, construcción y despliegue” (Pressman, 2010). Estas actividades se pueden llevar a cabo en un flujo de proceso lineal, iterativo, evolutivo o paralelo.

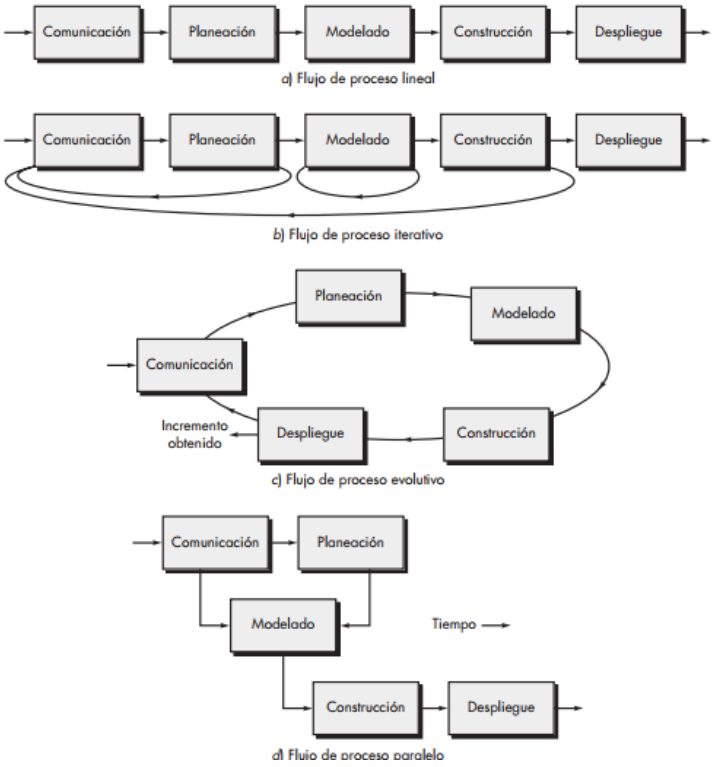

Figura 1. Flujos del proceso de ingeniería del software. (Pressman, 2010).

3.1.1 Metodologías de desarrollo tradicionales

- *Metodología en cascada*: También es llamado como ciclo de vida clásico, plantea un enfoque sistemático y secuencial para el desarrollo del software. Inicia con la

especificación de los requerimientos por parte del cliente, avanza a través de la planeación, modelado, construcción y despliegue, y concluye con el soporte al software terminado.

- *Metodología de proceso incremental*: Propone un flujo de proceso evolutivo, en donde cada secuencia lineal produce incrementos del software. Cuando se utiliza un modelo incremental, es frecuente que en el primer incremento se aborden los requerimientos básicos, pero no se proporcionan muchas características suplementarias. El cliente usa el producto y como resultado de su evaluación se desarrolla un plan para el incremento siguiente. El plan incluye la modificación del producto fundamental para cumplir mejor las necesidades del cliente, así como la entrega de características adicionales y más funcionalidad. Este proceso se repite después de entregar cada incremento, hasta terminar el producto final.
- *Metodología de proceso evolutivo*: Sigue un flujo de proceso iterativo y se caracterizan por que el concepto del producto se desarrolla progresivamente a medida que avanza el proyecto de tal manera que permite desarrollar versiones cada vez más completas del software. Para ello se puede utilizar dos modelos de proceso evolutivo: Hacer prototipos o desarrollo en espiral.

3.1.2 Metodologías de desarrollo ágil

Los modelos de desarrollo ágil tienen como objetivo proporcionar al ciclo de vida del desarrollo un dinamismo que permita la interacción entre diferentes equipos o unidades de forma ágil, centrando la mayor parte del esfuerzo en la codificación y proporcionando una alta capacidad de adaptación durante el desarrollo para cumplir las expectativas y requerimientos funcionales. Un proceso de software ágil debe adaptarse incrementalmente, y para lograr la adaptación incremental se requiere retroalimentación con el cliente de modo que sea posible hacer las adaptaciones apropiadas. Según Pressman (2010), “un catalizador eficaz para la retroalimentación con el cliente es un prototipo operativo”. Deben entregarse incrementos funcionales (prototipos ejecutables) en periodos cortos, de tal forma que la adaptación vaya al ritmo del cambio. En el origen de las metodologías ágiles, Beck et al (2001) definen los siguientes principios:

- “Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.”

Entre las metodologías de desarrollo ágil se destacan:

- Programación Extrema XP: Creada por Kent Beck, establece que el fundamento para todo trabajo realizado como parte de XP se debe basar en 5 valores: comunicación, simplicidad, retroalimentación, valentía y respeto. La programación

extrema usa un enfoque orientado a objetos como paradigma preferido de desarrollo, y engloba un conjunto de reglas y prácticas que ocurren en el contexto de cuatro actividades estructurales: planeación, diseño, codificación y pruebas.

- KANBAN: Es una metodología para gestionar la creación de productos con énfasis en la entrega continua sin sobrecargar el equipo de desarrollo. Kanban se fundamenta en tres principios: Visualizar el trabajo diario (flujo de trabajo) para estar informado del contexto de las tareas, limitar la cantidad de trabajo en progreso (WIP Work in progress) para equilibrar el enfoque de los equipos de desarrollo y mejorar el flujo comenzando una actividad solo cuando la anterior esté terminada. "Los equipos tienen así opciones de planificación más flexibles, resultados más rápidos, un enfoque más claro y transparencia a lo largo del ciclo de desarrollo" (Radigan, 2017).
- SCRUM: Dado a que es la metodología utilizada por la compañía del caso de estudio, se amplía su referencia en el siguiente apartado.

3.2 SCRUM

Según López, Menzinsky y Palacio (2016), SCRUM se caracteriza por adoptar una estrategia de desarrollo incremental, en donde la calidad del resultado se basa principalmente en el conocimiento tácito de las personas en equipos autoorganizados, solapando las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo de vida secuencial o de cascada. Este modelo fue identificado y definido por Ikujiro Nonaka e Hirotaka Takeuchi a principios de los 80, al analizar cómo desarrollaban los nuevos productos las principales empresas de manufactura tecnológica.

Dentro del marco de reglas de SCRUM que fue definido en 1995 por Ken Schwaber y Jeff Sutherland, se establecen tres elementos: Roles, Eventos y Artefactos.

- Roles: Dueño de producto, Equipo de desarrollo, Scrum master, intereresados.
- Eventos: Sprint, reunión de planificación, Scrum diario, revisión de sprint, retrospectiva de sprint.
- Artefactos: Pila del producto, pila del sprint, incremento.

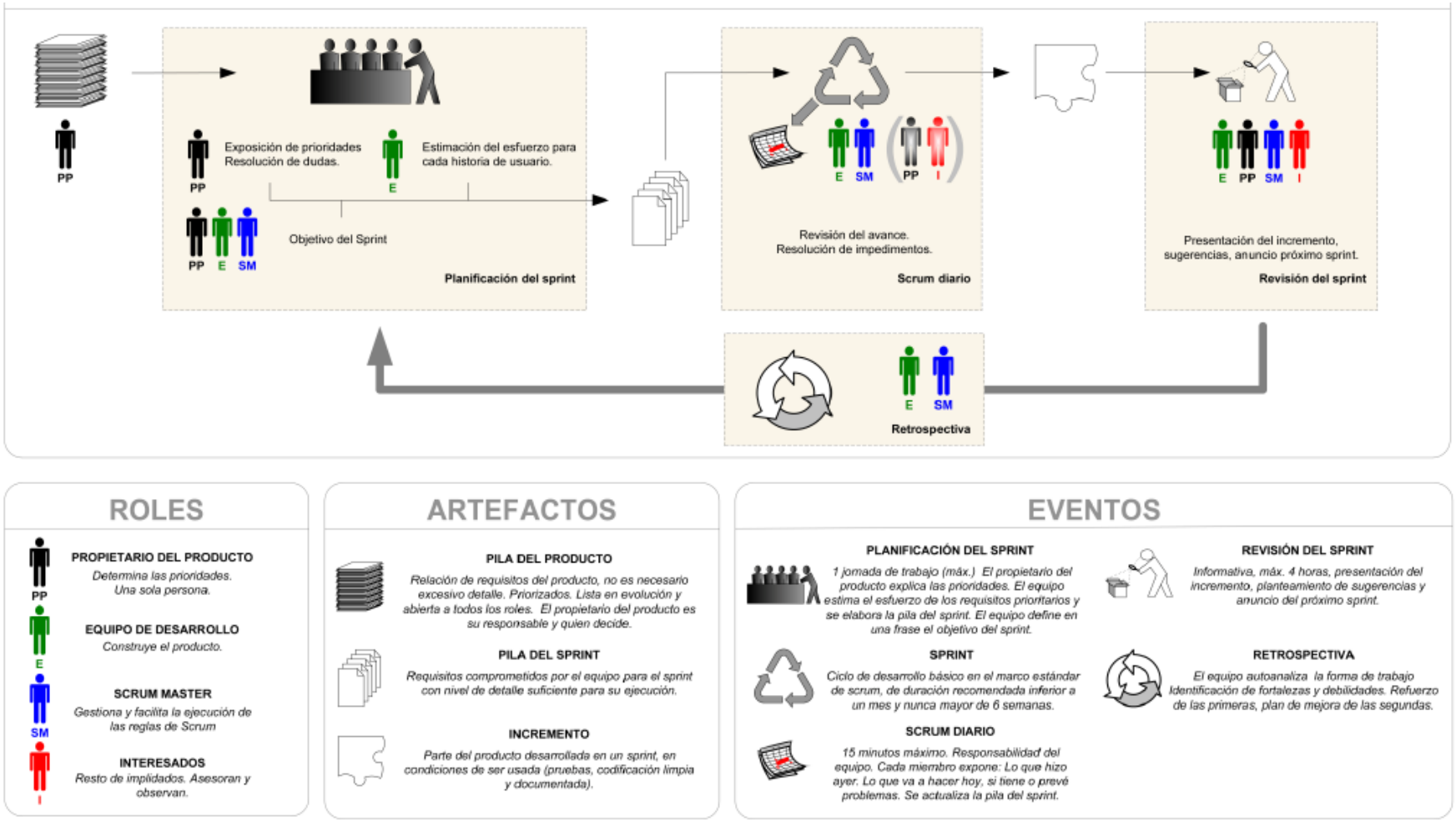

Figura 2. Marco Scrum. López et al (2016)

3.3 Software seguro

De acuerdo con McGraw (2004), el principal objetivo de la seguridad del software es la construcción de aplicaciones con mayor calidad, más robustas y libre de defectos. El valor de las aplicaciones ya no reside solamente en su capacidad de mejorar la productividad y la eficiencia de las organizaciones, sino también en que posea la aptitud de continuar operando de manera confiable aún ante eventos que amenacen su utilización; esto debido a la creciente dependencia de tareas críticas que se ejecutan a través de algún software.

Garantizar la seguridad del software se ha convertido en algo fundamental debido al incremento de amenazas atribuibles a la posibilidad de explotar vulnerabilidades presentes en el mismo, lo que pone en riesgo el objetivo de negocio de las organizaciones. El nivel de exposición al riesgo es cada vez mayor y en general se limita debido a los siguientes hechos (Castellaro, Ramos, Romaniz & Pessolani, 2009):

- El software es el eslabón más débil en la ejecución exitosa de los sistemas interdependientes y las aplicaciones de software.
- La externalización y la utilización de componentes de la cadena de suministro de software aumenta la exposición al riesgo.
- La sofisticación y la naturaleza cada vez más sigilosa de los ataques facilita la explotación.
- Reutilización de software heredado con otras aplicaciones introduce consecuencias no deseadas, lo que aumenta el número de objetivos vulnerables.
- Los líderes empresariales no están dispuestos a realizar inversiones de riesgo adecuado en materia de seguridad de software.

El concepto de aseguramiento implica que el software debe tener confiabilidad (reliability), es decir que debe operar libre de fallas durante un periodo o cantidad de operaciones en un ambiente especificado bajo condiciones esperadas; protección (safety), es decir la capacidad de persistir su confiabilidad de cara a escenarios no previstos; y seguridad (security), como la habilidad para resistir, tolerar y recuperarse de acontecimientos que intencionalmente amenazan su confiabilidad. (McGraw, 2006).

3.3.1 Propiedades de un software seguro

El software que se desarrolla teniendo en cuenta la seguridad, debe contar con una serie de propiedades a lo largo de su ciclo de vida (desde su desarrollo) que se interpreten y limiten con base a los requerimientos reales, tales como definir cuál es el nivel requerido de seguridad; cuáles son los aspectos más críticos a lo que se deben atender; y qué acciones resultan aptas para el costo y la programación del proyecto. (IATAC & DACS, 2007).

De acuerdo con Castellaro et al (2009), estas propiedades se pueden dividir en dos conjuntos: fundamentales, las que conforman la base de una aplicación segura (seguridad de la información) y conducentes, aquellas que permiten caracterizar cuán seguro es el software y están influenciadas por el tamaño, la complejidad y la trazabilidad del software.

3.3.1.1 Propiedades fundamentales

- Confidencialidad: El software debe garantizar que sus características, los activos de información que administra y su contenido sean accesibles sólo para las entidades autorizadas e inaccesibles para el resto.
- Integridad: El software debe ser resistente y flexible a las modificaciones no autorizadas del código, los activos administrados, la configuración o el comportamiento por parte de entidades no autorizadas.
- Disponibilidad: El software debe estar accesible siempre que sea requerido; y funcionar con un rendimiento adecuado para que los usuarios puedan realizar sus tareas en forma correcta y dar cumplimiento a los objetivos de la organización que lo utiliza.
- Trazabilidad: Todas las acciones relevantes efectuadas por los usuarios del software, se deben registrar con el fin de establecer responsabilidades.
- No Repudio: Prevenir que un usuario del software desmienta o niegue la responsabilidad de acciones que ha realizado. Asegura que no se pueda subvertir la propiedad de Trazabilidad.

3.3.1.2 Propiedades conducentes

- Exactitud (Correcto): Asegurar que el software siempre opere de la manera esperada sin defectos ni debilidades. La intencionalidad y el impacto resultante en caso de un

ataque determinan si un defecto o una debilidad realmente constituyen una vulnerabilidad que incrementa el riesgo de la seguridad.

- Previsibilidad: Las funcionalidades, comportamientos y condiciones (ambiente, entradas) del software siempre serán los que se espera que sean; así, el software nunca se desviará de su operación correcta bajo condiciones previstas. Asimismo, esta propiedad se extiende a la operación bajo condiciones no previstas, en las que si los atacantes intentan explotar fallas en el software o en su ambiente no obtendrán respuesta positiva del mismo.
- Confiabilidad: Preservar la ejecución predecible y correcta del software a pesar de la existencia de defectos no intencionales y otras debilidades, y de cambios de estado no predecibles en el ambiente.
- Protección: Capacidad de que el software se detenga o quede parcialmente operativo en un estado seguro de tal forma que persista su confiabilidad y se eviten daños mayores como pérdida de activos valiosos.

3.4 Metodologías enfocadas al desarrollo de software seguro

La construcción de software seguro sigue siendo en gran medida una cuestión de directrices, mejores prácticas y experiencias de expertos no documentada. Las prácticas actuales proporcionan orientación para áreas concretas, como el modelado de amenazas, la gestión de riesgos o la codificación segura. Según Figueroa (2016), El ciclo de vida de desarrollo de software seguro*, "es un conjunto de principios de diseño y buenas prácticas a implantar en el ciclo de vida de desarrollo de software (SDLC), para detectar, prevenir y corregir los defectos de seguridad en el desarrollo y adquisición de aplicaciones, de forma que se obtenga software de confianza y robusto frente a taques maliciosos, que realice solo las funciones para las que fue diseñado, que esté libre de vulnerabilidades, ya sean intencionalmente diseñadas o accidentalmente insertadas durante su ciclo de vida y se asegure su integridad, disponibilidad y confidencialidad*".

En el presente caso de estudio se expondrán tres metodologías que proporcionan un amplio conjunto de actividades que cubren un amplio espectro del SDLC: MS SDL, CLASP y NIST 800-64.

3.4.1 Microsoft Security Development Lifecycle MS SDL:

MS SDL es un proceso de desarrollo de software que ayuda a los desarrolladores a crear software más seguro y orienta el cumplimiento de los requisitos de seguridad, reduciendo al mismo tiempo los costes de desarrollo (Microsoft, 2017).

Microsoft definió SDL para abordar los problemas de seguridad que frecuentemente enfrentaban en muchos de sus productos, de allí que su objetivo principal sea incrementar la calidad de la funcionalidad del software mejorando su postura en seguridad. Para ello, MS SDL comprende un conjunto de actividades, que complementan el proceso de desarrollo y que van desde la etapa de entrenamiento, pasando por análisis estático y dinámico, y pruebas de código, hasta contemplar un plan de respuesta a incidentes.

Etapas de MS SDL

MS SDL presenta un proceso que consta de 7 etapas y 17 prácticas. A continuación se puede observar el flujo de fases en la metodología en donde cabe resaltar la existencia de una etapa de entrenamiento previo al uso de la misma y una etapa posterior al desarrollo orientada al soporte de incidentes.

1. TRAINING	2. REQUIREMENTS	3. DESIGN	4. IMPLEMENTATION	5. VERIFICATION	6. RELEASE	7. RESPONSE
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	Execute Incident Response Plan
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

Figura 3. Proceso de desarrollo de software MS SDL (Microsoft, 2017)

A. Entrenamiento

El entrenamiento es una práctica prerrequisito para implementar MS SDL. Se deben establecer los criterios de capacitación. El contenido que debe abarcar conceptos fundamentales como: diseño y desarrollo seguro, pruebas de seguridad, modelado de amenazas, y privacidad en el desarrollo de software. La frecuencia mínima de entrenamiento: MS SDL sugiere que los miembros del equipo de desarrollo de software que

cumplen con roles técnicos (programadores, probadores y administradores de programadores) deben asistir como mínimo una vez al año a una clase de formación en los temas mencionados. Y se deben establecer los umbrales mínimos aceptable de formación del equipo.

B. Requerimientos

En esta etapa, el equipo de desarrollo es asistido por un consultor de seguridad para establecer los requisitos mínimos de seguridad y privacidad, revisar los planes, hacer recomendaciones para cumplir con las metas de seguridad de acuerdo al tamaño del proyecto, complejidad y riesgo, despliega un sistema de seguimiento de errores y asignación de trabajo. El equipo de desarrollo identifica como será integrada la seguridad en el ciclo de vida, identifica los objetivos clave de seguridad, la manera que se integrará el software en conjunto y verificarán que ningún requerimiento pase desapercibido.

C. Diseño

En el diseño se identifican los requerimientos y la estructura del software. Se define y documenta una arquitectura segura y se identifican los componentes críticos para la seguridad, aplicando el enfoque de privilegios mínimos y la reducción del área de ataques. "Durante el diseño, el equipo de desarrollo conduce un modelado de amenazas a un nivel meticuloso de componente por componente, detectando los activos que son administrados por la aplicación y los medios por los cuales se pueden acceder a ellos. El proceso del modelado identifica las amenazas que potencialmente podrían causar daño a algún activo, establece la probabilidad de ocurrencia y fija medidas para mitigar el riesgo" (Brito, 2013).

D. Implementación

Durante la fase de implementación, se deben utilizar herramientas aprobadas. Es útil publicar y mantener actualizada una lista de herramientas y chequeos de seguridad asociados a las mismas, esto ayuda a automatizar y aplicar las prácticas de seguridad fácilmente y permite la inclusión de nuevas funcionalidades y protecciones. Asimismo, se debe evitar usar API's (Application Programming Interface) que se cataloguen como

inseguras y se recomienda que la codificación sea regida por estándares, para evitar la inyección de fallas que conlleven a vulnerabilidades de seguridad. Analizar el código fuente antes de la compilación proporciona un método de seguridad de revisión del código y ayuda a garantizar que se sigan las políticas de codificación segura.

E. Verificación

En esta etapa, se debe realizar la verificación del rendimiento en tiempo de ejecución de la funcionalidad del software, empleando herramientas que supervisan el comportamiento de la aplicación por daños en la memoria, problemas de privilegios de usuario y otros problemas críticos de seguridad. También se deben inducir fallos al programa mediante la introducción deliberada de datos malformados o aleatorios, esto ayuda a revelar posibles problemas de seguridad antes del lanzamiento y requiere una inversión baja de recursos.

F. Lanzamiento

Para el lanzamiento, se debe preparar un plan de respuesta a incidentes para ayudar a enfrentar las nuevas amenazas que pueden surgir con el tiempo. Por otra parte, la revisión de todas las actividades de seguridad que se desarrollaron ayuda a asegurar la disponibilidad de la versión del lanzamiento. Esta revisión suele incluir la verificación de modelos de amenazas, salidas de herramientas y desempeño en contra de las puertas de calidad y barras de errores definidas durante la fase de requisitos. “La certificación del software antes de un lanzamiento ayuda a garantizar que los requisitos de seguridad y privacidad se cumplieron. Archivar todos los datos pertinentes es esencial para realizar tareas de mantenimiento posteriores a la liberación y ayuda a reducir los costos a largo plazo asociados con la ingeniería de software sostenida”. (Microsoft, 2017).

G. Respuesta

Se debe estar en la capacidad de implementar el plan de respuesta a incidentes instituido en la fase de lanzamiento para ayudar a proteger a los clientes de vulnerabilidades de seguridad o privacidad de software que puedan surgir.

3.4.2 OWASP CLASP (Comprehensive Lightweight Application Security Process)

CLASP Fue aportado y revisado por varias compañías de seguridad líderes del consorcio OWASP (Open Web Application Security Project). CLASP es un modelo prescriptivo, basado en roles y buenas prácticas, que permite a los equipos de desarrollo implementar seguridad en cada una de las fases del SDLC en forma estructurada, medible y repetible. Está diseñado para ser fácil de adoptar y efectivo a la vez. (OWASP, 2009).

Estructuralmente, CLASP se descompone en 5 elementos:

A. CLASP View: 5 Perspectivas de alto nivel, que se desglosan en actividades que a su vez contienen componentes del proceso que se interconectan entre sí: Conceptos, roles, evaluación de actividades, implementación de actividades, vulnerabilidades.
B. CLASP Best Practices: Agrupación de 7 actividades de Seguridad: Programas institucionales de sensibilización, evaluar el rendimiento de las aplicaciones, obtener los requerimientos de seguridad, implementar prácticas de desarrollo seguro, construir procedimientos de solución de vulnerabilidades y publicar guías operacionales de seguridad.
C. CLASP Activities: 24 actividades diseñadas para permitir una fácil integración entre actividades de seguridad y el SDLC.
D. CLASP Resources: Ayudan a la planificación, ejecución y cumplimiento de las actividades relacionadas con la seguridad del software.
E. CLASP Taxonomy: Clasificación de alto nivel de 104 tipos de problemas o vulnerabilidades, divididos en 5 categorías de alto nivel: Errores de tipo y rangos, problemas del entorno, errores de tiempo y sincronización, errores de protocolos, y errores generales de lógica.

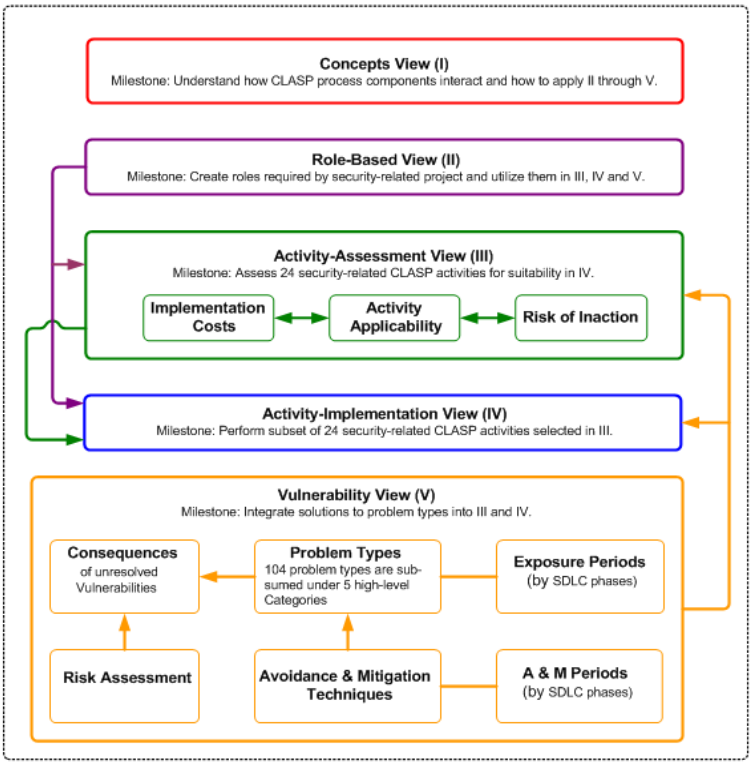

Figura 4. CLAPS View y sus interacciones. (OWASP, 2009).

3.4.3 NIST 800-64 Consideraciones de seguridad en el ciclo de vida de desarrollo de sistemas.

La publicación especial (SP) 800-64 del Instituto Nacional de Estándares y Tecnología (NIST) tiene como objetivo ayudar a las agencias del gobierno federal a integrar las actividades esenciales de seguridad en la metodología del SDLC que tienen establecida. (NIST, 2008).

Esta guía describe las funciones y responsabilidades clave de seguridad que se necesitan en el desarrollo de la mayoría de los sistemas de información identificando consideraciones de seguridad para cada fase (iniciación/análisis, adquisición/diseño, implementación/evaluación, operación/mantenimiento, disposición) del SDLC bajo los siguientes parámetros:

- Descripción de la fase.
- Identificación general de las puertas de control: Las puertas de control son valiosas ya que proporcionan a la organización la oportunidad de verificar que se están

abordando las consideraciones de seguridad, se está construyendo una seguridad adecuada y los riesgos identificados se comprenden claramente antes de que el desarrollo del sistema avance a la siguiente fase del ciclo de vida.

- Identificación y descripción de las actividades mayores de seguridad, incluyendo la descripción de la actividad en sí misma, los resultados esperados, el procedimiento de sincronización con otras actividades, y la interdependencia con otras tareas claves.

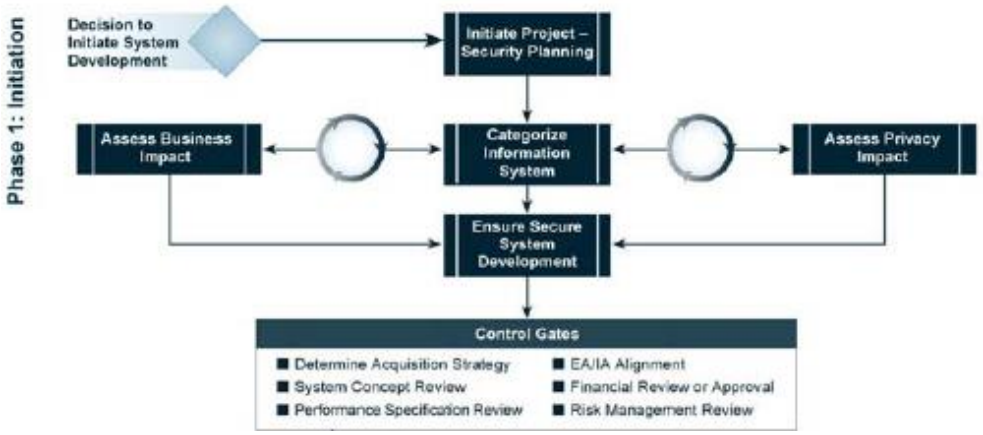

Figura 5. Fase de iniciación NIST 800-64. (NIST, 2008).

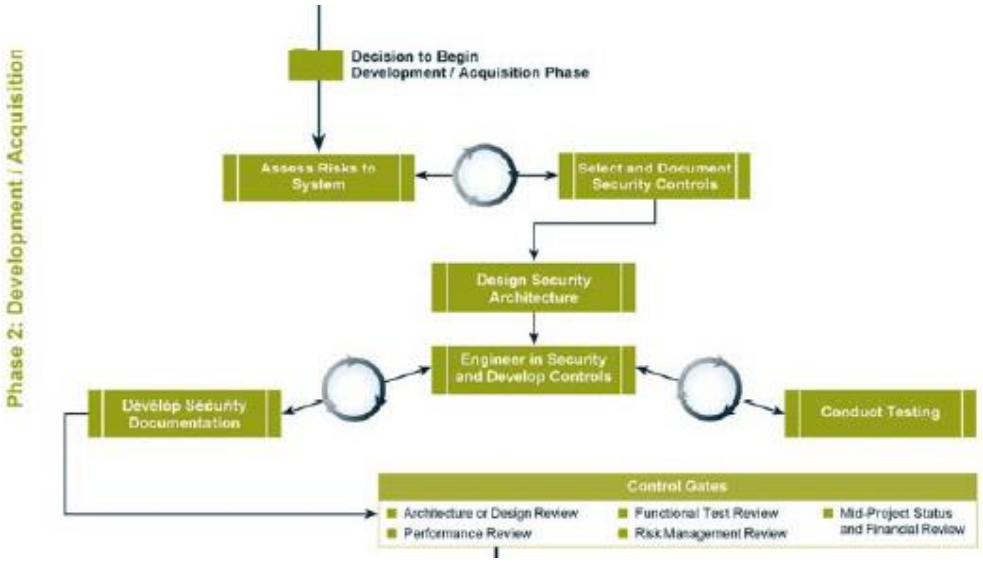

Figura 6. Fase de adquisición NIST 800-64. (NIST, 2008).

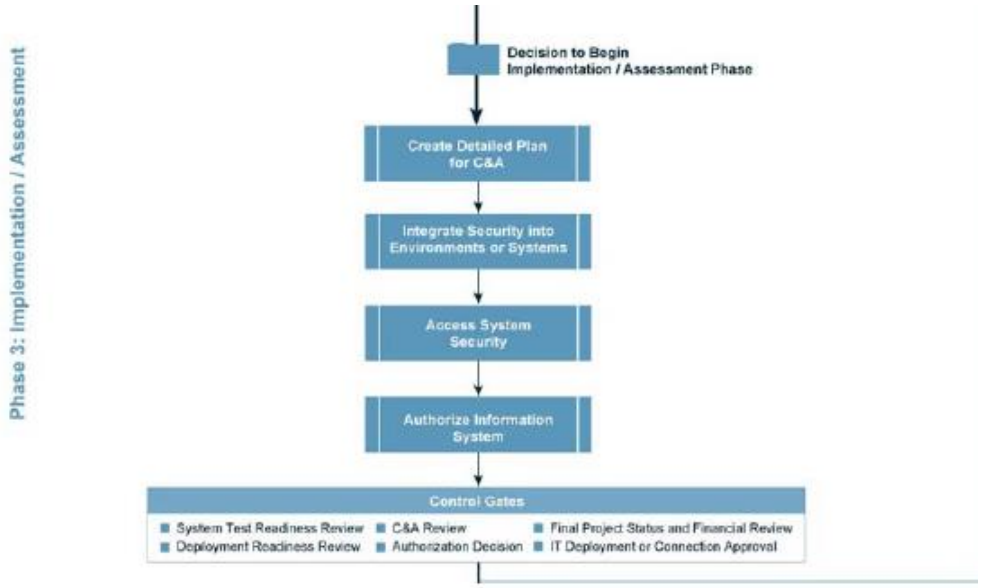

Figura 7. Fase de implementación NIST 800-64. (NIST, 2008).

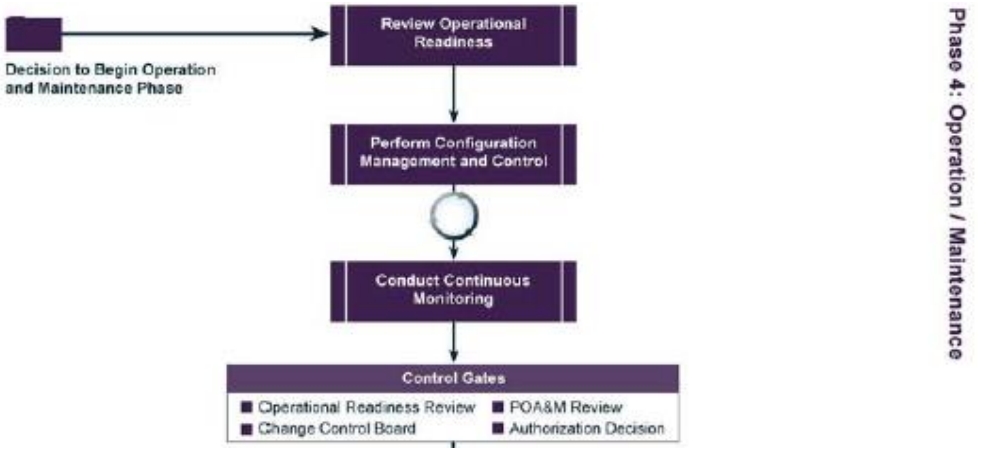

Figura 8. Fase de despliegue NIST 800-64. (NIST, 2008).

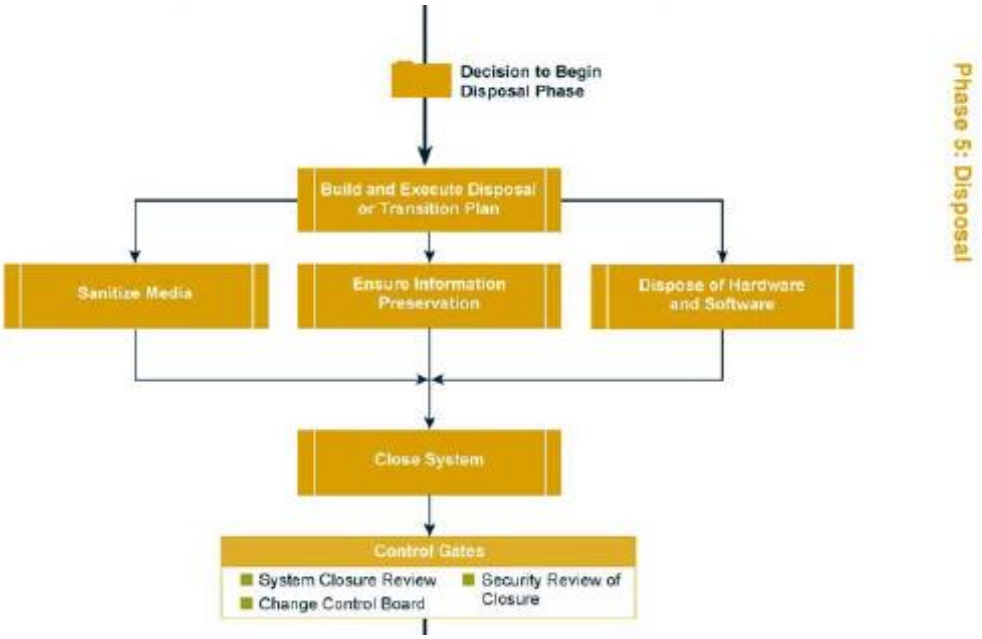

Figura 9. Fase de disposición NIST 800-64. (NIST, 2008).

3.5 Análisis de riesgos

"Las organizaciones de todo tipo y tamaño enfrentan factores e influencias, internas y externas, que crean incertidumbre sobre si ellas lograrán o no sus objetivos. El efecto que esta incertidumbre tiene en los objetivos de una organización es el riesgo". (NTC-ISO31000, 2011).

Otra definición de riesgo es catalogarlo como la "estimación del grado de exposición a que una amenaza se materialice sobre uno o más activos causando daños o perjuicios a la Organización" (MAGERIT, 2012). De allí entonces que el proceso de análisis de riesgos hace referencia a estimar la magnitud de los riesgos a los que está expuesta una organización, un procedimiento, una actividad y/u otro proceso.

Conociendo entonces que un riesgo puede causar daños, se deben tomar acciones para tratarlo, y para ello hay múltiples formas: evitar sus causas, reducir las posibilidades de que ocurra, limitar sus consecuencias, compartirlo con otra organización, o, en última instancia, aceptar que puede ocurrir y prever recursos para actuar cuando sea necesario. El análisis de riesgos proporciona entonces un modelo de un sistema en términos de activos, amenazas y controles, y es el fundamento para el manejo apropiado de todas las actividades; y la etapa

de tratamiento define las acciones de seguridad necesarias para satisfacer las falencias detectadas por el análisis.

El análisis de riesgos tiene en cuenta los siguientes elementos:

- Activos: elementos del sistema de información que soportan la misión de la organización. Cualquier cosa que tenga valor para la organización. (NTC-ISO/IEC 27001).
- Amenazas: Causa potencial de un incidente no deseado, que puede ocasionar daño a un sistema o a la Organización. (CELSIA, 2013).
- Vulnerabilidad: Debilidad de un activo o grupo de activos, que puede ser aprovechada por una o más amenazas. (CELSIA, 2013)
- Controles: Medidas de protección desplegadas para que las amenazas no causen (tanto) daño al explotar las vulnerabilidades.

Y con estos elementos se puede estimar el nivel de oportunidad de que un evento se presente (Probabilidad) y el nivel de afectación que puede ocasionar la materialización del riesgo (Impacto). (NTC-ISO/IEC 31000).

De acuerdo con MAGERIT (2012), el análisis de riesgos permite analizar la probabilidad y el impacto de forma metódica para llegar a conclusiones con fundamento y proceder a la fase de tratamiento. En otras palabras, se puede plantear que el análisis de riesgos permite racionalizar la gestión de la seguridad de un sistema de información.

3.5.1 MAGERIT: Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información

Creado por el Ministerio de Administraciones Públicas de España, como el marco general para la gestión de riesgos en sistemas de información para que los órganos de gobierno tomen decisiones teniendo en cuenta los riesgos derivados del uso de tecnologías de la información. MAGERIT aborda la gestión de riegos en su fase de identificación y evaluación y también aborda recomendaciones para la fase previa de identificación de activos así como para el establecimiento de controles de seguridad. Actualmente se encuentra disponible en su versión 3 y ofrece tres documentos de libre descarga (método,

catálogo de elementos, guía de técnicas) y el software PILAR que es una herramienta que implementa la metodología MAGERIT.

Dentro de los objetivos de esta metodología se encuentran:

1. Concienciar a los responsables de las organizaciones de información de la existencia de riesgos y de la necesidad de gestionarlos.
2. Ofrecer un método sistemático para analizar los riesgos derivados del uso de tecnologías de la información y comunicaciones (TIC).
3. Ayudar a descubrir y planificar el tratamiento oportuno para mantener los riesgos bajo control.
4. Preparar a la organización para procesos de evaluación, auditoría, certificación o acreditación.

Uno de los factores que diferencia a MAGERIT de otras metodologías es que los resultados de evaluación se obtienen y expresan en valores económicos, lo que permite que la presentación de planes de tratamiento y propuestas a la dirección se encuentre mejor formadas y fundamentadas permitiendo así explicar y lograr mejores resultados.

MAGERIT se fundamenta en la estructura de gestión de riesgos propuesta por ISO 31000:

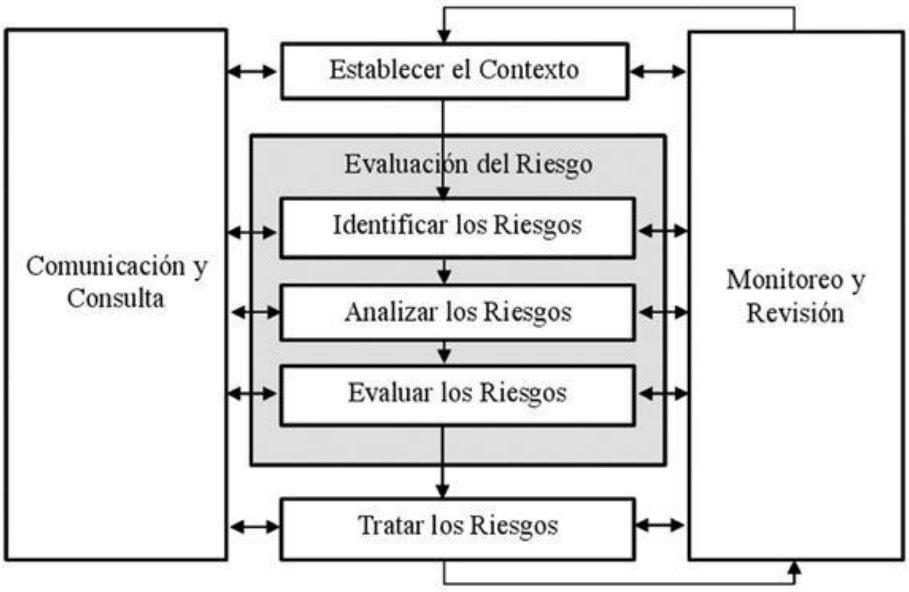

Figura 10. Proceso de gestión de riesgos. (MAGERIT, 2012).

Establecer el contexto permite determinar los parámetros y condicionantes externos e internos que orientan la política para gestionar los riesgos, en esta etapa cabe la identificación de los activos. La identificación de los riesgos busca una relación de las vulnerabilidades y amenazas partiendo de sus posibles causas. El análisis de los riesgos busca calificar los riesgos identificados, cuantificando sus consecuencias (análisis cuantitativo) u ordenando su importancia relativa (análisis cualitativo). La evaluación de los riesgos es más profunda que el análisis y transforma las consecuencias a términos del negocio. Aquí se decide cuales riesgos se aceptan y cuáles no, así como en qué circunstancias se pueden o no aceptar un riesgo y/o trabajar en su tratamiento. El tratamiento de los riesgos busca modificar la situación del riesgo. En cuanto a la comunicación y consulta, se debe buscar el equilibrio entre la seguridad y el soporte a la productividad de la organización contando para ello con todas las partes interesadas (Stakeholders). Entre tanto para el seguimiento y revisión es importante resaltar que se debe mantener en un entorno de crecimiento y mejora continua.

4 METODOLOGÍA

El presente proyecto de desarrollo tecnológico se orienta a la obtención de guía de buenas prácticas que sea aplicable a la metodología de desarrollo ágil SCRUM para fortalecer la seguridad de la información, siguiendo un método teórico de análisis y síntesis en el cual se llevará a cabo una etapa de indagación y consulta sobre software seguro que abarque definiciones y metodologías; y sobre marcos de trabajo para el análisis de riesgos específicamente MAGERIT. Posteriormente, se tendrá un periodo de análisis para identificar zonas de convergencia entre las metodologías de desarrollo seguro y establecer cuáles son las prácticas aplicables a un ciclo de vida ágil, así como se establecerán las vulnerabilidades y amenazas en la compañía New Soft en el desarrollo de sus productos. Finalmente se tendrá una etapa de definición y resultados, en la cual se construirá el comparativo de las tres metodologías de desarrollo de software seguro en aras de identificar mejores prácticas de seguridad de la información que se puedan aplicar a metodologías de desarrollo ágil; el análisis de riesgos del modelo de desarrollo ágil SCRUM en la compañía New Soft con el fin de identificar las vulnerabilidades, amenazas y/o errores comunes que llevan a fallos de seguridad en las aplicaciones, empleando para ello el marco de trabajo MAGERIT; y la guía con un modelo de buenas prácticas de seguridad que sea aplicable a la metodología de desarrollo ágil SCRUM para fortalecer la seguridad de la información durante las diferentes etapas del ciclo de vida de desarrollo de software.

5 RESULTADOS Y DISCUSIÓN

5.1 Comparativo de metodología de desarrollo de software seguro

Se decide comparar las metodologías MS SDL, CLASP y NIST 800-64, porque en lo que respecta a la exhaustividad, las tres proporcionan un amplio conjunto de actividades que cubren un amplio espectro del ciclo de vida del desarrollo. Asimismo cada una se enfoca en un tipo diferente de organización, MS SDL se considera ser más pesado y riguroso, haciéndolo más conveniente para las organizaciones grandes, CLASP, por otro lado, es ligero y más asequible para las pequeñas organizaciones con demandas de seguridad menos estrictas, y NIST 800-64 se proyecta para organizaciones gubernamentales. Otro punto de decisión fue la disponibilidad de documentación completa de las tres metodologías.

En el marco teórico se presentaron las características generales de las metodologías mencionadas delimitando su filosofía, en el Anexo A se identifican los puntos de convergencia y divergencia entre ellas, en orden de articular los puntos fuertes y débiles, validando que se cumplan con las propiedades de un software seguro y estableciendo cuales mejores prácticas son aplicables al desarrollo ágil en cada fase del SDLC: Educación y sensibilización; Inicio del proyecto (Requisitos); Análisis; Diseño; Implementación, pruebas y verificación; Despliegue y soporte.

5.2 Análisis de riesgos del proceso de desarrollo de software en New Soft

El análisis de riesgos del proceso de desarrollo de software en la compañía New Soft, la cual utiliza la metodología ágil SCRUM, tiene como finalidad identificar las vulnerabilidades y amenazas que llevan a los fallos de seguridad en las aplicaciones producidas por la organización, y sirve como fundamento para la toma de decisiones por parte de los directivos en cuanto al establecimiento de medidas y/o controles para mitigar los fallos detectados y mejorar la cadena de producción. Asimismo es una entrada para la elección de las buenas prácticas que son necesarias de aplicar en la metodología de desarrollo utilizada por la empresa en aras de fortalecer la seguridad de la información.

Se presenta entonces en el Anexo B, el empleo del método de análisis propuesto en el modelo MAGERIT v3.0, el cual sigue las siguientes etapas:

- Caracterización de los activos.
- Caracterización de las amenazas.
- Caracterización de las salvaguardas.
- Estimación del estado de riesgo.

5.3 Guía de buenas prácticas aplicable a la metodología de desarrollo ágil SCRUM para fortalecer la seguridad de la información.

A partir de la descripción de las características de un software seguro, el estudio de las metodologías MS SDL, CLASP y NIST 800-64, y el análisis de riesgos al proceso de desarrollo de software en la compañía New Soft, se logró la identificación de falencias comunes y repetitivas en el ciclo de vida de desarrollo de software que se lleva a cabo siguiendo el método ágil SCRUM. Asimismo se identificaron buenas prácticas con las cuales se plantea una capa aplicable a la metodología SCRUM que permita mejorar los controles de seguridad existentes en las aplicaciones, segmentando los mismos según los actores, eventos y artefactos dentro de cada proyecto, y se pueden sintetizar así:

- Rol de responsable de seguridad.
- Formación en seguridad para los equipos de desarrollo.
- Diseño de casos de abuso.
- Auditoria de seguridad
- Verificación de seguridad

En el anexo C se presentan los anteriores elementos, con los cuales se pretende preservar la confidencialidad, integridad y disponibilidad de las aplicaciones desarrolladas y la información contenida y administrada por ellas mediante la incorporación de procesos operativos y tácticos influenciados por las necesidades de los equipos de desarrollo, la organización y sus objetivos.

Dado el tiempo de desarrollo y entrega del presente proyecto, se plantea como desarrollo futuro incorporar tareas específicas a cada uno de los pasos propuestos.

6 CONCLUSIONES

- La seguridad debe ser considerada en todas las capas de un proyecto de software, incluyendo infraestructura, diseño, codificación y pruebas, sin dejar de lado a los usuarios y su interacción con el sistema.

- El software seguro es el resultado de múltiples actividades que incluyen un cambio cultural en las organizaciones, equipos de desarrollo y clientes. Por lo cual surge la necesidad de incrementar la inversión en la protección de las aplicaciones y adoptar estrategias de seguridad para el desarrollo que involucre a mencionados actores.

- La inclusión de un experto de seguridad como un rol central dentro de los proyectos es importante porque orienta a todos los involucrados desde el punto de vista de este requerimiento no funcional, emitiendo recomendaciones a considerar en el modelado, diseño, arquitectura, codificación y pruebas. Lo cual permite atender el aspecto de seguridad de forma iterativa en cada incremento de software que se desarrolle.

- El análisis de riesgos es un factor común aplicable a diferentes ramas y en este caso a cualquier metodología de desarrollo de software, porque es imperativo conocer cuáles son los activos críticos que van a controlar las aplicaciones y cuáles son las amenazas asociadas a los mismos.

- Cada compañía dedicada el desarrollo de software debe seleccionar e implementar los modelos que mejor se adapten a sus características para garantizar la protección de los principios de la seguridad de la información: Confidencialidad, Integridad y Disponibilidad, en sus productos.

7 BIBLIOGRAFÍA

Beck, K et al. (2001). Manifesto for Agile Software Development. Recuperado de: http://agilemanifesto.org/

Boehm, B. & Papaccio, P. (1988). Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering, volume (14), 1462–1477.

Brito, C. (2013). Metodologías para desarrollar software seguro. ReCIBE, Volumen 2. Recuperado de: http://recibe.cucei.udg.mx/revista/es/vol2-no3/pdf/computacion05.pdf

Castellaro, M. & Ramos, J. & Romaniz, S. & Pessolani, P. (2009). Hacia la ingeniería del software seguro. Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/21332/Documento_completo.pdf?sequence=1

Celsia S.A. (2013). Política de tecnologías de información y comunicación TIC. Recuperado de: http://www.celsia.com/

Chile Ágil (2016). El estado de la seguridad ágil. Recuperado de: https://foro.chileagil.cl/t/el-estado-de-la-seguridad-agil-lean-respaldo-del-blog/1140

CISCO (2016). Informe anual de seguridad. Recuperado de: http://www.cisco.com/c/dam/m/es_es/internet-of-everything-ioe/iac/assets/pdfs/security/cisco_2016_asr_011116_es-es.pdf

Figueroa, V. (2016). Secure Software Development Life Cycle. Recuperado de: https://www.owasp.org/images/9/9d/OWASP-LATAMTour-Patagonia-2016-rvfigueroa.pdf

Fowler, M., & Highsmith, J. (2001). El manifiesto ágil. Recuperado de: http://www.pmp-projects.org/Agile-Manifesto.pdf

Garbajosa, J., & Rodríguez P., & Yagüe, A. (2008). La implementación de métodos ágiles: Ventajas y problemas. Universidad Politécnica de Madrid. Recuperado de: https://www.ati.es/IMG/pdf/UPM08.pdf

Gilb, T. (1988). Principles of Software Engineering Management. Boston: Adison-Wesley.

Graham, D. (2016). Introduction to the CLASP Process. Recuperado de: https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/introduction-to-the-clasp-process

Hewlett Packard (2016). HPE Security Research. Cyber Risk Report 2016. Recuperado de: https://ssl.www8.hp.com/ww/en/secure/pdf/4aa6-3786enw.pdf

IATAC & DACS. (2007). Software security assurance. Recuperado de: https://www.researchgate.net/publication/279351339_Software_Security_Assurance_A_State-of-Art_Report_SAR

Jones, C. (1986). Programming Productivity. New York: McGraw-Hill.

López, G., Menzinsky, A., Palacio, J. (2016). Scrum Manager. Recuperado de: http://www.scrummanager.net/files/scrum_manager.pdf

MAGERIT. (2012). MAGERIT v3. Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información. Recuperado de: https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Magerit.html

McGraw, G. (2004). Software security. Recuperado de: https://www.cigital.com/papers/download/software-security-gem.pdf

McGraw, G (2006). Software security: Building security in. Boston, EEUU: Addison Wesley.

Microsoft (2016). Tendencias en seguridad informática 2016. Recuperado de: https://info.microsoft.com/rs/157-GQE-382/images/ES-XL-CNTNT-ebook-Security-Trends-in-Cybersecurity.pdf

Microsoft (2017). SDL. Recuperado de: https://www.microsoft.com/en-us/sdl/

NIST. (2008). NIST SP 800-64 Security Considerations in the System Development Life Cycle. Recuperado de: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf

NTC-ISO. (2011). ISO-31000 Gestión del riesgo. Bogotá D.C., Colombia: ICONTEC.

NTC-ISO. (2013). ISO 27001 Sistemas de gestión de la seguridad de la información (SGSI). Bogotá D.C., Colombia: ICONTEC.

OKHosting. (2016). Metodología del desarrollo de software. Recuperado de: https://okhosting.com/blog/metodologias-del-desarrollo-de-software/

OWASP. (2009). OWASP CLASP Project. Recuperado de: https://www.owasp.org/index.php/Category:OWASP_CLASP_Project/es

Patiño, J. (2013). Incorporación de la seguridad en el proceso unificado ágil para el desarrollo de software (Tesis maestría). Universidad Nacional Autónoma de México, México D.F. Recuperado de: http://132.248.9.195/ptd2013/julio/0697700/0697700.pdf

Pressman, R. (Ed.7). (2010). Ingeniería del software un enfoque práctico. México D.F, México: McGraw Hill.

Radigan, D. (2017). Kanban. Aplicación de la metodología Kanban en el desarrollo de software. Recuperado de: https://es.atlassian.com/agile/kanban

SafeCreative (2008). Introducción al modelo Scrum para desarrollo de Software. Recuperado de: http://www.safecreative.org/work/0803130492765

Vigo, G. (2015). Seguridad Ágil (Tesis pregrado). Universidad de Barcelona, Barcelona-España. Recuperado de: http://diposit.ub.edu/dspace/bitstream/2445/68455/2/memoria.pdf

8. ANEXOS

ANEXO A. Comparativo de metodología de desarrollo de software seguro

A continuación se identifican los puntos de convergencia y divergencia entre MS SDL, CLASP y NIST 800-64, en orden de articular los puntos fuertes y débiles, validando que se cumplan con las propiedades de un software seguro y estableciendo cuales mejores prácticas son aplicables al desarrollo ágil en cada fase del SDLC: Educación y sensibilización; Inicio del proyecto (Requisitos); Análisis; Diseño; Implementación, pruebas y verificación; Despliegue y soporte.

1. Educación y sensibilización

Actividad	MS SDL	CLASP	NIST 800-64
Entrenamiento en seguridad	✓	✓	✓

Todas reconocen que la educación es muy importante. Todos en el equipo deben por lo menos recibir la educación inicial, que consiste en concientizar la importancia de la seguridad, y conocer sobre los fundamentos de la ingeniería de seguridad incluyendo los conceptos básicos de seguridad, los tipos de fallos de seguridad, las posibles soluciones, etc. Como la seguridad es un campo en rápida evolución, con nuevas amenazas emergiendo con frecuencia, la educación periódica es crucial para todos los involucrados en el proyecto. MS SDL y NIST 800-64 se centran en la formación de los desarrolladores, mientras que CLASP reconoce que el entrenamiento debe ser recibido por todos los roles del proyecto incluyendo el gerente del mismo.

2. Inicio del proyecto (Requisitos)

Actividad	MS SDL	CLASP	NIST 800-64
Incluir expertos en seguridad	✓	✓	✓

Establecer los requerimientos de seguridad	✓	✓	✓
Evaluar los riesgos de seguridad y privacidad	✓		✓
Construir política de seguridad		✓	✓
Especificar ambiente operacional	✓		✓
Definir métricas de seguridad	✓	✓	✓
Establecer responsabilidades de seguridad		✓	

Las tres metodologías reconocen que para construir seguridad en los proyectos de software es necesario involucrar personal experto en seguridad para que ayude a los desarrolladores en la definición de los requisitos de seguridad, la definición del ambiente operacional y en sí con los problemas relacionados con la seguridad. Otro punto en común se presenta en la definición de métricas de seguridad o puertas de calidad, lo cual permite establecer niveles de calidad mínimos aceptables de seguridad y privacidad, comprender los riesgos asociados a los problemas de seguridad, identificar y solucionar errores de seguridad durante el desarrollo y aplicar los estándares en todo el proyecto.

MS SDL y NIST 800-64 plantean realizar evaluaciones de los riesgos de seguridad y privacidad, lo que permite determinar el nivel de impacto de la privacidad de una característica, producto o servicio, mientras que CLASP propone identificar una política de seguridad global que defina el conjunto de requisitos de seguridad base para los proyectos de software, esto balanza la carga de los especificadores de requisitos a largo plazo, proporciona una manera de comparar la postura de seguridad de las aplicaciones dentro de la organización y puede ser un marco de responsabilidad por proyecto.

3. Análisis

Actividad	MS SDL	CLASP	NIST 800-64
Analizar requerimientos de seguridad		✓	✓
Identificar límites de confianza		✓	
Identificar roles de usuario		✓	
Documentar requerimientos de seguridad		✓	✓

MS SDL no establece actividades específicas para la fase de análisis. Por su parte CLASP y NIST 800-64 se centran en el análisis de los requerimientos de seguridad, la identificación de los roles de los usuarios, el establecimiento de los límites de confianza y la documentación de los requerimientos de seguridad. Los límites de confianza indican dónde interactúan entidades confiables y no confiables. Por otra parte, la especificación de requisitos de seguridad es importante tanto para descubrir problemas de seguridad desde el principio en el SDLC como para guiar actividades de construcción específicas de seguridad más adelante.

4. Diseño

Actividad	MS SDL	CLASP	NIST 800-64
Diseñar arquitectura de seguridad	✓		✓
Realiza análisis de la superficie de ataque	✓	✓	
Usar modelado de amenazas	✓	✓	
Detallar casos de abuso		✓	
Aplicar principios de seguridad al diseño		✓	✓

El esquema de integración de la seguridad provee detalles sobre en donde, dentro del sistema, la seguridad se aplica y se comparte. La arquitectura de la seguridad se debe detallar de tal forma que se pueda observar en dónde y cómo se aplican los controles básicos de seguridad. Tanto MS SDL como CLASP, proponen realizar un análisis de la superficie de ataque para reducir las oportunidades de que posibles atacantes exploten puntos débiles o vulnerabilidades, esto incluye inhabilitar o restringir el acceso a los servicios del software, aplicando el principio de privilegios mínimos y empleando defensas estratificadas siempre que sea posible. Por otra parte, el modelado de amenazas ayuda al equipo de desarrollo a identificar de manera más eficaz y menos costosa vulnerabilidades de seguridad, determinando los riesgos de esas amenazas y mecanismos para una mitigación apropiada. CLASP también propone determinar casos de abuso. Los casos de abuso son idénticos a los casos de uso, excepto que están destinados a detallar los intentos comunes de abuso del sistema. Al igual que los casos de uso, los casos de abuso deben ser

diseñados para cada actor y estos actores deben ser asignados a las capacidades del sistema. Con la aplicación de principios de seguridad en el diseño, se busca fortalecer la aplicación, determinar estrategias de implementación de los servicios de seguridad y diseñar protocolos y API's seguros.

5. Implementación, pruebas y verificación

Actividad	MS SDL	CLASP	NIST 800-64
Usar herramientas certificadas	✓		
Despreciar funciones inseguras	✓		
Realizar análisis estático y dinámico	✓		
Implementar funciones de seguridad (políticas)	✓	✓	✓
Verificar atributos de seguridad		✓	✓
Identificar, implementar y realizar pruebas de seguridad		✓	✓
Completar la documentación del sistema		✓	✓
Definir planes de contingencia			✓
Especificar la configuración de las bases de datos		✓	

Para implementar los requerimientos de seguridad, los desarrolladores deben identificar cualquier ambigüedad que surja, incluyendo cualquier información adicional necesaria para construir una funcionalidad concreta asegurando que se cumplan todas las directrices de codificación establecidas, especialmente las relacionadas con la seguridad. MS SDL propone el uso de herramientas aprobadas para la implementación así como restringir el uso de API's que puedan dar origen a vulnerabilidades. Por otro lado, las tres metodologías se centran en la realización de pruebas funcionales y de seguridad pero con un foco diferente. MS SDL y NIST 800-64 se centran más en las pruebas de caja negra, mientras que CLASP hace hincapié en las pruebas de caja blanca.

En esta etapa, tanto CLASP como NIST 800-64 proponen que se culmine con la documentación de la aplicación, esto abre el camino para poder obtener una mayor facilidad de corrección y modificación ante problemas futuros. Asimismo, NIST plantea la definición de planes de continuidad ante posibles ataques o falles inesperados del software.

6. Despliegue y soporte

Actividad	MS SDL	CLASP	NIST 800-64
Definir plan de respuesta a incidentes	✓	✓	✓
Efectuar revisión de seguridad final	✓		
Administrar la configuración del sistema			✓
Ejecutar plan de respuesta a incidentes	✓	✓	✓

Aunque las tres metodologías organizan esta última fase de forma distinta, en gran medida comparten los mismos resultados. En primer lugar, hay una necesidad de planificación operativa y preparación, que incluye la definición de los manuales de usuario, la documentación de la arquitectura de seguridad, la configuración de los servidores y opciones del sistema, entre otros. Posteriormente, es necesario que exista un plan de respuesta que indique qué hacer cuando se descubre una nueva vulnerabilidad. Las tres metodologías reconocen que la comunicación con los clientes es muy importante. Se debe garantizar que se implementen las correcciones de forma efectiva y que los clientes tengan acceso a las mismas. Finalmente, es necesario que los planes de contingencia se puedan ejecutar cada vez que se descubre una nueva vulnerabilidad.

ANEXO B. Análisis de riesgos del proceso de desarrollo de software en New Soft

1. Caracterización de los activos

1.1. Identificación de activos

Para la identificación de los activos de información se establecen los siguientes elementos:

- Código: Para numerar el activo a modo de inventario (AI: Activo de información).
- Nombre: Corto, que indique la referencia del activo.
- Descripción: Explicación de lo que abarca el activo de información
- Responsable: Quien opera el activo, como su nombre lo indica, es el responsable del activo de información en todos los aspectos.
- Ubicación: Espacial o técnica.
- Observaciones: Otras características que sean necesarias aclarar.

Código	Nombre	Descripción	Responsable	Ubicación	Observaciones
AI01	Requerimientos (Pila del producto)	Listado de requerimientos de las aplicaciones.	Scrum master Equipo de desarrollo Propietario del producto	Contrato de producto.	El cliente como parte fundamental del desarrollo
AI02	Planeación	Cronograma de actividades a desarrollar	Scrum master	Documentación de proyectos.	
AI03	Diseño	Modelado de bases de datos, definición de usuarios y roles, casos de uso de las aplicaciones	Scrum master Equipo de desarrollo	Documentación de proyectos.	
AI04	Código fuente	Código fuente de las aplicaciones desarrollado por los programadores	Equipo de desarrollo	Hardware de desarrollo	

AI05	Software de desarrollo	Software usado para la codificación, diseño gráfico, depuración, compilación y pruebas.	Gerente de TI Scrum master Equipo de desarrollo	Hardware de desarrollo.	Incluye licencias de sistemas operativos y software de apoyo como máquinas virtuales y antivirus.
AI06	Hardware de desarrollo	Equipos informáticos empleados para el desarrollo de las aplicaciones: PC, Terminales móviles, servidores.	Gerente de TI Scrum master Equipo de desarrollo	Instalaciones New Soft.	
AI07	Servidores de bases de datos	Servidores empleados para el almacenamiento de datos de las aplicaciones en entornos de pruebas y de producción	Proveedor externo Gerente de TI Scrum master	Instalaciones New Soft. Proveedores externos	
AI08	Datos de pruebas	Información utilizada para realizar pruebas de las aplicaciones	Equipo de desarrollo Scrum master Proveedores externos	Servidores de bases de datos Casos de prueba	
AI09	Datos de usuarios	Información básica de los usuarios de las aplicaciones incluyendo credenciales de acceso	Equipo de desarrollo Scrum master Proveedores externos	Servidores de bases de datos.	

AI10	Enlace de datos	Medios de transporte de información	Proveedores externos	Red de datos	
AI11	Personas	Cualidades personales y Capacidades cognitivas: conocimientos y experiencia del equipo de desarrollo en herramientas, mecanismos, lenguajes, procedimientos, entre otros.	Gerencia Recursos humanos Scrum master Equipo de desarrollo	New Soft	

1.2. Dependencia entre los activos

La dependencia entre activos permite identificar y valorar la medida en que un activo de orden superior se puede ver afectado por una amenaza materializada sobre un activo de orden inferior.

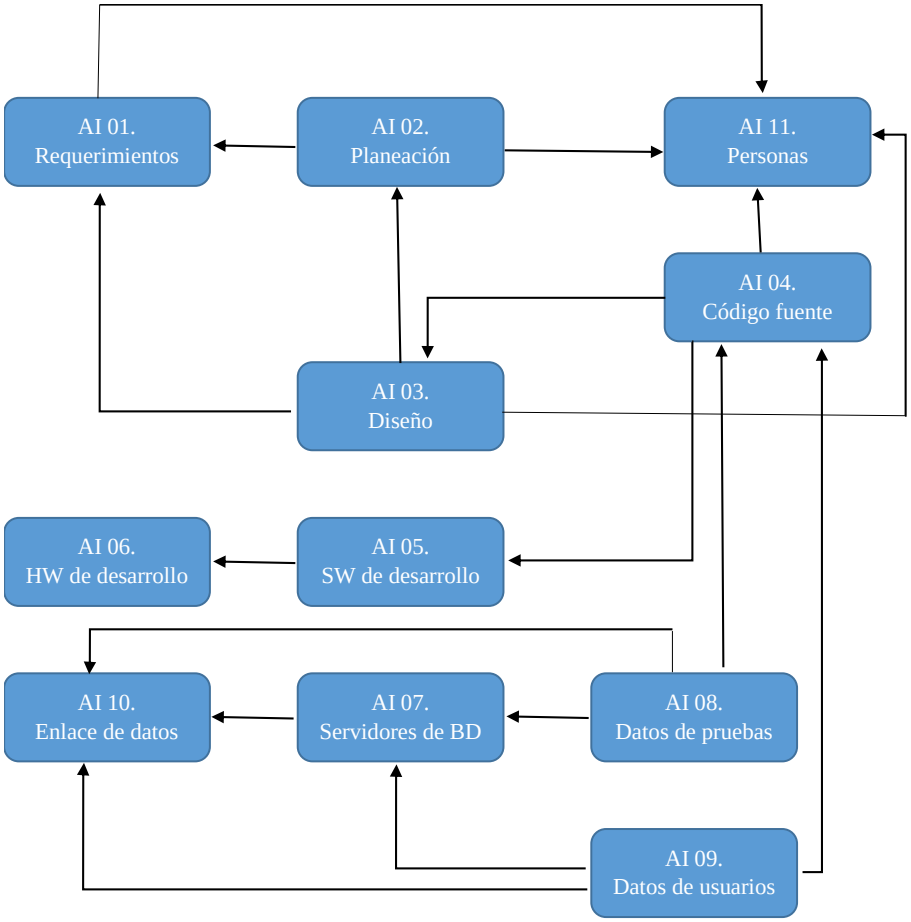

Figura 11. Dependencia de activos de información

1.3.Valoración de los activos

Para la valoración de los activos, se emplearan las dimensiones de la seguridad de la información: Disponibilidad, Integridad, y Confidencialidad. Valorando de forma cualitativa que importancia tendría la ausencia o afectación de una de estas propiedades en cada activo. Para ello se empleara una escala de 5 niveles: despreciable, bajo, medio, alto, y extremo. Los resultados obtenidos son el promedio de los valores al consultar a directivos y equipos de desarrollo de New Soft.

Valor	Nivel	Criterio
10	Extremo	Daño extremadamente grave
7-9	Alto	Daño muy grave
4-6	Medio	Daño importante
1-3	Bajo	Daño menor
0	Despreciable	Irrelevante a efectos prácticos.

Código	Activo	Disponibilidad	Integridad	Confidencialidad
AI01	Requerimientos (Pila del producto)	**10**	**9**	**3**
AI02	Planeación	**8**	**8**	**2**
AI03	Diseño	**8**	**10**	**7**
AI04	Código fuente	**10**	**10**	**9**
AI05	Software de desarrollo	**7**	**8**	**0**
AI06	Hardware de desarrollo	**7**	**8**	**0**
AI07	Servidores de bases de datos	**8**	**10**	**5**
AI08	Datos de pruebas	**6**	**5**	**6**
AI09	Datos de usuarios	**10**	**10**	**10**
AI10	Enlace de datos	**9**	**8**	**7**
AI11	Personas	**9**	**8**	**8**

2. Caracterización de las amenazas

2.1 Identificación de las amenazas

Para la identificación de amenazas, se establecieron los siguientes parámetros:

- Código: Para numerar la amenaza a modo de inventario (AM: Amenaza).
- Nombre: Corto, que indique la referencia de la amenaza.
- Tipo de activos: Referencia de los activos que puede afectar.
- Dimensiones: De seguridad, que pueden ser afectadas por la amenaza, en orden de relevancia.
- Descripción: Lo que puede llegar a ocasionar la amenaza sobre los activos, consecuencias.

A continuación se presentan las diez amenazas más significativas para New Soft en el SDLC.

Código	Nombre	Tipo de Activos	Dimensiones	Descripción (Consecuencias)
AM01	Requisitos no definidos correctamente	AI01, AI02, AI03, AI04	Disponibilidad, Integridad, Confidencialidad	No se cumplen expectativas de los clientes. Errores en el diseño e implementación. Retrasos en las entregas. Vulnerabilidades de seguridad.
AM02	La planificación no incluye tareas necesarias	AI03, AI04	Disponibilidad, Integridad, Confidencialidad	No se puede construir el producto en el tiempo determinado. Esfuerzo mayor que el estimado. Retrasos en las entregas. Vulnerabilidades de seguridad.
AM03	Diseños demasiado sencillos que no cubren requerimientos no funcionales	AI03, AI04	Integridad, Confidencialidad	Vulnerabilidades de seguridad. Implementaciones incorrectas.
AM04	Empleo de bibliotecas de código externas para implementar funcionalidades deseadas	AI04, AI07, AI08, AI09	Integridad, Confidencialidad	Comprobaciones extras. Retrasos en las entregas. Exposición de datos de usuario.

				Puertas traseras.
AM05	Omisión de patrones y estándares de diseño y desarrollo	AI04, AI07, AI08, AI09	Integridad, Confidencialidad	Código fuente vulnerable. Exposición de datos de usuario. Desarrollos erróneos. Retrasos en las entregas.
AM06	Entorno de software desconocido	AI04, AI05, AI07 AI11	Disponibilidad, Integridad, Confidencialidad	Curva de aprendizaje larga. Conflictos entre miembros del equipo. Necesidad de reorganizar el equipo. Retrasos en las entregas.
AM07	Herramientas de desarrollo no funcionan correctamente	AI06, AI07, AI04, AI05	Disponibilidad	Retrasos en las entregas. Desviación de la atención a solucionar problemas técnicos. Reducción de productividad. Desmotivación de los equipos de trabajo.
AM08	El cliente no participa de forma activa en la revisión de los sprint	AI01, AI02, AI03, AI04	Integridad, Confidencialidad	No se cumple con las expectativas de los clientes. Errores en el diseño e implementación. Retrasos en las entregas.

				Vulnerabilidades de seguridad. Pruebas insuficientes.
AM09	Falta de especialización en el personal	AI03, AI04, AI11	Disponibilidad, Integridad, Confidencialidad	Errores de implementación. Vulnerabilidades de seguridad. Retrasos en las entregas.
AM10	Casos de prueba débiles.	AI08, AI09, AI10	Disponibilidad, Integridad, Confidencialidad	Ausencia de pruebas a requerimientos no funcionales. Vulnerabilidades de seguridad.

2.2 Valoración de las amenazas

La influencia de las amenazas sobre el valor de los activos, se realiza en dos aspectos: Degradación o impacto, y probabilidad.

- Probabilidad: Oportunidad de que algo suceda (ISO 31000). Para el presente caso de estudio, la probabilidad de que una vulnerabilidad potencial pueda ser explotada por una fuente de amenaza se modela de manera cualitativa empleando la siguiente escala nominal:

Descripción	Valoración	Definición
Bajo	1	La fuente de amenaza carece de motivación. Los controles están listos para prevenir o para impedir significativamente que la vulnerabilidad suceda.
Medio	2	La fuente de amenaza es motivada y capaz. Los controles pueden impedir el éxito de que la probabilidad suceda.
Alto	3	La fuente de amenaza es altamente motivada y suficientemente capaz. Los controles para prevenir que la vulnerabilidad suceda son insuficientes.

- Degradación o impacto: Nivel de afectación que puede ocasionar la materialización de una amenaza. Para el caso de New Soft, se contempló un impacto operacional relacionado con las consecuencias que pueden provocar las amenazas identificadas en el proceso de desarrollo de software en la compañía, y para ello se definieron los siguientes niveles:

Descripción	Impacto Operacional
Leve 10	Afectación a las herramientas empleadas en los desarrollos, provocando indisponibilidad al proceso.
Moderado 20	Indisponibilidad en la ejecución de las fases del SDLC como consecuencia de errores humanos y/o procedimiento inadecuados. Causando pérdida de integridad y confidencialidad.
Mayor 30	Indisponibilidad a causa de factores externos para la ejecución del proceso. Detrimento de la imagen institucional en el mercado. Pérdida de integridad y confidencialidad a la información.

Para facilitar la evaluación de los riesgos, se presenta una matriz que contempla la magnitud del impacto y la probabilidad y que genera los niveles de gestión del riesgo. Esto permite comparar los resultados de la calificación del riesgo, con los criterios definidos para establecer el grado de exposición de la entidad; de esta forma es posible distinguir entre los riesgos aceptables, tolerables, moderados, importantes o inaceptables y fijar las prioridades de las acciones requeridas para su tratamiento.

Probabilidad	**Bajo**	**1**	10	11%	20	22%	30	33%
	Medio	**2**	20	22%	40	44%	60	67%
	Alto	**3**	30	33%	60	67%	90	100%
			10		**20**		**30**	
			Leve		**Moderado**		**Mayor**	
			Impacto					
B: Zona de riesgo baja: Aceptar el riesgo. Entre 0% y 25%								
M: Zona de riesgo moderada: Reducir el riesgo. Entre 26% y 50%								
A: Zona de riesgo Alta: Compartir o transferir el riesgo, reducir. Entre 51% y 70%								
E: Zona de riesgo Mayor: Evitar el riesgo, compartir o transferir, reducir. Mayor a 71%								

Código	Riesgo	Probabilidad	Impacto	Calificación
AM01	Requisitos no definidos correctamente	2	30	A 67%
AM02	La planificación no incluye tareas necesarias	2	20	M 44%
AM03	Diseños demasiado sencillos que no cubren requerimientos no funcionales	3	30	E 100%
AM04	Empleo de bibliotecas de código externas para implementar funcionalidades deseadas	3	20	A 67%
AM05	Omisión de patrones y estándares de diseño y desarrollo	2	2	M 44%
AM06	Entorno de software desconocido	1	3	M 33%
AM07	Herramientas de desarrollo no funcionan correctamente	1	2	B 22%
AM08	El cliente no participa de forma activa en la revisión de los sprint	2	30	A 67%
AM09	Falta de especialización en el personal	3	30	E 100%
AM10	Casos de prueba débiles.	2	30	A 67%

3. Caracterización de las salvaguardas

Dado que es la primera vez que en New Soft se aplica un análisis de riesgos al proceso de desarrollo de software, el establecimiento de controles se plantea como un proyecto a mediano plazo dentro de la compañía. Esto no significa que el proceso se encuentre desprotegido, puesto que se llevan a cabo actividades que de manera indirecta funcionan como salvaguardas para mitigar el estado de los riesgos valorados en el punto anterior, mencionadas actividades abarcan jornadas de capacitación, políticas de mantenimiento de

equipos, políticas de contratación de proveedores que incluyen control sobre configuraciones de servidores, política de restricción de BYOD (Bring Your Own Device), entre otros. Asimismo, se pretende que con la guía de buenas prácticas resultado de este trabajo también se aporte a la mitigación de las amenazas.

Por otra parte, la estimación del estado del riesgo (residual) queda planteado para ser llevado a cabo a futuro, una vez se aplique en un proyecto la guía de buenas prácticas que es el objetivo de este caso de estudio y los demás controles adicionales que se determinen en conjunto con los directivos de la organización.

ANEXO C. Guía de buenas prácticas aplicable a la metodología de desarrollo ágil SCRUM para fortalecer la seguridad de la información.

1. Responsable de seguridad

Este nuevo rol debe tener la capacidad y la experiencia suficiente para identificar los riesgos asociados al desarrollo y transmitirlos al resto de participantes del equipo para conseguir la implicación necesaria de las partes para que el producto final sea un software de calidad, robusto y principalmente seguro.

Dentro de sus funciones se encuentra definir los requisitos mínimos de seguridad al inicio del proyecto (pila del producto), recabarlas en cada incremento (pila del sprint) y supervisar su implementación en las reuniones diarias. También debe acompañar al Scrum Master durante todo el proyecto para tener una visión global del estado y los requisitos en todo momento. Por otra parte, este rol o uno adicional (tester de seguridad) deberá llevar a cabo pruebas estáticas y dinámicas sobre el código fuente a fin de detectar vulnerabilidades y emitir recomendaciones.

2. Formación en seguridad para los equipos de desarrollo.

El equipo de desarrollo (analistas, diseñadores, programadores) y el Scrum Master deben recibir una capacitación inicial en temas de seguridad del software y reforzarla de forma periódica, incluyendo temas como patrones de diseño, codificación segura, casos de abuso, entre otros. Asimismo, el responsable de seguridad deberá llevar a cabo actividades que permitan demostrar el impacto de los riesgos existentes y generar conciencia sobre la seguridad de los productos en todos los actores participantes del proyecto.

3. Diseño de casos de abuso

Los casos de uso permiten diseñar la aplicación para lograr los objetivos que pretende el software. Por el contrario los casos de abuso, hace referencia a las acciones que cualquier usuario legal o atacante pueden llegar a realizar en el producto, afectando los procesos y poniendo en riesgo la disponibilidad, integridad y confidencialidad de la información. Es

tarea de los diseñadores analizar y definir los posibles casos de abuso asociados a cada caso de uso definido y adicionalmente de forma transversal a todos ellos, y el responsable de seguridad debe supervisar la efectividad de mencionada definición.

4. Auditoria de seguridad

Se recomienda realizar una auditoría o inspección técnica de seguridad al final del proyecto como punto de control que facilite la certificación del proyecto en materia de seguridad. Si durante las fases iniciales del proyecto y/o los ciclos de desarrollo (sprint), se filtró algún riesgo a la supervisión del responsable de seguridad del proyecto o alguno de los procesos internos de revisión no fue eficiente, un equipo externo revisará la aplicación por completo, identificando vulnerabilidades y emitiendo recomendaciones para mitigarlas. Con el equipo independiente de seguridad (que puede ser subcontratado) que realiza esta inspección se garantiza legitimidad ya que los intereses no son los mismos del equipo de desarrollo.

5. Verificación de seguridad

Se sugieren los siguientes puntos de verificación PV para facilitar la implementación de la capa de seguridad que se propone sea aplicable a la metodología de desarrollo ágil SCRUM:

Punto de verificación	Objetivo	Elemento	Punto de Control
PV 1. Actores	Garantizar que los actores del proyecto conozcan sus responsabilidades y estén capacitados para desempeñar su rol.	PV 1.1. Capacitación y formación	Verificar que los perfiles profesionales de los miembros del equipo adecuen sus capacidades y experiencia para cumplir con los objetivos del proyecto en materia de seguridad, proveyendo la

			información necesaria para tal fin.
		P.V 1.2 Concienciación	El responsable de seguridad debe desarrollar un plan de generación de conciencia enfocado al cliente y el equipo de desarrollo mediante el cual se transmitan los riesgos asociados al software y el impacto de los mismos.
PV 2. Diseño	Definir aspectos de seguridad dentro del diseño que permitan integrar la totalidad de controles de seguridad en el ciclo de vida de desarrollo.	PV 2.1. Validación de historias	Las historias de usuario deben contener criterios de validación de seguridad que permitan medir la calidad de su aplicación una vez finalizado cada sprint.
		PV 2.2. Casos de abuso	El equipo de desarrollo junto el responsable de seguridad deben identificar los casos de abuso asociados a las historias de usuario para la posterior implementación de las recomendaciones de seguridad en el software y mitigar los riesgos.
		PV 2.3. Análisis	El responsable de

		de riesgos	seguridad debe realizar un análisis de riesgos que permita identificar los puntos débiles de la aplicación y emitir al equipo de desarrollo las recomendaciones de seguridad pertinentes.
PV 3. Implementación	Asegurar que durante el desarrollo de cada sprint se programe de forma segura	PV 3.1 Buenas prácticas de codificación	Los equipos de desarrollo deben programar empleando buenas prácticas de seguridad y patrones para los diferentes lenguajes de programación utilizados. Así cómo evitar la reutilización de código externo a la compañía.
		PV 3.2 Pruebas	Antes de finalizar cada sprint se debe validar que el código fuente es seguro contrastando que el software cumple con los criterios de validación definidos en las historias de usuarios.
PV 4. Auditoria	Realizar un test de seguridad final que garantice que el producto	PV 3.3 Auditoría de seguridad	Un equipo externo de seguridad debe llevar a cabo una prueba que

	es seguro.		permita identificar posibles vulnerabilidades no contempladas por el equipo de desarrollo.
		PV 3.4 Mitigación de vulnerabilidades	Las vulnerabilidades identificadas por el equipo externo deben ser corregidas.