Universidad de Alcalá

Escuela Politécnica Superior

Master Universitario en Dirección de Proyectos de Informáticos

Trabajo Fin de Master

"SEGURIDAD EN METODOLOGÍAS DE DESARROLLO ÁGILES"

Autor: D. Mercedes Iruela Martín

Julio 2016

Universidad de Alcalá

Escuela Politécnica Superior

Master Universitario en Dirección de Proyectos de Informáticos

Trabajo Fin de Master

"SEGURIDAD EN METODOLOGÍAS DE DESARROLLO ÁGILES"

Autor : **D. Mercedes Iruela Martín**

Director Master : **Dr. D. Roberto Barchino Plata**

Tribunal evaluador :

Presidente del Tribunal :

Vocal 1º:

Vocal 2º :

Calificación : _____

Alcalá de Henares a, 15 de Julio del 2016

Seguridad en metodologías de desarrollo ágiles

Proyecto Final de Máster: Dirección de Proyectos Informáticos

Mercedes Iruela Martín

Universidad de Alcalá | 2021

Resumen del proyecto

El proyecto "Seguridad en Metodologías de Desarrollo Ágiles" pretende aunar los conocimientos adquiridos durante el "Máster de Dirección de Proyectos Informáticos" relacionados con diferentes ramas de la Informática, desde gestión de proyectos, normativas y estándares y seguridad a motivación y gestión de equipos por objetivos.

Este proyecto se centrará en el auge de las Metodologías Ágiles que estamos viviendo hoy en día y lo integrará con la seguridad del *software* de forma que, el desarrollo de forma rápida y flexible no se refleje en un software poco seguro y que pueda poner en riesgo nuestras organizaciones.

Para ello, se definirá qué es la Filosofía Ágil, así como el Manifiesto Ágil y nos centraremos en concreto en la metodología de desarrollo *Scrum*. Además, se analizarán diferentes iniciativas, normas, metodologías y herramientas que existen hoy en día para apoyar el desarrollo de *software* seguro y veremos cómo pueden integrarse en *Scrum*.

Analizaremos buenas prácticas relacionadas con la seguridad a nivel de organización y buenas prácticas dentro del desarrollo de *software* para construir *software* que funcione, seguro y rápidamente.

Y finalmente, se hará una propuesta de una posible adaptación de *framework* de desarrollo ágil *Scrum* orientada a asegurar la seguridad del *software*.

Abstract

This project, “Security in Agile Development Methodologies”, aims to put together the knowledge gained in the “Master Dirección de Proyectos Informáticos”. This comprises different areas of the Development of Software from Project Management, Regulations, Standards, and Security to Employees Motivation and Team Management by Objectives.

This project is mainly focused on the increasing trend in the use of Agile Methodologies that we are living with nowadays. It examines how security can be integrated into Agile, rejecting the idea that fast and flexible software means software that lacks security and risks to our organizations.

To achieve my objective, the Agile Philosophy and the Agile Manifesto will be explained, paying special attention to the Scrum Methodology. In addition, Regulations, Methodologies, and some Tools to support Secure Software Development will be analyzed to figure out how to integrate them into the Software Development Life Cycle.

Best practice related to Software Security will be described considering the most effective application on the organizational side and in relation to Software Development.

To conclude, a personal proposal will describe how to integrate Security Best Practice outlined in the project, within the Scrum Methodology.

Introducción

Presentación del Proyecto

Las metodologías ágiles han supuesto un cambio importante en la forma de desarrollar *software*. Ahora se opta por un modo más rápido y adaptable, en lugar de realizar varias etapas largas en las que todos los requisitos y funcionalidades debían estar definidos desde el comienzo del Ciclo de Vida de Desarrollo del *software* y un cambio en ellos podía ocasionar incremento en los tiempos de desarrollo y en los costes.

El desarrollo rápido y adaptable puede tener efectos negativos. Críticos con la metodología consideran que las Metodologías Ágiles apuestan más por la funcionalidad que por la calidad y la seguridad del *software*, lo que puede ocasionar problemas por no haber realizado meditados diseños teniendo en cuenta todos los requisitos. Por otro lado, esta metodología también tiene muchos fanáticos, ya que permite rápidamente tener *software* funcionando y adaptarlo de forma inmediata a las necesidades que un mercado cambiante va fijando. Esperar meses o años en tener un sistema funcionando puede ocasionar frustración en el cliente y que después de todo el tiempo de desarrollo ese *software* no se adapte a las necesidades actuales de la organización.

Sin embargo, como hemos comentado, la velocidad a la hora de aportar valor no debe dejar de lado la calidad, y más en concreto, la seguridad del *software*.

El siguiente proyecto tiene como objetivo el indagar en la Filosofía Ágil y estudiar diferentes ideas, métodos y normas que se centran en seguridad y ver la forma de integrarlos en un entorno ágil.

Para ello, se hará una descripción teórica de los principales temas a tratar en este proyecto. Se empezará por la descripción de la Filosofía Ágil, centrándonos en la metodología de trabajo *Scrum* donde describiremos los roles que considera parte del proceso, eventos y artefactos con los que trabaja.

Posteriormente, nos centraremos en la seguridad del *software*, donde se describirán diferentes metodologías, normas, estándares centrados en la seguridad en el desarrollo de *software*, y veremos si pueden integrarse en un entorno ágil o no. De este análisis obtendremos una serie

de buenas prácticas relacionadas con el desarrollo de *software* seguro en las diferentes fases del ciclo de vida.

Además, aplicaremos todo lo estudiado a una metodología *Scrum*. Por un lado, veremos diferentes temas relacionados con *Scrum* y cómo incluir seguridad en este marco de trabajo. Por otro lado, se tratarán los diferentes enfoques e ideas que actualmente existen sobre estos temas.

Finalmente, en base a todo lo visto hasta el momento, se procederá a hacer una propuesta de una posible implementación de *Scrum* Seguro.

Objetivos

El proyecto "Seguridad en Metodologías de Desarrollo Ágiles" tiene como objetivo principal el aunar los conocimientos adquiridos durante el "Máster de Dirección de Proyectos Informáticos" pero orientado a la metodología de trabajo Ágil, ampliamente utilizada actualmente, y centrándonos en concreto en la Seguridad de *Software*, un tema que se puede considerar que se desatiende cuando se construye *software* de forma rápida y flexible.

Por otro lado, también se persiguen algunos objetivos secundarios como son:

- Conocer más en detalle la Filosofía Ágil, sus valores y principios.
- Detallar la metodología de trabajo Scrum
- Estudiar diferentes normas, estándares y metodologías relacionados con la seguridad y cómo algunas se pueden adaptar al desarrollo de *software* ágil.
- Definir buenas prácticas de seguridad en el desarrollo de *software*.
- Estudiar cómo integrar seguridad en desarrollos ágiles
- Describir una propuesta de metodología *Scrum* que integre buenas prácticas de seguridad en sus procesos.

Marco teórico

La Filosofía Ágil está siendo ampliamente utilizada hoy en día en la construcción de *software*, esto es por la rapidez y flexibilidad que aporta a las empresas, aportando valor desde las primeras iteraciones y permitiendo la colaboración con el cliente en todo el ciclo de vida de desarrollo de *software*.

El desarrollo rápido y flexible, que apuesta por la funcionalidad sobre todo, puede llevar a descuidar la seguridad del *software* desarrollado, esto podría ser un problema ya que supondría un peligro grande para la organización, su imagen y la confiabilidad de los clientes.

A continuación, los siguientes puntos se centrarán en la definición de la Filosofía Ágil, centrándonos en la Metodología *Scrum* en concreto. Posteriormente, estudiaremos diferentes, normas, estándares y metodologías relacionadas con seguridad en el *software* y de ahí se deducirán un conjunto de buenas prácticas a la hora de desarrollar *software* seguro.

Esta base teórica nos permitirá centrarnos en el desarrollo de una propuesta de metodología *Scrum* con consideraciones de seguridad en todas sus fases.

¿Qué es Ágil?

Ágil es una metodología de desarrollo software que permiten una mayor flexibilidad y adaptabilidad a la hora de desarrollar *software*, una mayor implicación del usuario/cliente en el ciclo de desarrollo, así como, una entrega temprana y continua de un producto con valor.

Si hablamos de Ágil a nivel interno de la organización del proyecto, destaca porque permite revisar el proyecto continuamente, ya que se caracteriza por la realización de ciclos cortos, que permiten una constante retroalimentación. Esto nos ayudará a detectar y corregir situaciones, que pudieran llevar al fracaso del proyecto, en etapas tempranas, evitando los sobrecostes asociados que esto pudiera ocasionar.

La idea de estas metodologías rápidas de desarrollo *software* comenzó a desarrollarse en los años 90, ante las carencias observadas en las metodologías de *software* más pesadas existentes: Cascada, espiral o incremental. Metodologías como la metodología en Cascada tenía puntos débiles como la complejidad, la escasa comunicación entre los implicados, así como la poco frecuente validación del código. Cada etapa de la metodología era

independiente y hasta que una no acababa no podía empezar la siguiente, haciendo que los proyectos se extendieran en el tiempo sin que el cliente pudiera ver los avances en él y si realmente el resultado se adaptaba a sus necesidades.

Dada su flexibilidad y dinamismo, las metodologías Ágiles se han convertido en una de las metodologías de desarrollo *software* más utilizada actualmente, principalmente por el hecho de que, al vivir en un mundo en constante movimiento y donde el nivel de incertidumbre es muy alto, Ágil permite una rápida adaptación a las necesidades que van surgiendo en el mercado, pero sin reducir la calidad del producto entregado.

Como Scott Amber+Associates remarcaba ya en sus estudios de 2013, la metodología Ágil es una de las metodologías de desarrollo de software con mayores ratios de éxito. Proporciona mejores resultados en retorno de la inversión, calidad, satisfacción de los grupos de interés, así como en la moral del equipo y en los tiempos de entrega (Amber & Associates, 2013).

Por otro lado, si tenemos en cuenta el estudio de Standish Group Chaos de 2019, estadísticamente, los proyectos ágiles tenían del doble de probabilidades de éxito y un tercio menos de probabilidades de fracaso si los comparamos con los proyectos que seguían una metodología en cascada (Mersino, 2018).

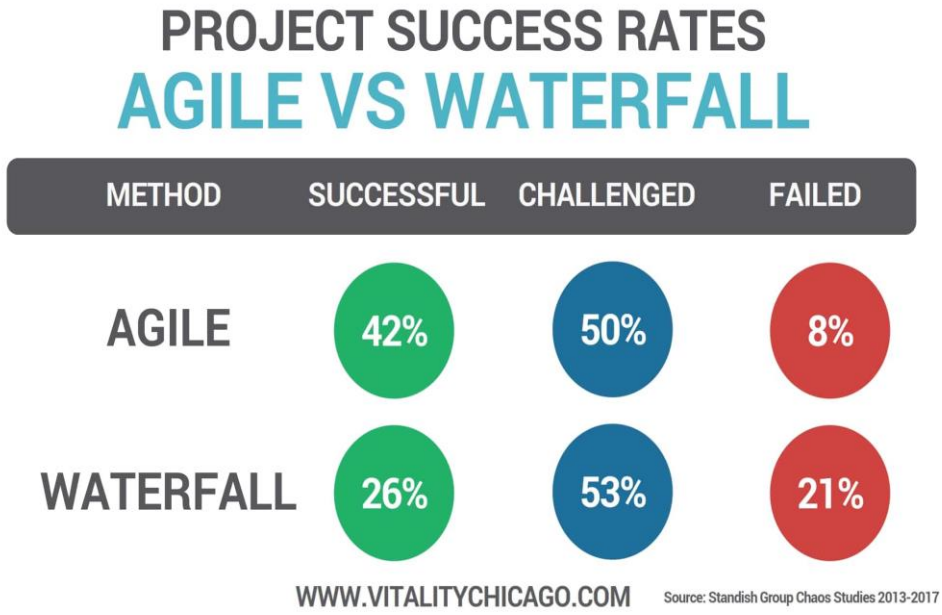

Figure 1Project Success Rates from vitalitychicago.com

Como comentaba Ana Djurovis (Djurovic, 2020) sobre las estadísticas del año 2020, destaca que:

- El 71% de las compañías están adoptando una filosofía Agil.
- La implementación de metodologías ágiles ha ayudado en el 98% de las compañías donde se ha llevado a cabo.
- El 60% de las compañías que adoptan Ágil experimentan un crecimiento en los beneficios.
- La ratio de fracaso en entornos ágiles es del 8%.

Esto deja de manifiesto que el futuro y el presente del desarrollo *software* queda marcado por el uso de metodologías Ágiles.

A continuación, se definirá más en detalle qué son las metodologías Ágiles, en qué principios y valores se basan, y veremos cómo adaptar esta metodología al desarrollo de *software* seguro.

Manifiesto Ágil

El Manifiesto Ágil (Beck, 2001) es un documento que fue desarrollado por expertos en el desarrollo de *software* y que supuso un cambio radical en la forma de construir aplicaciones. Este documento engloba los valores fundamentales sobre los que se cimientan los métodos ágiles. De estos cuatro valores se derivan los 12 principios que dan forma a esta filosofía de trabajo.

En febrero de 2001, tuvo lugar una reunión de un grupo de expertos para debatir sobre las metodologías, principios y valores que debían regir el desarrollo de *software* de buena calidad, que permitiera la entrega de producto más rápidamente y que a su vez fuera flexible a cambios. De esta reunión surgió el término de Agilidad y "*The Agile Alliance*", una organización sin ánimo de lucro dedicada a promover los conceptos relacionados con el desarrollo ágil de *software* y acompañar a las organizaciones para que adopten dichos conceptos. (Uribe, 2007).

Esta reunión sirvió para fijar las bases de la Metodología Ágil, así como para acoger bajo el nombre de Agilidad las metodologías o iniciativas que ya existían bajo esta misma filosofía

de desarrollo software como podrían ser *Lean*, en procesos de manufactura, o *Extreme programming*.

Así pues, de este manifiesto se deduce que la Agilidad no es una metodología como tal, sino una filosofía de trabajo, que puede verse plasmada en diferentes formas de trabajo en las que, como punto común, se respetan una serie de valores y principios que a continuación vamos a describir.

Valores ágiles

Los valores ágiles definen los pilares a la hora de hacer un desarrollo software, son 4 y definen, más que una forma de hacer software, las prioridades para tener en cuenta:

1. *Individuos e interacciones sobre procesos y herramientas:*

Las metodologías ágiles definen que lo que realmente marca la diferencia a la hora de hacer software de calidad es el factor humano. Las personas que trabajan en equipos ágiles son la clave principal del éxito de estos proyectos y no los procedimientos o las herramientas usadas. Quienes realmente hacen que se incremente la productividad de un equipo son las personas implicadas, la motivación, la colaboración y el compromiso para conseguir un objetivo.

Este valor se encuadra muy bien dentro de las teorías motivacionales en las que se aboga por la afiliación, la confianza y el reconocimiento de los individuos para sacar lo mejor de ellos como se describe en teorías como la de Maslow, la Teoría de la Motivación-Higiene de Herzberg o la teoría Y entre otras.

2. *Software funcionando sobre documentación extensiva*

En las metodologías tradicionales, se daba mucha importancia a dejar todo perfectamente definido y plasmado en documentos que cualquier persona implicada en el proyecto pudiera consultar: planes de trabajo, documentos de análisis y diseño, requisitos detallados del sistema. Estos documentos, en la mayoría de los casos, eran creados al comienzo del proyecto durante las primeras semanas o meses. Lo que la filosofía Ágil propone es empezar pronto a implementar funcionalidades, de esta forma, el producto puede empezar a usarse antes y se puede obtener un *feedback* temprano que permite ir adaptando el producto a las necesidades

reales de los usuarios. Esto no quiere decir que no se documente, quiere decir que se documente lo estrictamente necesario, dando prioridad al contenido sobre la forma

3. *Colaboración con el cliente sobre negociación contractual*

El cliente debe estar involucrado en el desarrollo *software* desde el principio las primeras fases, ayudando a definir los requisitos y participando en las distintas iteraciones, siendo parte del proceso, dando su opinión en el avance de éste y ayudando a definir prioridades. Además, al poder utilizar el software antes, el cliente puede evaluar si realmente se adapta a lo que necesita o no.

4. *Respuesta ante el cambio sobre seguir un plan*

Vivimos en un mundo en cambio constante y rodeado de incertidumbre. Poder ir adaptando el *software* desarrollado a las nuevas necesidades permite adaptarse rápidamente la empresa a los cambios del mercado facilitando una mejor capacidad de reacción y adaptación sobre el resto de los competidores.

Estos valores no pretenden que no se le dé importancia a los procesos y herramientas, a la documentación, a las negociaciones contractuales o a la planificación, todos estos elementos son importantes y necesarios, lo que remarca esta filosofía es que, aunque esos elementos sean importantes, es necesario priorizar los elementos de la izquierda de la frase, sobre los de la derecha.

Principios ágiles

Cómo parte del Manifiesto Ágil (Beck, 2001), y teniendo en cuenta los valores que lo sustentan, se definen una serie de principios que son necesarios seguir para que esta filosofía de desarrollo pueda funcionar:

1. *Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.*
2. *Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.*

3. *Entregamos software funcional frecuentemente, entre un par de semanas y un par de meses, preferentemente en el periodo de tiempo más corto posible.*
4. *Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.*
5. *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.*
6. *El método más eficiente y efectivo de comunicar información al equipo de desarrollo, y entre los miembros del equipo, es la conversación cara a cara.*
7. *El software funcionando es la medida principal de progreso.*
8. *Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.*
9. *La atención continua a la excelencia técnica y al buen diseño permite la mejora de la agilidad.*
10. *La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.*
11. *Las mejores arquitecturas, requisitos y diseños emergen de equipos auto organizados.*
12. *A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.*

En base a estos valores y principios ágiles se han creado diferentes formas de trabajo como *Scrum*, XP, *Crystal*, DSDM, *Extreme Programming*, *Adaptive Sotfware Development*, *Feature-Driven Development*, *Kanban*, *Lean*, etc.

Cabe destacar que el uso de la filosofía Ágil está muy focalizado en el área de desarrollo de *software*, pero realmente, estas directrices podrían ser y son, utilizadas en otras muchas áreas de producción, trabajo o servicios.

Metodología vs filosofía

Una metodología se define según la RAE como "Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal", teniendo en cuenta que Ágil se sustenta más sobre unos valores y principios que sobre unos métodos concreto, podríamos considerar que se trata más de una filosofía de trabajo en lugar de metodología de trabajo.

Como conclusión, podríamos considerar Ágil como una filosofía de trabajo y sus diferentes variantes como metodologías o frameworks. Con variantes nos referimos tanto a *Scrum*, la más utilizada, como a Lean, *Kanban* o *Extreme programming*. Cada una de ellas tiene ventajas e inconvenientes y la organización o el equipo de desarrollo podrán elegir unas u otras dependiendo de las características del sistema, del proyecto a desarrollar o de lo que más se adapte a la forma del trabajo del equipo.

Las metodologías que siguen la Filosofía Ágil más populares son:

- *Scrum:* este método de trabajo se caracteriza por utilizar *Sprints*, que son periodos fijos de tiempo de un mes o menos. Un *Sprint* es una iteración y cuando un *Sprint* termina, empezará el siguiente y así hasta la finalización del proyecto o el producto. Se explicará con más detalle a continuación. Contempla 3 roles principales: *Product Owner*, *Scrum Máster* y Equipo de desarrollo.
- *Extreme Programming (XP):* es una metodología exclusiva para el desarrollo de *software*. Se basa principalmente en fortalecer el trabajo en equipo y la colaboración.
- *Dynamic Systems Development Method (DSDM)* su premisa básica es entregar la solución adecuada en el momento adecuado (DSMD Consortium, 2015). Se basa en construir *software* en base al principio de *Rapid Application Development*, por lo que se centra en minimizar el tiempo necesario para desarrollar un sistema.
- *Feature Driven Development (FDD)*: es una forma de desarrollo centrada en características o *features*. Se define *feature* como una característica pequeña en forma de función que aporta valor al cliente. Los desarrolladores se agrupan en dos tipos, "dueños de clases" o "programadores jefes".
- *Kanban:* es una metodología de trabajo que se desarrolla entorno a un panel de tareas, este panel permite organizar el trabajo de forma visual y que las tareas pasen de un estado a otro de forma ordenada.

Cabe destacar que en el caso de las Metodologías Ágiles no siempre hay que decantarse por una u otra, sino que se pueden combinar entre sí. El objetivo final pasa por maximizar el valor del resultado del proyecto, evitando las tareas que no aportan valor, generando el menor desperdicio posible y mejorando de forma continua.

Teniendo en cuenta estas bases, nos vamos a centrar en la metodología de trabajo *Scrum* y vamos a definir sus características básicas para después poder centrarnos en el objetivo del proyecto que es la inclusión de seguridad dentro del desarrollo rápido de *software*.

¿Qué es Scrum?

Según la Guía de *Scrum* de 2020 (Schwaber & Sutherland, 2020), *Scrum* es un marco ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptables para problemas complejos. Para ello, *Scrum* utiliza periodos fijos de tiempo llamados *Sprints*. De forma que, cuando un Sprint termina, comenzará el siguiente y así continuamente hasta que la aplicación o el producto esté finalizado (Schwaber & Sutherland, 2013).

A continuación, se definen las principales características de una metodología *Scrum*: eventos, artefactos y roles implicados.

Roles

La Metodología *Scrum* cuenta principalmente con 3 roles bien diferenciados: *Product Owner*, *Scrum Master* y Equipo de desarrollo. Posteriormente, dependiendo del proyecto, pueden incluirse otros roles de apoyo que ayuden en aspectos concretos del proyecto: Especialista en Seguridad, Especialista en Derecho, etc.

Scrum Master:

El *Scrum Marter* es el encargado de hacer el seguimiento del proyecto de forma que se cumplan principios de *Scrum*. Vela porque se lleven a cabo las distintas reuniones necesarias para el correcto desarrollo de la metodología. También es su función facilitar al equipo de desarrollo todo lo necesario para que puedan llevar a cabo sus tareas sin interrupciones e incidencias.

El rol de *Scrum Master* se puede definir también como el de facilitador, ya que sus tareas son de utilidad para *Product Owner*, para Equipo de Trabajo y para la propia organización

El *Scrum Máster* sirve al Equipo *Scrum* de varias maneras, incluyendo:

- Capacitar a los miembros del equipo en autogestión y multifuncionalidad;

- Ayudar al equipo de *Scrum* a centrarse en la creación de incrementos de alto valor que cumplan con la definición de hecho;
- Promover la eliminación de los impedimentos para el progreso del equipo *Scrum*.
- Asegurar que todos los eventos de *Scrum* se lleven a cabo, sean positivos, productivos y que se respete el tiempo establecido para cada uno de ellos.

El *Scrum Master* ayuda al Propietario del Producto (*Product Owner*) de varias maneras, incluyendo:

- Ayudar a encontrar técnicas para una definición eficaz de los objetivos del producto y la gestión de los retrasos en el producto;
- Ayudar a que el equipo de *Scrum* comprenda la necesidad del trabajo, ayudándole a definir productos claros y concisos;
- Ayudar a establecer la planificación de los productos;
- Facilitar la colaboración entre las partes interesadas según sea necesario.

El *Scrum Master* sirve a la organización de varias maneras, incluyendo:

- Liderar, capacitar y actuar como mentor en la adopción de *Scrum*;
- Planificar y asesorar sobre la implementación de *Scrum* dentro de la organización;
- Eliminar las barreras entre las partes interesadas y los equipos de *Scrum*.

Product Owner:

El *Product Owner* o Dueño del. Producto (PO) es la persona encargada de definir las características del producto, cómo será el producto y describir las épicas e historias de usuario necesarias para hacer comprender al equipo de desarrollo lo que se espera. El Propietario del Producto es el encargado de ordenar y priorizar el trabajo a realizar reflejándolo en el *Backlog* del Producto, de forma que se maximice el valor del producto resultante del trabajo del equipo de *Scrum*. Sus tareas principales son:

- Desarrollar y comunicar el Objetivo del producto,
- Creación y comunicación clara de los elementos de trabajo pendiente del producto.
- Priorización del trabajo pendiente en el producto

- Asegurarse de que el trabajo pendiente del producto sea transparente, visible y comprendido.

Equipo de desarrollo:

El Equipo de desarrollo se encarga de convertir el *Backlog* del Producto priorizado en incrementos desarrollados durante los *Sprints*. El equipo de trabajo *Scrum* debe tener las siguientes características:

- *Multifuncionales:* cada miembro del equipo tiene las habilidades necesarias para generar valor.
- *Auto gestionado:* el propio equipo es el que decide quién desarrollará las diferentes tareas, cómo y cuándo se llevarán a cabo.
- *De 3 a 9 miembros:* en varios sitios se habla de estos números, pero básicamente la idea es que el equipo sea lo suficientemente pequeño para poder ser ágiles y flexibles, y lo suficientemente grande para poder completar una cantidad de trabajo significativa en un *Sprint*. Cabe destacar que, por lo general, los equipos pequeños se comunican mejor y son más productivos.
- *Formado:* para poder desarrollar software seguro y de calidad desde el principio el equipo tiene que estar bien formado y actualizado a las últimas tendencias y herramientas.

Responsabilidades del equipo de trabajo:

- Crear un plan para el *Sprint*, lo que se conoce como *Backlog* del *Sprint*.
- Inculcar la calidad adhiriéndose a una Definición de Hecho (DoD)
- Adaptar su plan cada día hacia el objetivo del *Sprint*
- Responsabilizarse mutuamente como profesionales.

Eventos

Scrum define una serie de eventos o reuniones que son necesarias para el correcto desarrollo de la metodología. Como ya hemos comentado, el *Scrum Máster* es la persona encargada de velar por el correcto desarrollo de estos eventos de forma que sean positivos y beneficiosos para el avance del proyecto y la consecución del objetivo del proyecto.

- *Sprint:* La *Scrum.org* (Scrum.org, s.f.) define un *Sprint* como el contenedor de todos eventos de *Scrum*. Los *Sprints* son periodos fijos de tiempo, cuya longitud suele ser fija y como máximo de un mes. El resto de los eventos definidos en esta sección se llevan a cabo dentro del *Sprint*. Se debe definir un objetivo del *Sprint*, con el que el equipo se comprometa a su realización. Esto hace que en muchas definiciones se hable de un *Sprint* como de un "miniproyecto".
- *Planificación del Sprint:* La planificación del *Sprint* marca el comienzo del *Sprint*. En ella se establecerá el trabajo que se realizará durante el mismo de forma consensuada por el Equipo de Desarrollo. En esta reunión se definirá el objetivo del *Sprint* y las tareas que se desarrollarán para la consecución de este objetivo. Estas tareas se dividirán en tareas más pequeñas que puedan ser completadas en un día o menos. Como forma de predecir el trabajo que se puede realizar en un *Sprint*, se suelen utilizar Puntos de Historia, que son asignados por el Equipo de Desarrollo en función de la complejidad de las tareas de usuario.
- *Reuniones diarias:* sirven para hacer seguimiento del estado del bbjetivo del *Sprint* y permite al equipo exponer si se está encontrando alguna dificultad. Estás reuniones no deben durar más de 15 minutos, en algunas ocasiones se hacen de pie (stand-ups) para que sean más rápidas. El equipo debe informar sobre qué esta haciendo, qué va a hacer después y dificultades que se pueda estar encontrando en sus tareas.
- *Demo o revisión del Sprint:* Esta reunión se lleva a cabo al finalizar el *Sprint* y sirve para mostrar el resultado del *Sprint* a las partes interesadas y analizar como va el avance hacia el objetivo del producto. Se trata de una sesión de trabajo dónde se muestra la situación actual y los avances llevados a cabo y de definen futuros pasos para la consecución del Objetivo del Proyecto.
- *Retrospectiva del Sprint:* La idea principal de este evento es aprender de la experiencia y planificar formas de mejorar la calidad y eficacia del producto desarrollado y del equipo de trabajo. En esta reunión se analiza, en base al último *Sprint*, como han sido las interacciones, procesos y herramientas, se evalúa la Definición de Hecho y si es necesario hacer algún cambio. Si ha habido algún problema durante el *Sprint*, en esta reunión debe tratarse y definir futuros pasos para evitar que la situación se repita. La retrospectiva marca el fin del *Sprint*,

donde se aprende de la experiencia y se evalúan nuevas acciones para seguir mejorando dentro de un marco de mejora continua.

- *Reunión de refinamiento del Backlog*: esta reunión no se define dentro del marco *Scrum*, pero es utilizada por muchas compañías para poder ir trabajando sobre el *Backlog* ayudando al *Product Owner* a poder detallar más las historias de usuario, mejorar la estimación, asignando puntos de historia a los elementos del *Backlog* y desarrollar un trabajo previo de análisis para mejorar la pila de producto y así tener más claro el objetivo a definir para siguientes *Sprints*. El refinamiento del *Backlog* de producto consiste en descomponer y definir más en detalle los elementos de trabajo pendiente, de forma que se trabaje con tareas más pequeñas y precisas.

Artefactos:

Existen una serie de artefactos o activos necesarios asociados a la metodología de *Scrum*. Estos artefactos son mantenidos o desarrollados por los diferentes roles que participan en él:

- *Backlog de producto o Pila de Producto:* es una lista de tareas pendientes, ordenada por prioridades, que define qué es necesario implementar para mejorar el producto y lograr el Objetivo del Producto. En la reunión de planificación se trabaja con la Pila de Producto, así como en la de refinamiento, mejorando las historias de usuarios, descomponiéndolas y estimando los puntos de historias asociados a cada tarea.
- *Épicas, historias de usuario y tareas:* El *Backlog* está formado por épicas, historias y tareas que son unidades independientes de trabajo que será necesario llevar a cabo para la consecución del Objetivo del Producto.
- *Backlog del Sprint*: El *Backlog* del *Sprint* es definido por el equipo *Scrum* por consenso y ayuda a definir lo que será el objetivo del *Sprint*. El equipo *Scrum* se compromete a la implementación de una serie de historias o tareas y si se encuentra dificultades, se negociará con el *Product Manager* el alcance a conseguir.
- *Entregables:* también llamado incremento, valor que se añade en cada *Sprint* a la consecución del Objetivo del Producto. Una tarea o historia de usuario se considerará parte de un incremento siempre que cumpla con la Definición de Hecho.

- *Definición de hecho (DoD):* La definición de hecho aplica a todas las tareas, historias y entregables. Define una serie de medidas necesarias de calidad requeridas para el producto. La Definición de Hecho permite dar transparencia sobre qué trabajo se completó como parte del incremento. Una tarea, historia o épica no se puede presentar en la revisión del *Sprint* como completada si la Definición de Hecho no se cumple. La Definición de Hecho puede ser a nivel de estándares de la organización o definirse como parte del producto, el Equipo de Trabajo también podría modificarla si encontrara deficiencias o posibles mejoras como parte del proceso de mejora continua.

Desarrollo seguro de software

La seguridad en los sistemas de información consiste en la protección de los sistemas de información de accesos o modificación de datos no autorizados, ya sea relativo a almacenamiento o tránsito o contra la denegación de servicio a usuarios autorizados, incluyendo aquellas medidas necesarias para detectar, documentar y tener en cuenta esas amenazas (Kissel, 2013).

Esta seguridad puede afectar a muchas partes de la infraestructura o incluso a la forma de uso del sistema por parte de los usuarios. Por lo que es importante que, de forma continuada, prestemos atención a todos los puntos de nuestra organización que pudieran poner en peligro la integridad, disponibilidad y confidencialidad de nuestro sistema.

A continuación, se van a describir algunos de los problemas más frecuentas a la hora de desarrollar software seguro, tanto de forma general, como específicamente en entornos *Scrum*. Después se analizarán diferentes iniciativas, normativas, estándares y herramientas que se centran en ayudar en la tarea de integrar la seguridad en el desarrollo de *software*.

Como resultado de este análisis, se ha desarrollado un listado de buenas prácticas a llevar a cabo para asegurar la seguridad en el Ciclo de vida de Desarrollo de *software*. Este listado, está dividido en dos partes: buenas prácticas a nivel organizacional y buenas prácticas a la hora de desarrollar software.

Problemas encontrados en el Ciclo de vida de desarrollo de software seguro

En un estudio sobre Seguridad en los Ciclos de vida de desarrollo software (Assal & Chiasson, 2018), se tuvo en cuenta a 13 participantes de diferentes proyectos y empresas y se les preguntó sobre actividades relacionadas con la seguridad durante el desarrollo de *software*, su actitud y la de su equipo hacia la seguridad del *software*, conocimientos sobre seguridad del *software*, procesos relacionados con seguridad y actividades que llevaban a cabo durante las pruebas. En el estudio se pudieron observar dos tipos claros de actitudes, la actitud de los adoptantes de prácticas de seguridad y los que desatienden a la seguridad en el desarrollo de *software*. A continuación, se muestra la tabla de resumen de los resultados obtenidos en dicho estudio de 2018:

Adoptantes de seguridad	Ambos	Desatentos a la seguridad
	Diseño	
• Es muy importante diseñar la seguridad	• La seguridad no se considera normalmente en la fase de diseño	• La seguridad en las fases de diseño suele ser Adhoc
	Implementación	
• La seguridad es una prioridad en la fase de implementación	• Se espera concienciación sobre seguridad de los desarrolladores durante la implementación.	• Seguridad no es una prioridad durante la implementación • Los desarrolladores dan la seguridad por sentado. • Los desarrollares hacen un mal uso de los frameworks • Lo desarrolladores no tiene conocimientos en seguridad • Los desarrolladores consideran que sus conocimientos en seguridad no son claros • Descubrir una vulnerabilidad puede ayudar a motivar sobre la seguridad.
	Pruebas de los desarrolladores	
• Los desarrolladores prueban la seguridad de forma fortuita • Seguridad es una prioridad durante las pruebas de desarrollo.	• Las pruebas de los desarrolladores no incluyen seguridad	• Las pruebas sobre seguridad de los desarrolladores son orientadas a funcionalidades.
	Análisis del código fuente	
• La seguridad es una prioridad durante el análisis del código fuente		• La seguridad un objetivo secundario durante el análisis del código. • Los desarrolladores rara vez hacen análisis del código, y si se hace, nunca es por seguridad. • No todos los desarrolladores conocen bien las herramientas de análisis de código
	Revisiones de código	
• La revisión de código es un proceso formal que incluye seguridad • Una revisión preliminar del código se debe hacer como punto de control antes de proceder a la revisión formal		• La seguridad no está considerada durante las revisiones de código • La seguridad es considerada de forma mínima durante las revisiones de código.
	Pruebas de post-desarrollo	
• La seguridad es una prioridad en las pruebas de		• La seguridad no se considera en las pruebas de post-desarrollo.

post-desarrollo • Las pruebas de post-desarrollo se utilizan para encontrar vulnerabilidades de seguridad o para una verificación final • Los encargados de pruebas son los que realizan la aprobación final.		• Los planes pruebas de post-desarrollo incluyen una dimensión de seguridad. • Las pruebas de seguridad en el post-desarrollo son orientadas a la funcionalidad. • Las pruebas de seguridad en el post-desarrollo son adhoc. • Las pruebas de seguridad en el post-desarrollo son externalizadas.

De este estudio de sacaron los siguientes factores que propician que no se sigan las buenas prácticas en cuanto a seguridad.

- *División de tareas:* Aplicar el principio de seguridad en todas las fases del ciclo de vida puede ser un problema ya que, normalmente, la seguridad no es la prioridad y los roles implicados en cada fase consideran que las actividades relacionadas con asegurar la seguridad deben ser llevadas a cabo por otros grupos, por ejemplo, los desarrolladores se encargan de desarrollar funcionalidades y esperan que la gente de pruebas verifique la seguridad. Este método de distribución de tareas choca con la buena práctica de "Aplicar defensa en profundidad". Otro factor relacionado con este tema es que la seguridad no suele ser la prioridad, a nivel empresarial se da más importancia a la funcionalidad y a la productividad del equipo. Esto lleva a que algunos equipos se centren en las funcionalidades y dejen de lado la seguridad del *software*. La seguridad debería considerarse una prioridad y transmitirse así desde la dirección de la empresa.
- *Conocimientos en seguridad:* Existen dos factores que pueden influir en la motivación de los desarrolladores en cuanto a la seguridad. Por un lado, si la compañía espera que se tengan conocimientos de seguridad, entonces se tiende a meter tareas de seguridad en el Ciclo de Vida del Desarrollo de *Software*, en cambio, si no se espera que tengan conocimientos en seguridad de forma explicita, se tiende más a desatender la seguridad en las diferentes fases desarrollo. Podría ser interesante tener un perfil más experto en seguridad dentro de los equipos de desarrollo que apoye y forme al resto del equipo o dar formación continua sobre el tema a todos los miembros del equipo.
- *Cultura empresarial:* La compañía debe fomentar la seguridad desde su cultura y políticas, esto hará que los desarrolladores estén más comprometidos con la

seguridad. Integrar y premiar la seguridad como parte de la cultura empresarial puede ayudar a motivar significativamente las buenas practicas en seguridad

- *Disponibilidad de recursos:* Algunas compañías aplican las buenas prácticas de seguridad en función del presupuesto asignado o si tiene empleados que ya tengan el conocimiento requerido. Esto puede ser un problema en empresas pequeñas con pequeños presupuestos o equipos pequeños en los que una persona no puede estar exclusivamente dedicada a la seguridad, primero porque puede llegar a cierto nivel de fatiga por estar únicamente centrado en eso y, por otro, porque toda la seguridad estaría centrada en una única persona, lo que puede llevar a ser un problema en caso de que esa persona abandone la empresa.
- *Presión externa:* Por empresas externas o por el gobierno, por ejemplo, una certificadora externa que esté monitorizando los procesos y productos de la empresa haría que los miembros del equipo sean más conscientes de las tareas de seguridad y presten mayor atención a éstas. Legislaciones como RGPD, obligan a las compañías a prestar especial atención a la privacidad de los datos en los desarrollos.
- *Sufrir un incidente de seguridad:* El experimentar un incidente de seguridad en el proyecto o el conocer que una vulnerabilidad ha sido encontrada en el propio código ayuda a que aumente la concienciación sobre seguridad y la motivación por crear *software* seguro. En gran número de proyectos se considera que el producto no va a intentar ser atacado por ciberdelincuentes o que no van a tener la mala suerte de tener una vulnerabilidad y que esa sea utilizada para provocar cualquier tipo de incidente. Para que no se llegue al caso de tener que sufrir un ataque para realmente empezar a pensar en la seguridad, es de utilidad dar formación a los empleados haciéndoles entender las consecuencias que podrían desencadenar una vulnerabilidad en su proyecto, con ejemplos que puedan entender y acciones que permitan disminuir esos posibles problemas.

En consecuencia, muchos de los problemas detectados estarían relacionados con el presupuesto y cómo ese presupuesto se destina prioritariamente a completar nuevas funcionalidades en lugar de asegurar las existentes, así como con las presiones que hay en desarrollo por sacar el máximo partido al presupuesto disponible. Esto viene a reflejar un problema a nivel más alto, no exclusivamente relacionado con el área de desarrollo. Esto nos hace considerar que existe una necesidad de poder incluir automatizaciones adicionales que

ayuden con las tareas de desarrollo y aseguramiento de la seguridad que permitan reducir la presión sobre el equipo y seguir produciendo de forma ágil y eficiente.

Problemas encontrados en el Ciclo de vida de desarrollo de software seguro usando frameworks Ágiles

Si nos centramos en concreto en *frameworks* de desarrollo Ágiles, se pueden detectar los siguientes problemas a la hora de desarrollar software seguro (Nguyen, 2015):

- Presión debida a las iteraciones cortas, en ocasiones lo cortas que son las iteraciones pueden no facilitar de la tarea de hacer suficientes pruebas (Securosis, 2013)
- Falta de conocimiento sobre seguridad, lo que lleva a pasarla por alto y no incluirla como parte del desarrollo (Securosis, 2013)
- Falta de concienciación sobre la seguridad por parte de los usuarios, en muchas ocasiones debido a la idea de que desarrollar software seguro es costoso y no da beneficios económicos directos a la empresa.
- Incompatibilidad de las actividades de seguridad con los *frameworks* Ágiles. Algunas actividades como formación y concienciación, aplicar seguridad desde el principio del ciclo de vida de desarrollo de software, realizar análisis estático de código o revisar diseños de seguridad, se pueden adaptar a un modelo Agile. Sin embargo, uno de los mayores retos es adaptar a los modelos ágiles el Modelado de Amenazas y la Gestión de Riesgos, esto es debido a su complejidad y a que no es algo que requiera colaboración de los usuarios, por lo que choca con los valores ágiles, sin embargo, es necesario llevar a cabo estas tareas.

Muchos de los problemas detectados de forma genérica en el punto anterior, también se ven reflejados en estos problemas definidos por Tran Nguyen en su estudio, pero cabe destacar el último, que es muy específico de la agilidad y está directamente relacionado con los valores y principios de esta filosofía de trabajo.

Software seguro: iniciativas, metodologías y normas

Esta sección está enfocada en descubrir y analizar iniciativas, metodologías y normas que han surgido en torno a la seguridad en el Ciclo de Vida del Software. Éstas no tienen por qué estar destinadas al desarrollo ágil en concreto, lo que se pretende es hacer una pequeña investigación sobre las diferentes iniciativas que han surgido sobre seguridad, determinar las buenas prácticas en las que coinciden y cómo esas buenas prácticas se pueden aplicar en una metodología de desarrollo Ágil como es *Scrum*.

S-SDLC

Cuando se habla de desarrollo de *software* seguro es necesario hablar sobre S-SDLC (*Secure Software Development Life Cycle*), S-SDLC se trata de un conjunto de principios de diseño y buenas prácticas a la hora de implantar un SDLC (*Software Development Life Cycle*), con el fin de detectar, prevenir y corregir defectos de seguridad en el desarrollo y adquisición de aplicaciones, de forma que se obtenga *software* confiable y resistente a fallos, es decir se asegure la integridad, disponibilidad y confiabilidad del producto. La idea que defiende esta metodología es verificar los requisitos de seguridad a lo largo de las distintas fases de construcción del software: análisis, diseño, desarrollo, pruebas y mantenimiento; sobre todo, durante las dos primeras fases, análisis y diseño, ya que gran parte de las debilidades de los sistemas se generan antes de comenzar las tareas de programación. Las claves del S-SDLC son la atención al detalle, para favorecer la identificación inmediata de las vulnerabilidades; y la mejora continua (Encinar, 2018).

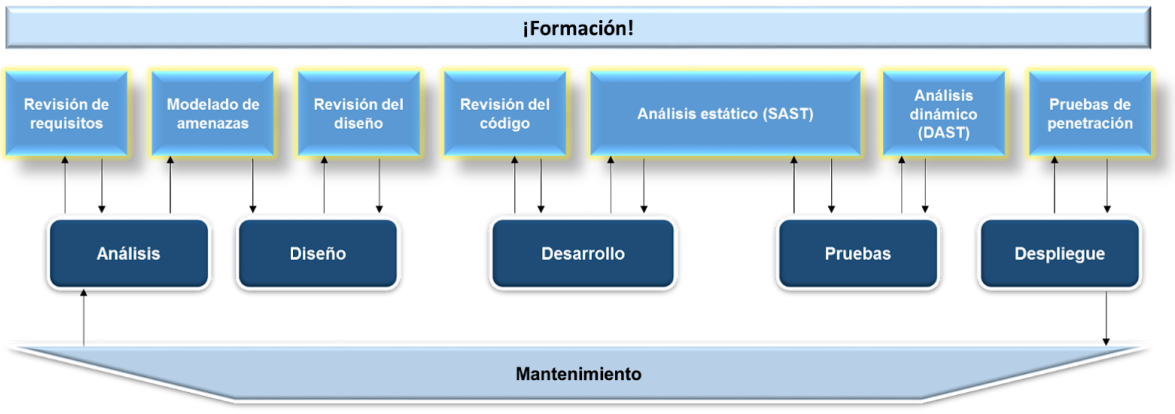

Figure 2. Diagrama S-SDLC

Para poder construir software confiable y seguro es necesario la formación del equipo para ser consciente de los problemas de seguridad relacionados con el desarrollo de *software* y cómo afrontarlos y evitarlos.

S-SDLC puede ser aplicado a metodologías ágiles sin problema.

UMLSe

La metodología de desarrollo UMLSec es una extensión del UML incluyendo Seguridad. Especifica requisitos de seguridad relacionados con integridad y confidencialidad. UMLSec permite expresar información relevante de seguridad dentro de los diagramas de especificación de sistemas.

El modelo define un conjunto de operaciones que puede efectuar un atacante a un estereotipo y trata de modelar la actuación del subsistema en su presencia. La metodología UMLSec pretende reutilizar los diseños ya existentes, aplicando patrones para formar nuevos a partir de una transformación de los existente.

En el "*Modelling security requirements through extending Scrum agile development framework"* (Alotaibi, 2016) se hace una disertación sobre cómo utilizar UMLSec en entornos de desarrollo ágiles, incluyendo un nuevo rol, *Security Owner*, que será encargado de facilitar las herramientas y requerimientos de seguridad que son necesarios considerar dentro del proyecto.

Microsoft Security Development Lifecycle MS SDL

Microsoft Security Development Lifecycle es la primera iniciativa que comenzó a remarcar la necesidad de integrar la seguridad en el SDLC desde las primeras fases.

MS SDL presenta un proceso que consta de 7 etapas, que se puede aplicar de forma general con independencia de la plataforma. Destaca la fase inicial de formación, así como una etapa posterior al desarrollo orientada al soporte de incidentes.

Figure 3. Diagrama MS SDL

A. *Formación:* La formación es una práctica indispensable y prerrequisito para implementar MS SDL.Se estipula que debe haber una frecuencia mínima de formación en seguridad y se sugiere que sea de una vez al año para los roles técnicos

B. *Requerimientos:* se establece que un consultor de seguridad debe apoyar al equipo de desarrollo en la fase de descripción y toma de requisitos. En esta fase se establecerán requisitos propios de seguridad, se crearán umbrales de calidad y límites de errores y se evaluarán los riesgos de seguridad y privacidad.

C. *Diseño:* 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. En esta fase, además de establecer los requisitos de diseño seguro, se analizarán el tipo de ataques que se podrían tener y se definirá el Modelo de Amenazas asociado al proyecto.

D. *Implementación:* Durante la fase de implementación se definen varias consideraciones:

a. Utilizar herramientas aprobadas: Definiendo una lista de herramientas y comprobaciones de seguridad asociados a las mismas.

b. Evitar usar APIs inseguras: existen herramientas y sitios especializados en darnos información sobre APIs o librerías que incluyen vulnerabilidades.

c. Codificación siguiendo los estándares y buenas prácticas de implementación de software.

d. Analizar el código fuente antes de la compilación, se pueden usar herramientas de Análisis de código estático (SAT).

E. Verificación: del rendimiento en tiempo de ejecución del software, teniendo en cuenta daños en la memoria, problemas de privilegios de usuario y otros problemas críticos de

seguridad. Incluiría, análisis dinámico, *Fuzz testing* y revisión de los tipos de ataques detectados en la fase de diseño.

F. Lanzamiento: definir un plan de respuesta a incidentes para ayudar a enfrentar las nuevas amenazas que puedan surgir con el tiempo y pruebas de seguridad finales.

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.

Debido al auge de las metodologías ágiles, posteriormente surgió una inicitativa orientada a adoptar MSSDL siguiendo los principios ágiles vistos anteriormente, no tiene nombre oficial, pero lo denominaremos SDL/A como se hizo en el artículo Agile SDL: *Streamline Security Practises for Agile Development* en la revista MSDN de Microsoft (Sullivan, 2019). Para adaptar SDL a la filosofía de *Scrum*, se definen 3 tipos de requisitos SDL:

- *The Every-Sprint Requirements:* requisitos que deben considerarse y ser completados en cada *Sprint*. Estos requisitos se eligen teniendo en cuenta importancia del requisito o cuántas vulnerabilidades previene un requisito concreto y el otro factor es la facilidad de ser automatizado.
- *On-boarding requirements*: se refiere a requisitos que son necesarios completar una vez al comienzo del proyecto. Pueden ser tareas de apoyo al resto del proyecto, por ejemplo, preparación del entorno de la integración continua o la primera versión del Plan de Gestión de Riesgos. SDL/A da tres meses para completar este tipo de requisitos.
- *The Bucket Requirements*: el resto de los requisitos entran en este grupo y se dividen en Verificación de Seguridad (*fuzz testing),* Revisión de diseño y Planes de respuesta (*Plan de Disaster Recovery*). Por defecto, estos requerimientos deben ser completados una vez al año, aunque dependiendo del requisito se puede establecer un rango de tiempo diferente.

SDL/A también tiene en cuenta la parte de Plan de Gestión de Riesgos y considera fundamental el conocer las vulnerabilidades que se sabe que no están cubiertas (ya sea por esfuerzo o falta de tiempo). La propuesta es, en lugar de desarrollar este documento completo en las primeras fases del ciclo de vida, ir completándolo con la funcionalidad incluida en cada Sprint.

Actualmente, estas referencias se han eliminado del sitio web de Microsoft, pero se ha actualizado el modelo del ciclo de vida integrando procesos de DevOps y otras herramientas para ser utilizadas en su servicio *Cloud*.

BSIMM

BISMM (*Building Security In Maturity Model)* es un estudio de iniciativas de seguridad relacionadas con *software* en el que se recogen las iniciativas de diferentes organizaciones de forma que ayudan a otras organizaciones a construir su plan de SDLC seguro y a comparar sus iniciativas con iniciativas de otras organizaciones. BISMM consiste en 12 prácticas organizadas en 4 dominios diferentes:

- Gobernanza: prácticas que ayudan a organizar, gestionar y medir las iniciativas de software seguro: estrategia y métricas, cumplimiento y política, capacitación.
- Inteligencia*:* conjunto de prácticas basadas en el conocimiento de las organizaciones que ayudan a establecer unas actividades de seguridad en el software: modelos de ataque, características de seguridad y diseño, normas y requisitos.
- Puntos de contacto con el SSDL*:* Describe prácticas asociadas con el análisis y aseguramiento de los artefactos y procesos de cierto software en desarrollo: Análisis de la arquitectura, revisión del código, pruebas de seguridad.
- Despliegue*:* Describe una serie de tareas y buenas prácticas a desarrollar una vez el software está desplegado, como, por ejemplo: pruebas de penetración, entorno de software, gestión de configuración y de vulnerabilidades.

BISMM compara prácticas de diferentes organizaciones y la filosofía ágil es usada de forma amplia en distintos tipos de empresas, por lo que se podría decir que BISMM también lo tiene en cuenta y las prácticas y dominios también podría considerarse a la hora de desarrollar *software* de forma iterativa.

OSSA

OSSA (*Oracle Software Security Assurance*) se trata de la metodología propuesta por Oracle que permite garantizar que los productos cumplan con cada uno de los requisitos que se enfocan en seguridad. Incluye una guía de buenas prácticas que los desarrolladores deberían tener en cuenta en el momento de producir un código basado en estándares de seguridad y

enfocado en diseños básicos, proporcionando una orientación en cuanto a la validación de datos, privacidad de la información, administración de accesos.

Los estándares de codificación segura se definen como un componente clave de esta metodología en la cual se evalúa que se cumpla estos criterios en toda la vida útil de los productos de Oracle.

OWASP

OWASP (*Open Web Application Security Project*) (OWASP, 2021) es una organización sin ánimo de lucro que educa a los equipos de desarrollo *software* a aprender, desarrollar, operar y mantener aplicaciones seguras. Los proyectos que gestiona OWASP se dividen en dos categorías principales: proyectos de desarrollo y proyectos de documentación.

A continuación, se tendrán en cuenta algunos de esos proyectos que se consideran más importantes y relacionados con el tema que nos atañe.

OWASP Top 10 Web Application Security Risks

OWASP Top 10 es uno de los proyectos más conocidos de esta organización. Consiste en un informe en el que se detallan los 10 riesgos de seguridad más críticos a los que se enfrentan las organizaciones. Lo que OWASP persigue con este informe es crear conciencia de algunos de los riesgos más importantes y que se tengan en cuenta por las distintas organizaciones a la hora de crear software. Estos son los 10 riesgos más importantes según OWASP:

- A1 Injección
- A2 Pérdida de autenticación y Gestión de sesiones
- A3 Exposición de datos sensibles
- A4 XML Entidad Externa de XML (XXE)
- A5 Pérdida de Control de Acceso
- A6 Configuración de seguridad incorrecta
- A7 Cross-Site Scripting XSS
- A8 Deserialización Insegura
- A9 Uso de componentes con vulnerabilidades conocidas
- A10 Registro y Monitorización insuficientes

OWASP SKF

OWASP Security Knowledge Framework es una aplicación web de código abierto que explica los principios de programación segura en diferentes lenguajes de programación. El objetivo principal de este proyecto es enseñar e integrar la seguridad desde el diseño en el desarrollo de software. Esta iniciativa incluye listas de verificación (usando OWASP-ASVS/OWASP-MASVS o listas de verificación personalizadas) y laboratorios prácticos para la verificación de la seguridad en las aplicaciones (SKF-Labs, OWASP Juice-shop y ejemplos de buenas prácticas en codificación).

OWASP WSTG

OWASP Web Security Testing Guide (WSTG) es un proyecto que provee una serie de recursos para las pruebas relacionadas con ciberseguridad en aplicaciones web. Esté proyecto está orientado a desarrolladores y profesionales de la seguridad. WSTG provee información más especifica y técnica sobre los controles de seguridad que se deben contemplar a la hora de realizar pruebas de penetración en aplicaciones web. Además de WSTG, existe *OWASP Web Aplication Penetration Checklist*, que ayuda a contabilizar y categorizar los controles de seguridad que son necesarios teniendo en cuenta la información que provee OWASP Top 10 y OWASP WSTG.

OWASP CLASP (*Comprehensive Lightweight Application Security Process*)

OWASP CLASP fue desarrollado por John Viega, experto en seguridad de software y arquitecto jefe de seguridad y vicepresidente de McAfee. La idea central de esta metodología es incluir la seguridad en cada fase del ciclo de vida.

CLASP consta de un conjunto de actividades que se centran en la seguridad. Estas actividades se pueden integrar en el proceso de desarrollo de cualquier proyecto. CLASP brinda un enfoque orientado a la recomendación y buenas prácticas y proporciona documentación continua de actividades que las organizaciones deben realizar para mejorar la seguridad. Algunas de las 30 actividades clave incluyen lo siguiente (OWASP,2016):

- Monitorizar las métricas de seguridad
- Identificar las funciones y los requisitos del usuario
- Investigar y evalúar las soluciones de seguridad
- Realizar análisis de seguridad del diseño del sistema

- Identificar e implementar pruebas de seguridad.

Estructuralmente, CLASP se descompone en 5 elementos:

a) Perspectivas de alto nivel: 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) Buenas prácticas: son actividades de seguridad, por ejemplo, evaluación del rendimiento de las aplicaciones, definición de requerimientos de seguridad, implementación de prácticas de desarrollo seguro, construcción de procedimientos de resolución de vulnerabilidades o publicación de guías operacionales de seguridad.
c) Actividades: se trata de una serie de actividades diseñadas para permitir una fácil integración entre actividades de seguridad y el SDLC.
d) Recursos: que ayudan a la planificación, ejecución y cumplimiento de las actividades relacionadas con la seguridad del software.
e) Taxonomía: clasificación de alto nivel 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.

Estás buenas prácticas y principios pueden utilizarse o ser aplicados en el ciclo de desarrollo del software de forma general independientemente de la filosofía que se siga.

OWASP SAMM

OWASP Software Assurance Maturity Model (SAMM) es un modelo de madurez que permite a las organizaciones analizar y mejorar la seguridad de su software. Este modelo está asociado al todo el ciclo de vida del software y es independiente del la tecnología o los procesos utilizados. Se basa principalmente en:

- Evaluar las prácticas de seguridad existentes
- Construir un programa de seguridad en iteraciones bien definidas
- Demostrar mejoras concretas para el aseguramiento del software
- Definir y medir las actividades relacionadas con seguridad

El modelo de madurez para el aseguramiento de Software (SAMM) que sirve como marco de trabajo para exponer e implementar estrategias de seguridad para el software, fue definido para ser flexible y que pueda ser aplicado en toda organización (Chandra, 2013)

Ilustración 2: SAMM / Software Assurance

Fuente: (Chandra, 2013)

Figure 4. SAMM / Software Assurance

Como se ha comentado anteriormente, SAMM es independiente de la metodología, tecnología o procesos que se lleven a cabo. Hace hincapié en ser Ágil agnóstico, esto significa que no tiene en consideración en ningún momento los principios y valores definidos por el Manifiesto Ágil. Sin embargo, sí que tiene en consideración los avances que se están dando en la automatización de prácticas y procesos de *software*. Los proyectos de OWASP trabajan en herramientas y software que ayudan en la implementación de SAMM.

OWASP Enterprise Security API

La ESAPI es una API que provee una colección gratuita y abierta de todos los métodos de seguridad que un desarrollador necesita para construir una aplicación web segura. Cabe destacar que esta API es independiente del lenguaje, sin embargo, las primeras versiones del proyecto están orientadas al lenguaje de programación Java y una referencia de implementación Java.

NIST 800-64

Iniciativa del NIST (*National Institute of Standards and Technology* de Estados Unidos), provee indicaciones para evangelizar a la organización acerca de la importancia de la

seguridad informática; proteger el *software* de uso habitual ante hipotéticos ataques; orquestar un desarrollo seguro de *software*; y detectar y solucionar con rapidez cualquier vulnerabilidad.

NIST-800-64 se caracteriza por identificar consideraciones de seguridad, funciones y responsabilidades, en cada fase del SDLC teniendo en cuenta los siguientes parámetros:

- Fase
- Identificación de riesgos de seguridad*:* permiten verificar a la organización que se está considerando la seguridad en cada fase, así como que da la oportunidad de confirmar que se está implementando la seguridad de forma adecuada y que la organización es consciente de los riesgos actuales y cómo solucionarlos.
- Identificación y descripción de las actividades mayores de seguridad: por cada fase, describiendo la actividad en sí misma, los resultados objetivos que se persigue y las dependencias con otras actividades o tareas claves

Seven Touchpoints for Software Security

Cuando se tratan temas de seguridad de *software*, es frecuente encontrar referencias sobre Gary McGraw, por esa razón hemos considerado interesante incluir una de sus teorías sobre seguridad. McGraw, en su libro *Building Security*, identifica las que a su juicio son las 7 prácticas de seguridad más destacadas que denomina puntos de contacto. Estos puntos de contacto se definen en base a la efectividad y experiencia se ordenan de la siguiente manera: (McGraw, 2006):

1. Revisión del código
2. Análisis de riesgos en la arquitectura
3. Pruebas de penetración
4. Pruebas de seguridad basadas en el riesgos
5. Casos de abuso
6. Requisitos de seguridad
7. Operaciones de seguridad

Además de estos 7 puntos que pueden ser realizados internamente, McGraw añade un octavo que consiste en un análisis externo e independiente, que consistiría en la realización de

pruebas, evaluaciones y revisiones de software por organizaciones o personal externo a la empresa o proyecto.

Secure Code Standards

Secure Code Standards son una serie de reglas y guías que ayudan a prevenir vulnerabilidades de seguridad. Usadas de forma adecuada ayuda a prevenir, detectar y eliminar errores que pudieran comprometer la seguridad del *software*.

- CWE (*Common Weakness Enumeration*) (CWE, 2021): consiste en una lista de debilidades de seguridad asociadas a *software* y *hardware* y que incluye diferentes lenguajes de programación. Esta lista es generada a partir del *feedback* que se obtiene en la comunidad CWE. Además, cabe destacar el CWE Top 25, que es una iniciativa que engloba las debilidades más extendidas y críticas existentes y que podrían convertirse en vulnerabilidades de seguridad severas.
- CERT *Coding Standards* (CERT Coding Standards, 2021): está orientada a dar soporte en la codificación a lenguajes de programación de uso extendido como C, C++ y Java. Junto a cada directriz que se define, se asocia una evaluación del riesgo cuyo fin es ayudar a determinar el grado de riesgo asociado con esa recomendación específica.
- CVE (CVE, 2021): es una lista de vulnerabilidades conocidas en productos software. Auna información de diferentes bases de datos de vulnerabilidades y ayuda a los consumidores a comparar herramientas y servicios desde la perspectiva de seguridad.
- NVD (NVD, 2021): es un repositorio del gobierno de Estados Unidos que está conectado con CVE y lo extiende incluyendo información sobre cómo arreglar vulnerabilidades, da puntuaciones sobre la severidad de la vulnerabilidad y ratios de impacto. Para calcular el puntuación de severidad se hace uso de CVSS.
- CVSS (*Common Vulnerability Scoring System*) es un estándar abierto que permite asignar una severidad o impacto a las vulnerabilidades del software.

Modelo SQUARE

La ISO 25000 o modelo SQUARE (*System Quality Requirements Engineering*) considera la calidad del *software*, pero una de las características que definen esa calidad es la seguridad.

Este modelo define seguridad como la capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no pueda leerlos o modificarlos. SQUARE subdivide la característica de seguridad en 5 subcategorías:

- Confidencialidad.
- Integridad.
- No repudio.
- Responsabilidad.
- Autenticidad.

El modelo SQUARE además propone varios pasos para construir modelos de seguridad desde las etapas tempranas del ciclo de vida del *software*, se hace un análisis enfocado a la seguridad, los patrones de ataque, las amenazas y las vulnerabilidades y se desarrollan malos casos de uso/abuso. El modelo permite indagar, categorizar y priorizar los requisitos de seguridad en aplicaciones y sistemas a través de una serie de pasos que define:

1. Acuerdo sobre las definiciones: el equipo técnico y los interesados en el proyecto llegan a un acuerdo sobre definiciones técnicas que ayudarán y servirán como base para comunicación durante el proyecto.
2. Objetivos de seguridad: es necesario alienar los objetivos de seguridad del proyecto con los objetivos de negocio.
3. Desarrollar artefactos que cumplan con los objetivos de seguridad definidos
4. Realizar una evaluación de riesgos*:* esta evaluación de riesgos debe incluir los riesgos de seguridad y éstos deben ser priorizados.
5. Seleccionar una técnica de obtención de información
6. Indagar en los requisitos de seguridad: recoger los requisitos de seguridad iniciales que se deban tener en cuenta para implementar los objetivos y las definiciones de seguridad descritas.
7. Catalogar los requerimientos en los niveles adecuados.
8. Priorizar requisitos de seguridad
9. Revisión de requisitos: esta segunda revisión debe incluir la revisión de los requisitos de seguridad.

Es interesante remarcar que existe un estudio que incluye una propuesta sobre cómo integrar los 9 pasos que define SQUARE en DSDM (Mead, et al., 2008)

ISO 21827 (SSE-CMM)

La norma ISO 21827 es un estándar internacional Ingeniería de Seguridad de Sistemas creado por la ISSEA (*International Systems Security Engineering Association).* El modelo SSE-CMM (*Secure Software Engineering Capability Maturity Model*) presume de contener las mejores prácticas en cuando a seguridad. Este modelo formaliza el trabajo de seguridad en un exhaustivo conjunto de procesos de seguridad que ha servido como base a muchos organismos regulatorios.

SSE-CMM especifica un Modelo de Madurez de la Capacidad de Ingeniería de la Seguridad de Sistemas (SSE-CMM) que es una herramienta para evaluar y ayudar a mejorar las prácticas y métodos de seguridad de una organización, describe las características esenciales que llevan al éxito a una organización en cuanto a sus procesos de ingeniería de seguridad. ISO/IEC 21827no define un proceso o secuencia particular, lo que hace es recopilar buenas prácticas utilizadas por las organizaciones. El modelo es una métrica estándar de prácticas de ingeniería de la seguridad que cubre los siguientes puntos:

- Ciclo de vida del proyecto, incluyendo las actividades de desarrollo, operaciones, mantenimiento y desmantelamiento.
- Actividades de la dirección, la organización y la ingeniería, teniendo en cuenta a toda la organización
- Interacciones concurrentes con otras disciplinas, como *hardware*, *software*, factores humanos, ingeniería de pruebas, gestión de sistemas, operaciones y mantenimiento.
- Interacción con otras organizaciones, incluyendo adquisiciones, gestión de sistemas, certificación, acreditación y evaluación.

ISO/IEC 15408 – Common Criteria

El estándar internacional ISO/IEC 15408 para seguridad informática, tiene como objetivo definir requisitos seguros que les permitan a los desarrolladores especificar atributos de seguridad y evaluar y establecer un nivel de confianza en el grado en el que un producto satisface la funcionalidad de seguridad y ha superado las medidas de evaluación aplicadas.

Common Criteria se basa en la reutilización de requisitos de seguridad en las primeras etapas del desarrollo de *software* de una manera, sistemática e intuitiva, proporcionando un

repositorio de recursos de seguridad e integrando los criterios comunes en el ciclo de vida del *software*.

ISO-27000

La norma ISO 27000 proporciona una visión general de los Sistemas de Gestión de la Seguridad de la Información (SGSI), además es aplicable a organizaciones de todo tipo y tamaño. El *software* y su desarrollo es parte del SGSI por esa razón también se ha tenido esta norma en cuenta.

ISO 27000 define los principios que rigen la seguridad informática: confidencialidad, integridad y disponibilidad. Como consecuencia, de acuerdo con estos tres conceptos tenemos dos tipos de seguridad:

- la personal, que se da a nivel de hackers o cracker,
- la lógica que se da por medio de controles de acceso, autenticación, encriptación, etc.

El Estándar Internacional de Gestión de la Seguridad viene definido por la ISO/IEC 27001 y recoge el conjunto de controles de seguridad que deberían estar implados. La ISO 27001 también determina que es necesario la recopilación de los requisitos de seguridad desde las primeras fases del desarrollo de software.

Como establece el estándar, cada cierto tiempo deben realizarse actualizaciones y mejoras a los planes, lo que se denomina mejora continua, adecuándolos a la introducción de nuevos procesos y tecnologías. Para esto se utiliza el ciclo de Deming o también conocido como PDCA el cual es aplicado de la siguiente forma:

- Planificar (*Plan*): Diseñar o revisar procesos que soportan servicios de tecnologías de la información.
- Hacer (*Do*): Implementación del plan y gestión de los procesos.
- Verificar (*Check*): Medición de los procesos y de los servicios de tecnologías de la información, comparación con los objetivos marcados y generación de informes.
- Actuar (*Act*): Planificación e implementación de cambios para la mejora de los procesos.

ENS (Esquema Nacional de Seguridad)

El Esquena Nacional de Seguridad (ENS, 2015) es aplicable al ámbito de la Administración Electrónica Española y tiene como objetivo definir una serie de medidas organizativas y técnicas de seguridad que permitan proteger la información que se maneja en los servicios que se prestan, de riesgos provenientes de acciones malintencionadas o ilícitas, particularmente ciberamenazas, errores o fallos y accidentes o desastres. El ENS plantea los siguientes puntos dentro de la Administración Electrónica Española:

- Preparar y aprobar la política de seguridad: incluyendo la definición de roles y asignación de responsabilidades
- Categorizar los sistemas: teniendo en cuenta la información manejada y los servicios prestados.
- Realizar análisis de riesgos: incluyendo la valoración de las medidas de seguridad existentes.
- Preparar y aprobar la declaración de aplicabilidad: teniendo en cuenta las medidas de implantación del ENS.
- Elaborar un plan de adecuación para la mejora de la seguridad en base a las posibles mejoras detectadas.
- Implantar, operar y monitorizar las medidas de seguridad encontradas a través de la mejora continua.
- Auditar la seguridad para verificar el cumplimiento de los requisitos de la ENS.
- Obtener y publicitar la conformidad con el ENS.
- Informar sobre el estado de la seguridad.

Cabe destacar que ENS está inspirado en la familia de estándares ISO 27000 y, más concretamente, en la ISO 27001, por lo que su estructura y aplicación responde al modelo de mejora continua (PDCA), considerando análisis de riesgos e implantación de medidas y controles.

AEPD y RGPD

El Reglamento General de Protección de Datos (RGPD), también hemos considerado que es importante destacarlo en este estudio sobre seguridad, ya que se centra la regulación del tratamiento y protección de datos de los ciudadanos europeos

Por otro lado, la Agencia Española de Protección de datos (AEPD, 2019) es la autoridad de control independiente que vela por el cumplimiento de la normativa de protección de datos de carácter personal

Volviendo al tema que nos atañe, si tenemos en cuenta la normativa española y europea, se puede también comprobar el compromiso que existe con seguridad, en este caso centralizada en la privacidad y como se remarca la premisa de privacidad desde el diseño, concepto también introducido por otras muchas normas y metodologías donde se aboga por introducir requisitos de seguridad en las primeras fases de ciclo de vida de desarrollo de software.

Como establece el artículo 25 del RGPD, "es de obligatorio cumplimiento el implementar la protección de datos desde el diseño y ésta es aplicable a todos los responsables del tratamiento con independencia de su tamaño, tipo de datos tratos o la naturaleza del tratamiento".

En esta resolución se reconocía la importancia de incorporar los principios de privacidad dentro de los procesos de diseño, operación y gestión de los sistemas de la organización para alcanzar un marco de protección integral en lo que a protección de datos se refiere. Además, se animaba a la adopción de los Principios Fundacionales de la Privacidad desde el Diseño definidos por Ann Cavoukian.

1. *Proactivo, no reactivo; Preventivo, no correctivo:* este punto se consigue con formación de los empleados. El tener empleados formados y que sepan lo que deben hacer ayudar a prevenir desde el diseño y el desarrollo del software.
2. *La privacidad como configuración predeterminada:* recoger sólo los datos necesarios limitado a la finalidad del tratamiento. Restringir los accesos en base al principio "*need to know*" de forma que los datos sean accedidos únicamente por quienes necesitan utilizarlos. Anonimización y conservación de datos los plazos necesarios.
3. *Privacidad incorporada en la fase de diseño:* requisito necesario en el ciclo de desarrollo.
4. *Funcionalidad total:* pensamiento "todos ganan"
5. *Aseguramiento de la privacidad en todo el ciclo de vida.* La privacidad nace en el diseño y debe asegurarse durante todo el ciclo de vida. Privacidad
6. *Visibilidad y transparencia*

7. Enfoque centrado en el sujeto de los datos

Se ha considerado interesante la inclusión de estos principios que, aunque tienen como objetivo velar por la privacidad de la información, también se podría aplicar a la Seguridad en el Ciclo de Vida del desarrollo de software.

Buenas prácticas en desarrollo seguro

En el apartado anterior se han definido algunas herramientas, normativas y metodologías centrada en el desarrollo de *software* seguro. De este análisis se han podido destacar algunas buenas prácticas y prácticas comunes a la hora de desarrollar *software* seguro. Se ha podido observar que la seguridad no es una cuestión única de los desarrollos de *software* o de los desarrolladores, la organización también tiene que ser consciente de la importancia de la seguridad y aplicar acciones genéricas con sus empleados para hacerles conscientes de su importancia y de la importancia que representa para la organización.

A continuación, vamos a definir algunas de las buenas prácticas relacionadas con seguridad más representativas según lo estudiado hasta ahora y también vamos ha explicar algunos de los problemas más frecuentes con los que se encuentran las organizaciones a la hora de implantar buenas prácticas relacionadas con el aseguramiento de la seguridad del software en sus ciclos de vida:

Buenas prácticas a nivel organizacional

i. Cultura empresarial y Política de seguridad

Es prioritario que se incluya la Seguridad como parte de la cultura empresarial. Esto quiere decir que la organización debe fomentar la seguridad en las actividades que lleva a cabo, facilitando la formación de todos los empleados, independientemente de si trabajan en áreas técnicas o no, además de promover el uso de herramientas de aseguramiento de la seguridad, tanto a nivel desarrolladores como a nivel de usuario de las aplicaciones.

Los empleados deben ser conscientes de qué se espera de ellos, de que la seguridad es algo que debe estar siempre presente en sus tareas, así como de las posibles consecuencias que un fallo de este tipo podría significar para la organización.

En uno de los estudios utilizados en este trabajo (Assal & Chiasson, 2018), se hicieron encuestas a personal técnico de diferentes empresas sobre las medidas y buenas prácticas de seguridad que utilizaban en sus proyectos. Algunos de los desarrolladores encuestados no se sentían responsables de la seguridad de las aplicaciones ni de la creación de aplicaciones seguras, de hecho decían que sus empresas no les demandaban ningún tipo de conocimiento

técnico. Esto viene reflejar la falta de importancia por la seguridad dentro de la compañía, algo que es básico para extrapolar la seguridad a las diferentes áreas.

ii. Formación en seguridad

En ocasiones, los desarrolladores son vistos como el eslabón más débil en cuanto a la seguridad software, ya que los problemas y responsabilidades en cuanto a la seguridad siempre recaen en ellos (Green & Smith, 2016). Por ello, es necesario su formación técnica en seguridad, para que se tenga en cuenta desde la toma de requisitos en el comienzo del desarrollo para así aplicar la seguridad desde el principio, saber detectar y definir los riesgos y la forma de solventarlos. También es importante que estén actualizados sobre técnicas y herramientas existentes para detectar y prevenir vulnerabilidades lo antes posible y utilizarlas en las diferentes fases del ciclo de vida de desarrollo de *software*.

Nuevas amenazas y vulnerabilidades son detectadas continuamente, por lo que es necesario una formación continua, metodologías como Microsoft SDL incluyen la formación como elemento indispensable en el principio del Ciclo de Vida de Desarrollo de Software, haciendo formaciones de reciclaje de forma anual por defecto.

iii. Defense in Depth.

La estrategía de Defensa en profundidad o *Defense in Depth* consiste básicamente en ser conciente de la seguridad en las distintas capas de la organización: usuarios finales, diseño del producto, seguridad de sistemas y redes, así como en las diferentes fases del SDLD. De esta forma, si una vulnerabilidad escapa a una de las fases de desarrollo, la probabilidad de detectarla y solventarla podrá ser posible en fases posteriores.

En contraposición a la estrategia de Defensa en profundidad, se encuentra la conocida como estrategia de Simplicidad en seguridad (*Simplicity-in-security)*, que destaca que demasiadas medidas de seguridad pueden llevar a problemas y huecos que incluyan vulnerabilidades.

Buenas prácticas en el Ciclo de vida de Desarrollo del Software

i. Aplicar Seguridad a todas las aplicaciones.

En ocasiones, se dedica un mayor esfuerzo de seguridad en aplicaciones que representan un riesgo mayor que en las que se determinan como de bajo riesgo, es decir, aplicaciones que

tratan con datos de clientes o proveedores o aplicaciones de las que depende el negocio de la empresa. Esto es un error ya las aplicaciones que en teoría representan un bajo riesgo podrían ser una puerta de entrada para acciones maliciosas en nuestra organización que podría llegar a afectar información sensible o a la disponibilidad del sistema.

La seguridad no debería ser ignorada ni siquiera en las aplicaciones que son catalogadas como de bajo riesgo.

ii. Identificar los requisitos de seguridad

Identificar los requisitos de seguridad de una aplicación desde la fase inicial de recogida de requisitos es importante para evitar problemas más costosos que si se encontraran en fases posteriores. Conocer los requisitos desde el principio permite un mejor análisis y diseño de la solución a implementar.

Identificar los requisitos de seguridad desde el principio también permite evaluar su cumplimiento en las diferentes fases del ciclo de vida del desarrollo software.

Muchas de las metodologías y normas estudiadas en el apartado anterior remarcan este punto.

iii. Seguridad desde el diseño

Tener en cuenta la seguridad desde el diseño permite hacer diseños más simples y que se reduzcan la probabilidad de errores de implementación. Normalmente los diseños más complejos son lo que más errores de implementación pueden tener asociados.

Es necesario diseñar el software teniendo en cuentas las políticas de seguridad y el cumplimiento de los principios básicos de seguridad como son:

- Seguridad por defecto
- Denegación por defecto
- A prueba de fallos
- Mínimo privilegios.

Diseñadores, analistas y arquitectos deben documentar claramente las premisas y decisiones tomadas, así como ser conscientes de los posibles ataques que podría sufrir el *software*. Por esta razón, trabajar en el Análisis de Riesgos desde el inicio del proyecto es importante para

que el equipo de desarrollo pueda tenerlos en cuenta y trabajar en su mitigación. En ocasiones es necesario una revisión externa del diseño por personal independiente al proyecto.

iv. Hardering Sprints

Hardering Sprints (Levison, 2019) es una técnica que se utiliza frecuentemente para corregir o mejorar cosas que se quedan pendientes durante los *Sprints* y vuelven al *Backlog* para ser retomadas más adelante según prioridades. También se pueden considerar en estos *Sprints* tareas como mejorar las pruebas de regresión, hacer pruebas de rendimiento o arreglar defectos. Muchos equipos realizan este tipo de *Sprints* para, en lugar de implementar nuevas funcionalidades, preparar el software para ser desplegado en el entorno de producción.

Existen opiniones a favor y en contra de esta práctica, las opiniones en contra remarcan que si este tipo de *Sprint* son necesarios es porque no se está haciendo correctamente *Scrum*. Con una completa Definición de Hecho (DoD) este tipo de trabajo de rematar detalles no debería ser necesario.

Otras opiniones al respecto también remarcan que esta práctica puede llevar a generar una mayor deuda técnica o *technical debt*, ya que se dará por hecho que se va a tener *Sprints* posteriores para completar tareas que no se han desarrollado: aumentará el volumen de código sin testear, defectos más difíciles de arreglar, mayor número de defectos, pruebas de regresión más complicados e incluir fragilidad en el *software*.

v. Environmental hardering

La práctica de *environmental hardering* ayuda a reducir las vulnerabilidades heredadas del *software* de terceros. Consiste básicamente en actualizar las versiones de *software* de otros que utilizamos, incluyendo mejoras y arreglos: sistema operativo, software de redes, librerías de terceros, etc.

En general, utilizar las últimas versiones ayuda a estar más seguros en un área que está en continuo cambio.

vi. Utilizar Software validado

Utilizar únicamente herramientas, APIs y frameworks aprobados y que se hayan evaluado como seguros y efectivos.

vii. Análisis estático SATs (Static Analysis Tools) y escaneo de vulnerabilidades

Este tipo de herramientas permite verificar la conformidad del *software* con los estándares sobre codificación de software. Permite evaluar algunos rasgos o buenas prácticas en el código como la validación de las entradas de datos, los datos enviados a terceros, contraseñas embebidas en el código, excepciones no controladas o la utilización de funciones inseguras o obsoletas.

Muchos desarrolladores lo utilizan como un paso inicial a la hora de optimizar código y asegurar que sea fácilmente legible antes de pasar a la revisión del código. Aunque, en general, se considera que este tipo de análisis no es suficiente para asegurar la seguridad del software y que puede llevar a consumir mucho tiempo del desarrollo debido a la gran cantidad de falsos positivos que deben ser analizados.

Se puede incluir dentro de este punto el escaneado de vulnerabilidades en librerías de terceros utilizadas en el proyecto. Existen herramientas, como parte de la construcción del software que analizan las librerías utilizadas y confirman si hay alguna vulnerabilidad conocida y su nivel de amenaza, para ello, hace uso de las listas de vulnerabilidades reportadas por organismos reconocidos como son CVE, NVD o CVSS, entre otros.

viii. Revisiones de código de riesgo

Cuando se cataloga una funcionalidad como de riesgo, porque afecte el acceso a datos, la forma de encriptación o cualquier otro área sensible, es recomendable hacer una revisión más en profundidad de esos cambios e involucrar a personal experto en seguridad.

ix. Seguimiento y arreglo de bugs de seguridad

El seguimiento y arreglo de bugs de seguridad es una práctica muy importante, sin embargo, si se es capaz de detectar una vulnerabilidad y no se aborda, no sirve de nada.

Cuando se detecta un bug de seguridad se le debe dar la prioridad necesaria para ver el nivel de riesgo y el esfuerzo que requeriría su implementación. Esto permitirá tomar la decisión de

si se debe resolver y con qué prioridad, o se decide asumir el riesgo y crea un plan de acción o contingencia en caso de que la amenaza se viera materializada.

x. ***Fuzzing***

Es una técnica de pruebas de software (Wikipedia, s.f.), a menudo automatizada o semiautomatizada, que implica proporcionar datos inválidos, inesperados o aleatorios a las entradas de un programa de ordenador para ver cómo se comporta el sistema. Esto permite monitorizar caídas, incongruencias en las respuestas del código o potenciales filtraciones de memoria.

La detección de vulnerabilidades mediante *fuzzing* se realiza en base a pruebas altamente intrusivas que buscan causar inestabilidad en el sistema, comportamientos extraños o incluso la denegación total del mismo

La técnica de *fuzzing* se utiliza normalmente para descubrir problemas de seguridad en *software* y *hardware* que, aunque da buenos resultado, no asegura que se encuentren todos los posibles errores existentes en el sistema.

xi. Code reviews

Code review o *Peer Code Review*, es una práctica muy extendida en entornos de desarrollo ágiles dónde se da mucha importancia a la colaboración y consiste en que otros compañeros realicen una revisión el código implementado por una persona buscando defectos, así se tendrán diferentes puntos de vista sobre un mismo desarrollo y permitirá discernir si la solución aportada es la más adecuada o no.

Actualmente existen herramientas dentro de los propios sistemas de control de versiones, que permiten añadir uno o varios revisores al código subido para su aceptación antes de integrarlo en la rama maestra.

xii. Perform post-development testing

Dado que las herramientas de análisis estático y las revisiones de código pueden no ser suficientes para detectar todos los errores o vulnerabilidades en el código, es recomendable

hacer otro tipo de pruebas por ejemplo pruebas de penetración o revisiones externas de seguridad que puedan aportar nuevos puntos de vista.

xiii. Threat modeling

Thread modeling o modelado de amenazas es un análisis de los riesgos de seguridad a los que nuestra aplicación va a estar expuesta una vez esté implantada en el entorno de producción. La idea básica de esta táctica es encontrar los puntos vulnerables y las medidas que serán necesarias tener en cuenta para evitar que las posibles amenazas detectadas se manifiesten y si se produjeran que tengamos la forma de reaccionar de forma rápida y mitigar el riesgo.

El modelo de amenazas se contempla en metodologías estudiadas anteriormente como Microsoft SDL o OWASP y además se incluye dentro de la Gestión de Riesgos que ya hemos visto en varias normas y estándares como la ISO 27000.

Se trata de un documento que se debería comenzar a desarrollar desde las primeras fases del ciclo de vida de desarrollo software y que seguirá vivo durante toda su vida. Nos permitirá saber vulnerabilidades a las que estamos expuestos, probabilidad de que ocurran, daño potencial si la amenaza tiene éxito, prioridad o importancia y cómo vamos a abordar el problema.

Este modelo de amenazas nos permitirá detectar y ser consciente de las prácticas de seguridad en las diferentes fases del ciclo de vida de desarrollo del software y la información sobre el mismo será continuamente revisada y actualizada teniendo en cuenta los procesos de mejora continua.

El modelado de amenazas tiene especial importancia en las vulnerabilidadesque se han detectado, que no son cubiertas desde el diseño y que solventarla puede requerir un gran esfuerzo de rediseño en caso en que se encuentre en fases posteriores del ciclo de desarrollo.

Si aplicamos la filosofía Ágil al modelo de amenazas, habría que remarcar el principio de no planificar más trabajo del que se completará en el *Sprint* actual. Este principio también se puede adaptar al modelo de amenazas, no es necesario crear un modelo de amenazas detallado de la aplicación completa durante el primer *Sprint*, sólo la funcionalidad que se está desarrollando en el *Sprint* actual debe modelarse como amenaza en ese Sprint concreto.

xiv. Gestión de riesgos

La gestión de riesgos es un punto clave dentro de la ISO 27000 y el SGSI (Sistema de gestión de la seguridad de la información). Este punto se podría abordar junto con *Thread modeling*, definiendo también planes de contingencia y continuidad del negocio en caso de que una amenaza se viera materializadas.

xv. Pruebas de penetración

Una prueba de penetración o *pen test* consiste en un ataque a un sistema con la intención de encontrar debilidades de seguridad en cuanto a funcionalidad y datos. Una prueba de penetración puede determinar si un sistema es vulnerable a los ataques, si las defensas son suficientes. Las pruebas de penetración permiten:

- Determinar la posibilidad de éxito de un ataque
- Identificar vulnerabilidades de alto riesgo que resultan de una combinación de vulnerabilidades de menor exportadas de forma particular
- Identificar vulnerabilidades que pueden ser difíciles o imposibles de detectar con software de análisis de vulnerabilidades (SATs).
- Comprobar la capacidad del sistema para responder a ataques

xvi. Seguridad en los tests

Muchas metodologías de las estudiadas hablan sobre la necesidad de incluir la seguridad en las distintas fases del desarrollo de software, esto incluye especialmente la fase de pruebas y verificación de la funcionalidad. Aunque se tenga en cuenta la seguridad en el diseño y desarrollo de software, es necesario que se verifique como parte de la fase de pruebas utilizando herramientas y métodos necesarios para ello.

La seguridad también debe ser incluida en las pruebas unitarias y pruebas de integración, no únicamente como parte de las pruebas funcionales del sistema.

xvii. Devops

Se conoce como DevOps a la metodología para la creación de software que se basa en favorecer la integración entre desarrolladores de software y administradores de sistemas. La

idea que se persigue es fabricar software más rápidamente, con mayor calidad, menor coste y que permita hacer despliegues en producción frecuentemente.

Esta metodología es un complemento a las metodologías ágiles y su principio iterativo de proveer valor y nuevas versiones del producto frecuentemente. En los siguientes apartados hablaremos más en profundidad sobre DevOps y sobre su variante SecDevOps, que surgió de la idea de trabajar en un marco de colaboración, donde la seguridad se convierte en una responsabilidad compartida e integrada durante todo el proceso de desarrollo.

xviii. Seguridad como responsabilidad compartida:

Abordar la seguridad del software es una responsabilidad colectiva que incluye a todos los implicados en el ciclo de vida del desarrollo de *software*, desde desarrollares a la gente de calidad, diseñadores, arquitectos, *product owners*, *scrum masters*, etc.

Cómo desarrollar software seguro siguiendo metodologías Agiles

El desarrollo de software seguro está sustentado sobre procesos o modelos que en ocasiones son difíciles de integrar dentro de las metodologías Ágiles donde los procesos son más flexibles y adaptables y se aboga por una entrega continua de valor con despliegues frecuentes cada cierto número de semanas.

La idea que se persigue intenta combinar dos enfoques, el desarrollo flexible de software no debe estar reñido con software de calidad y seguro, así como que los procesos relacionados con seguridad no deberían disminuir la efectividad de los procesos de desarrollo de software.

Manifiesto de Seguridad Ágil

Al igual que existe el Manifiesto Ágil, que nos proporciona los valores y principios que sustentan a todas las metodologías ágiles, existe un Manifiesto de Seguridad en Ágil (*The Agile Security Manifesto*) (Chinnaraju, 2017). Este manifiesto surgió del problema visto anteriormente de que las actividades de seguridad eran realizadas fuera del proceso de desarrollo de software, por lo que los principios que define este manifiesto tratan de incluir la seguridad del software dentro de la metodología ágil. Para ello, el manifiesto propone añadir los siguientes principios relacionados con seguridad al ya conocido Manifiesto Ágil:

- *Confiar en los desarrolladores y probadores más que en los especialistas en seguridad:*

Este principio se basa en la idea de que la seguridad es parte del trabajo del Equipo Ágil. La existencia de Expertos en Seguridad es útil, pero puede no resultar práctica a la hora de realizar un desarrollo flexible e iterativo. Por un lado, no todos los equipos de desarrollo ágiles pueden tener un especialista, de hecho, se debe abogar por perfiles multifuncionales en los que todos los miembros del equipo puedan llevar a cabo cualquier tipo de tarea. Iría en contra de la metodología el tener que esperar a que el especialista en seguridad revise los cambios antes de poder incluirlos en el entorno de Integración Continua y Despliegue Continuo (CI/CD – Continuous Integration / Continuous Deployment).

La seguridad debería ser integrada en el proceso de forma continua y a poder ser automatizada, por ello el propio equipo Ágil debe ser el responsable de la seguridad del

software, de la misma forma que es responsable de que este funcione perfectamente, de que sea fiable y se adecue a las características que necesita el usuario final.

Por otro lado, la imagen del especialista en seguridad puede ser de utilidad en ciertas tareas. El experto en seguridad puede ayudar al Equipo Ágil a definir los objetivos de seguridad y cómo conseguirlos y medir los resultados, por ejemplo, debería estar involucrado en la definición de las historias de usuario y en la priorización del *Backlog*. También pueden ser de utilidad en tareas muy específicas que necesiten de experiencia y un conocimiento más concreto, como puede ser ayudar a integrar herramientas de seguridad en los entornos de desarrollo.

- *Asegurar mientras desarrollamos mejor que cuando el trabajo está hecho*

Este principio se basa en la idea de integrar la seguridad dentro del proceso de desarrollo Ágil, las actividades relacionadas con seguridad no deberían parar al equipo Ágil, cuando el equipo está trabajando, la idea de seguridad debería estar constantemente presente.

Si se tiene un especialista en seguridad o un departamento de seguridad, las tareas que surjan y la relación con el equipo de desarrollo deben seguir los procesos determinados por el equipo Ágil y utilizar las herramientas que ellos utilicen. Como resultado, la herramienta que el equipo de desarrollo utilice para planificar y gestionar el trabajo, debe incluir también prioridades relacionadas con seguridad.

Detectar un defecto de seguridad cuando el software ya está desarrollado puede ocasionar costes altos a la hora de arreglarlo o la necesidad de buscar una estrategia para mitigarlos, por ello es recomendable tener los posibles riesgos de seguridad localizados ya en el diseño, y así diseñar e implementar una solución con estos posibles riesgos en mente.

- *Implementar las características de manera segura mejor que añadir características de seguridad*

En el desarrollo Ágil nos centramos en dar valor a negocio no en implementar únicamente características de seguridad. La seguridad debería ir implícita en cada tarea que se lleva a cabo, lo que se denominaría “Seguridad por Defecto”, como ya se ha visto en las buenas prácticas estudiadas anteriormente.

Por otro lado, este principio también se centra en no reinventar la rueda, es decir, utilizar soluciones que funcionan realizadas por especialistas en lugar de implementarlas nosotros mismos: algoritmos de encriptación, sistemas de gestión de usuarios y autenticación.

- *Mitigar los riesgos mejor que arreglar fallos*

Hay que empezar con la premisa de que vamos a trabajar en el desarrollo del software seguro. Sin embargo, es posible que se comentan errores que puedan ponernos en riesgo. Estos riesgos no son siempre sencillos de arreglar, por lo que es necesario encontrar una forma de mitigar el impacto. La Gestión de riesgos nos ayuda a conocer los riesgos a los que se expone nuestro software y priorizarlos.

En ocasiones, se escucha que los equipos ágiles trabajan bajo la premisa de 'hacer lo mínimo que pueda funcionar'. Esto también aplicaría en seguridad, pero la solución más simple que se puede ofrecer podría no estar relacionada con código. Quizá una solución del lado de negocio, es decir, un proceso de negocio o monitorizar los sistemas, etc., podría ayudar a mitigar le impacto asociado al riesgo.

La filosofía Ágil, plantea una forma muy diferente de trabajar si la comparamos con las metodologías tradicionales. A continuación, se definen algunas directrices que se consideran fundamentales a la hora de incluir la seguridad en el desarrollo *software* de forma ágil. Cabe destacar que los puntos hablan sobre temas concretos y que cada equipo o proyecto podrá adaptarlo a su metodología según vean se adapte mejor hacerlo de una forma u otra, ahí radica la fortaleza de la adaptabilidad de la filosofía Ágil.

Roles y habilidades

Los equipos ágiles se caracterizan por estar formados por roles multifuncionales, esto quiere decir que cada miembro del equipo debería poder hacer cualquier tarea de la misma forma que cualquier otro compañero.

Si queremos que un equipo ágil desarrolle software de calidad y seguro, es necesario que se el equipo tenga los conocimientos necesarios para poder desempeñar su trabajo de forma efectiva, esto es, *software* que funcione como se espera, de calidad y seguro.

La formación es un pilar básico en general en el desarrollo de *software*, pero más aún para un Equipo Ágil. Si hablamos de seguridad, es necesario que esta formación sea de calidad, adaptada a las necesidades del proyecto o la empresa y periódica, ya que las amenazas de seguridad cambian constantemente.

La seguridad debería estar incluida dentro de la cultura empresarial y hacer a todos los empleados partícipes de su importancia, desarrollando seminarios, definiendo directrices a la hora de trabajar y dando necesario soporte para que se incluya en todos los proyectos de la empresa.

Tal y como se ha visto como parte de los principios ágiles de seguridad, no es buena idea tener un role de especialista en seguridad dentro del equipo, ya que podría convertirse en un cuello de botella, se recomienda más que esa figura pueda actuar como soporte del equipo ágil de forma externa y que actúe como *Product Owner* sobre la seguridad el sistema o del producto.

Refinamiento de historias: Pensar como un tipo malo.

El desarrollo ágil se centra en la velocidad y la flexibilidad, por lo que es necesario que el equipo sea capaz de adelantarse a posibles amenazas que el *software* podría sufrir. Esto significa afrontar el desarrollo de historias y su refinamiento pensando en el punto de vista de los posibles atacantes de nuestro sistemas o hackers, alguien que intenta obtener acceso a partes privadas de nuestro sistema a través de un agujero de seguridad. Es por esto por lo que es importante que el equipo analice el proyecto y las implicaciones de seguridad desde las distintas fases del proyecto, incluyendo los inicios, el desarrollo y las pruebas.

Para conseguir esto es necesario que los miembros del equipo estén formados y tengan a su disposición las herramientas adecuadas para poder evitar los problemas de seguridad en código más comunes.

Esto no significa que no tenga que haber equipo de seguridad, el que el equipo ágil cubra los errores de seguridad y tenga conocimientos en seguridad ayudará a que el código y las aplicaciones sean más seguras y que el número de vulnerabilidades se vea reducido y que el equipo de seguridad se pueda centrar en otro tipo de tareas o nuevas amenazas que vayan apareciendo.

Criterios de Aceptación de Seguridad (DoD) vs Historias de seguridad

En ocasiones se habla de historias de usuario de seguridad, pero no es normal que un usuario se preocupe por cierto tipo de vulnerabilidades, por lo que es más común es que este tipo de criterios aparezcan en la parte de refinamiento de las historias y se incluya la seguridad y ciertas buenas prácticas dentro de los criterios de aceptación de una historia de usuario y como parte de su definición de hecho. Se considera que la seguridad se tratará de dos formas a la hora de definir historias:

- Historias de seguridad, que más que historias se tratará de tareas de seguridad que es necesario hacer y que debemos poder priorizar e incluir en los *Sprints*.
- Criterios de aceptación de seguridad o incluir seguridad en la definición de hecho: la seguridad así se integra dentro de cada historia de usuario.

Existen diferentes enfoques a la hora de incluir la seguridad en las historias de usuario:

- Security Stories: este enfoque ha sido creado por SAFECode y consiste en un conjunto de historias de seguridad que siguen el patrón de creación de historias de usuario, pero que, en lugar de estar orientadas al usuario, están orientadas el equipo de desarrollo (arquitectos de software, desarrolladores o probadores). Estás historias incluye la protección contra ataques típico tipo XSS, CSRF, OS command injection, SQL injection, ataques por fuerza bruta, gestión de sesiones, gestión de excepciones, etc. No son historias que resultaran en una funcionalidad palpable y ejecutable, son requisitos no funcionales que se deben de tener en cuenta a la hora de implementar el código, pero que no se pueden considerar historias como tal. Estás puntualizaciones deberían estar incluidas como criterios de aceptación o como parte de la definición de hecho. Por ejemplo: “Como desarrollador, quiero que la información sensible se mantenga restringida a usuario que no tengan los permisos de acceso apropiados”.
- *Evil User Stories:* OWASP en cambio, opta por la creación de historias que llamen la atención sobre los posibles problemas que pueda causar. La idea es remarcar lo que un hacker podría hacer para comprometer nuestro sistema. Por ejemplo, “Como hacker, puedo leer y modificar toda la información de entrada y salida de tu aplicación”
- *Historias abusivas:* este es el enfoque que propone Dr. Gary McDraw y consiste en, a la hora de refinar las historias de usuario, en lugar de únicamente darle el enfoque del

usuario que hace bien su trabajo, pensar en la perspectiva de cosas que un usuario no haciendo bien su trabajo no se quiere que pueda llegar a hacer, por ejemplo, acceder a información confidencial sobre la que no tiene acceso, romper el sistema por introducir datos erróneos, etc.

Cada uno de estos enfoques es válido y puede ser útil a la hora de definir la seguridad en las historias de usuario, cada equipo elegirá el que más le convenga o los podrá mezclar de la forma que considere que mejor le puede ayudar a la hora de desarrollar software seguro.

Secure Scrum

Secure Scrum (Nguyen, 2015) es un método para integrar Seguridad en Scrum, se trata de una variación del *framework* que se centra en el desarrollo de *software* seguro a lo largo del Ciclo de vida de desarrollo de software (Pohl & Hof, 2015).

Secure Scrum remarca 4 componentes que deben integrarse dentro del marco *Scrum* tradicional:

- Identificación: consiste identificar aspectos relacionados con la seguridad en las historias de usuario definidas por el *Product Owner* o por los clientes. Estas historias son marcadas como relativas a seguridad con una señal específica denominada S-Mark y se le asocia un número, que determina la severidad que se le asocia a ese problemas o vulnerabilidad detectada. Dependiendo de la severidad existen unos rangos (S-Tags) que permiten relacionar la incidencia con la severidad asociada. Esta severidad asociada a las historias puede variar según avance el proyecto y evolucione el producto como parte de las Reuniones de Refinamiento.
- Implementación: cuando exista una historia marcada con el S-Mark y el S-Tak, se deberá abordar, o bien buscando una solución por código o si fuera muy complejo algún tipo de acción que permita mitigarla. Por consiguiente, en este punto se actualizaría el Modelo de Amenazas y el Plan de Gestión de Riesgos.
- Verificación: Cuando una historia está marcada con S-Mark, se tiene que prestar especial atención a las pruebas y correcta verificación del arreglo.
- Definición de Hecho (DoD): Se añadirá la verificación como parte de la Definición de Hecho.

Code reviews

Las revisiones de código son importantes para poder detectar errores. Podríamos dividir estas revisiones en dos tipos, automáticas y manuales

- *Revisiones código automáticas:* Se podrían considerar revisiones de código automáticas a la utilización de herramientas de análisis estático (SATs) que analizar que el código cumpla con ciertos estándares de codificación. Los SATs no son infalibles, por lo que esto además se podría complementar con pruebas unitarias, realizadas por los desarrolladores e incomparadas a las comprobaciones automáticas que se realizan del código frecuentemente.
- *Revisiones de código manuales:* este tipo de revisiones es de utilidad porque permite tener otro punto de vista sobre una implementación diferente del nuestro, lo que puede llevar a soluciones y código más completo y de mejor calidad. Un tipo muy utilizado de revisiones de código manuales podría ser también las revisiones por parejas (*Peer review*) del código nuevo que también permitiría obtener otro punto de vista, pero al estar juntos en pareja revisando el código la comunicación es más rápida y efectiva.

Modelado de Amenazas y Plan de Gestión de Riesgos

El modelado de Amenazas nos permite conocer las formas en las que nuestro sistema puede verse afectado, conocer nuestros puntos débiles y cómo mitigarlos o bien por código o bien por otros medios.

Se trata de un documento que se debería comenzar a desarrollar desde las primeras fases del Ciclo de Vida de Desarrollo Software y que seguirá vivo durante toda su vida. Nos permitirá saber vulnerabilidades a las que estamos expuestos, probabilidad de que ocurran, daño potencial si la amenaza tiene éxito, prioridad o importancia y cómo vamos a abordar el problema.

El modelado de Amenazas y el Plan de Gestión de Riesgos en una filosofía de trabajo Ágil se irá completando según se vaya añadiendo funcionalidad. Si tenemos en cuenta el proceso de desarrollo SDL/A, visto en punto anteriores, podremos ver que también tiene en cuenta la parte de Modelado de Amenazas y Plan de Gestión de Riesgos y considera algo fundamental el conocer las vulnerabilidades que se sabe que no están cubiertas (ya sea por esfuerzo o falta de tiempo). Lo que propone este método es, en lugar de desarrollar este documento completo

en las primeras fases del ciclo de vida, ir completándolo con la funcionalidad incluida en cada Sprint.

Por lo que, si aplicamos la filosofía Ágil al modelo de amenazas y a la Gestión de Riesgos, habría que enfocarse en el principio de no planificar más trabajo del que se completará en el *Sprint* actual, es decir, no es necesario crear un modelo de amenazas detallado de la aplicación completa durante el primer *Sprint*, sólo la funcionalidad que se está desarrollando en el *Sprint* actual debe modelarse como amenaza en ese *Sprint* concreto. Así pues, se definirá una primera tarea de creación de un documento, diagramas, etc. de Modelado de Amenazas y Plan de Gestión de Riesgos genéricos y esto se irá completando en cada iteración, con cada funcionalidad incluida, en el refinamiento de historias y como parte de las revisiones y retrospectivas de los *Sprints*.

SecDevOps

Al igual que las metodologías de desarrollo han ido evolucionando, nuevas herramientas han ido surgiendo para adaptarse a las nuevas necesidades, de ahí han surgido herramientas que se aseguran de hacer el desarrollo más sencillo, permitiendo integración continua de los nuevos desarrollos y asegurando la seguridad de los sistemas. DevSecOps o SecDevOps se centra en la integración de la seguridad dentro del DevOps. Básicamente consiste en un conjunto de buenas prácticas que integran completamente controles de seguridad en el ciclo de vida de desarrollo de software de forma automatizada.

La seguridad de las aplicaciones, como hemos dicho, es una responsabilidad compartida, SecDevOps permite esta colaboración entre los departamentos de Seguridad, Desarrollo y Operaciones para trabajar en construir entornos más fiables y seguros.

Así, se puede realizar de forma automatizada un análisis del código fuente, para detectar malas prácticas o posibles fuentes de errores, ejecutar automáticamente las pruebas unitarias del proyecto, así como pruebas de integración, e incluso integrar herramientas especializadas en seguridad que escaneen posibles vulnerabilidades en el código.

Además, las operaciones de DevOps nos permiten definir todos los pasos por los que deberá pasar el código de forma automática antes de llegar ser implantado en los entornos de producción. Desde que el código es implantado en el entorno de pruebas, añadiendo herramientas de análisis estático, pruebas dinámicas, etc, hasta llegar a producción donde se

monitorizaría el sistema y se actualizaría el software necesario de forma automática antes de hacer el despliegue en producción.

A continuación de hace un resumen de distintos tipos de herramientas que podemos introducir en el ciclo del software gracias a las prácticas introducidas por DevOps:

- Herramientas de gestión de la configuración: nos permite gestionar todas las versiones de software que se utilizan para poder recrear nuevos entornos de forma rápida y efectiva.
- Gestión de Logs: estas herramientas permiten centralizar *logs*, analizarlos más fácilmente y facilitar la monitorización con la posibilidad de incluir alertas específicas en caso de errores concretos.
- Herramientas de gestión de despliegues: estas herramientas permiten simplificar la multitud de pasos manuales que pueden ser requeridos para hacer un pase a producción. La idea es automatizar todos estos pasos con herramientas y scripts; de esta forma los despliegues serán más rápidos y habrá menos posibilidades de cometer errores por tener pasos manuales.
- Herramientas de control central: este tipo de herramientas permiten gestionar las tareas repetitivas cuando existen varios servidores que tienen que ser actualizados. Estas herramientas serían las que se encargan de que las diferentes tareas necesarias sean ejecutadas en todos los servidores implicados.
- Integración Continua: actualmente estás herramientas se pueden considerar una necesidad en el ciclo de desarrollo de software. Permite incorporar el código fuente nuevo, empaquetar la aplicación, ejecutar test automáticos y generar informes de análisis sobre el código. También permite rápidamente encontrar problemas y los causantes de esos problemas, así como facilitar la tarea de deshacer los cambios que han producido los problemas en la aplicación.
- Monitorización y medidas: existen herramientas que nos permiten monitorizar nuestras instancias y de forma sencilla ayudarnos a encontrar problemas gracias a ciertas medidas o indicadores, por ejemplo, tiempos de respuestas, consumo de CPU, errores 500, excepciones, etc. Esto nos permite reaccionar más rápidamente en caso de un incidente.

- Herramientas de gestión de entornos: permiten generar completamente entornos de forma que facilita el encapsular todo el software y configuración necesaria en un contenedor que puede usarse para replicar entornos siempre que sea necesario.

Propuesta de implementación de Scrum Seguro

Este apartado se va a centrar en definir una propuesta de implementación de la metodología *Scrum* que tenga en consideración la seguridad en cada una de sus fases.

Como ya se ha comentado en varias ocasiones, cuando se siguen metodologías ágiles, la adaptación a las necesidades es una parte crucial, por lo que esto es una propuesta de implementación que podrá ir cambiando en el tiempo según el equipo considere necesario.

Después de cada Sprint tendrá lugar una retrospectiva donde, entre otras cosas, los involucrados podrán dar su opinión sobre el proceso y proponer cambios para mejorarlo y adaptarlo a las necesidades del proyecto en concreto o del equipo.

Scrum Seguro: Consideraciones

1. **Historias y refinamiento**

Para esta propuesta se han tenido en cuenta la información descrita en el apartado anterior, por lo que a la hora de definir historias de usuarios se tendrá en cuenta lo siguiente:

- ***Backlog* de Seguridad:**

Se considera que la seguridad tiene relevancia suficiente para tener su propio *backlog*, este *backlog* tendrá como *Product Owner* a el departamento de Seguridad/Experto en Seguridad y permitirá incluir tareas concretas de seguridad dentro del proyecto. En este *backlog* se incluirán las tareas y actividades genéricas de seguridad que no se puedan asociar a una historia de usuario en concreto, si no que sean generales del proyecto:

i. Plan de Contingencia: respuesta a incidentes
ii. Pruebas de penetración periódicas
iii. Hacking ético
iv. Monitorización de sistema (alertas)
v. Auditorias de código regulares.

Este *backlog* tendrá una reunión de refinamiento propia, donde expertos en seguridad y el equipo de desarrollo tratarán las tareas marcadas con el S-Tag, se les asignará la severidad asociada y priorizarán. Se estudiarán los casos y se actualizará el Modelo de Amenazas y el Plan de Riesgos según sea necesario.

Esta reunión servirá de forma de colaboración, los expertos en seguridad enseñarán al Equipo de desarrollo y les ayuden a afrontar los retos de seguridad que se vayan encontrando en el proyecto.

De esta reunión saldrán las historias de seguridad que deben incluirse en el *backlog* de producto con mayor prioridad.

Los objetivos de la reunión de refinamiento del backlog de seguridad serán principalmente:

- Revisar y priorizar las tareas y actividades de aeguridad genéricas del proyecto.
- Revisar y analizar los aspectos de seguridad propios del proyecto.
- Analizar, asignar severidad y priorizar historias de usuario con implicaciones de seguridad
- Actualizar Modelo de Amenazas y Plan de Gestión de Riesgos.

Teóricamente, el *backlog* de Seguridad debería ser pequeño y el Equipo *Scrum* debería poder trabajar en las historias del *backlog* de producto de forma independiente. Las reuniones de refinamiento serán optativas por *Sprint* y se harán siempre que haya algún tema específico de seguridad. Aún así, serán obligatorias al menos una vez al mes y el *Scrum Master* será el encargado del cumplimiento de esta premisa.

- **Backlog de Producto**

El *backlog* de producto incluirá historias de usuario creadas por el *Product Owner* de la aplicación y las historias y tareas priorizadas en el *backlog* de seguridad.

El *backlog* de producto tendrá su propia reunión de Refinamiento donde se tendrán en cuenta las siguientes consideraciones:

- Usuario final malvado: a la hora de definir y refinar las historias, además de tener en cuenta lo que se quiere que se haga en el caso positivo, se tendrán en cuenta los casos negativos y cómo reaccionará la aplicación ante estas situaciones: introducción de datos incorrecto, si la información es sensible, que un usuario si permisos no pueda acceder; cross-site scripting, etc.
- Se categorizarán las historias que tengan consideraciones sobre seguridad y se les asociará la severidad apropiada. Posteriormente, se analizarán como parte de la

reunión de refinamiento y se determinará la prioridad con la que se debe abordar o si es más conveniente establecer acciones de mitigación. En caso de complejidad grande, se pasará al *backlog* de seguridad para ser analizada con los expertos.

- Se trabajará en el Modelo de Amenazas y la Gestión de Riesgos, para ello se evaluarán los riesgos conocidos y se definirá si pueden ser arreglados fácilmente o si necesitan un Plan de Mitigación. El Modelo de Amenazas y el Plan de Gestión de Riesgos irá evolucionando a lo largo de los *Sprints* según se introduzcan nuevas funcionalidades o se detecten nuevas amenazas o vulnerabilidades. Los cambios introducidos en el Modelo de Amenazas y en el Plan de Gestión de Riesgos se discutirán como parte de la reunión de refinamiento del *backlog* de seguridad.

A la hora de refinar las historias deberemos prestar especial atención a las siguientes cuestiones:

- Si la historia afecta a la entrada y/o salida de datos y a cómo esos datos son gestionados
- Si la historia incluye algún riesgo potencial o incluye nuevas amenazas
- Si la historia es suficientemente delicada como para necesitar el asesoramiento y revisión de un experto en seguridad

Estas preguntas nos permitirán ser conscientes de la seguridad y afrontar así adecuadamente el diseño e implementación de una solución adecuada, así como de sus pruebas asociadas.

2. Definición de Hecho

Se ha definido una Definición de Hecho (DoD) entre todos los implicados en el proyecto, por lo que se ha llegado a un acuerdo de que todas las historias de usuario deben cumplir los siguientes puntos para determinar que la historia ha sido completada. Hay tres tipos de criterios en la Definición de Hecho: criterios de calidad, criterios de procedimiento y criterios de seguridad:

Criterios de procedimiento	• Código subido al sistema de control de versiones. • Código mezclado a la rama *master*. • Definición del Plan de marcha atrás (*rollback*).

Criterios de Calidad	• Código construido sin errores. • Pruebas unitarias pasadas. • Pruebas de mutación pasadas. • Código revisado. • Pruebas funcionales pasadas. • Pruebas de integración pasadas. • Pruebas de despliegue automático pasadas. • Pruebas de regresión pasadas. • Pruebas de aceptación pasadas. • Monitorización de *logs* y aplicación.
Criterios de Seguridad	• Verificar marca de identificación de implicaciones de seguridad. • No existen componentes vulnerables. • Paquetes del sistema actualizados. • *Hardening* de producto y sistema. • Pruebas automáticas de incidencias de seguridad pasadas. • Revisión del Plan de Gestión de Riesgos.

3. Formación adecuada

Otro de los puntos definidos que hay que tener en cuenta es la formación del equipo en seguridad. El equipo debe estar formado y actualizado sobre vulnerabilidades de seguridad, herramientas para detectar vulnerabilidades o brechas de seguridad y cómo utilizarlas. Además, en casos de proyectos que incluyan características específicas de seguridad, el equipo deberá formarse en esa funcionalidad o herramienta de forma adecuada y planificada.

Parte de esta formación será realizada como parte de las reuniones de refinamiento del backlog de seguridad.

4. Análisis y diseño

Es necesario diseñar el software teniendo en cuentas las políticas de seguridad y el cumplimiento de los principios básicos de seguridad como son:

- Seguridad por defecto
- Denegación por defecto
- A prueba de fallos
- Mínimo privilegios.

Modelado de Amenazas y Plan de Gestión de Riesgos

Durante esta fase de Análisis y diseño, así como durante el Refinamiento, se tendrán en cuenta si se debe actualizar el Modelo de Amenazas y/o el Plan de Gestión de Riesgos. Sólo la funcionalidad que se incluye en el *Sprint* actual debe analizarse e incluirse en el Modelo de Amenazas o el Plan de Gestión de Riesgos si así se considera necesario.

5. Desarrollo y pruebas

Las fases de desarrollo y pruebas son muy importantes para la seguridad. Para que se pueda llevar a cabo un desarrollo ágil y seguro necesitamos centrarnos en la automatización de ciertas tareas:

- Herramientas de Análisis Estático en el IDE de desarrollo: Esto permite a los desarrolladores detectar los bugs de seguridad en su propio código, minimizando el tener que re-implementar código a posteriori y favoreciendo el aprendizaje.
- Comprobaciones de integraciones o dependencias externas: Existen herramientas que se pueden incorporar localmente a la hora de construir o empaquetar la aplicación que consultan repositorios públicos de vulnerabilidades (CVE, etc.) para comprobar si alguna de las librerías que usamos incluye alguna vulnerabilidad.
- Integrar las comprobaciones anteriores en el entorno de integración continua (CI/CD). De esta forma si alguien del equipo no está haciendo las comprobaciones necesarias en local, habrá un error que no permitirá continuar con el empaquetado y despliegue de la aplicación.
- Integrar herramientas de pruebas dinámicas en el entorno de CI/CD cuando la aplicación ya está desplegada en los entornos de pruebas: esto permitirá hacer pruebas sobre el software ya desplegado como si fuera un entorno real y las pruebas

configuradas pueden simular comportamientos reales. Estas pruebas se irán ampliando con cada historia o funcionalidad añadida.

6. Despliegue

En la fase de despliegue también hay una serie de automatizaciones que nos pueden ayudar a continuar integrando la seguridad en el software desarrollado.

- Actualizando el software de nuestros servidores y terceros, usando las últimas versiones o integrando parches de seguridad.
- Incluyendo la monitorización o alertas necesarias en nuestra infraestructura.

Scrum Seguro: Eventos y Artefactos

Teniendo en cuenta los diferentes eventos de *Scrum*, el siguiente diagrama nos ayuda a ver cómo se aplica la seguridad en cada evento y qué tipo de tareas de seguridad y artefactos incluye este enfoque ágil y seguro.

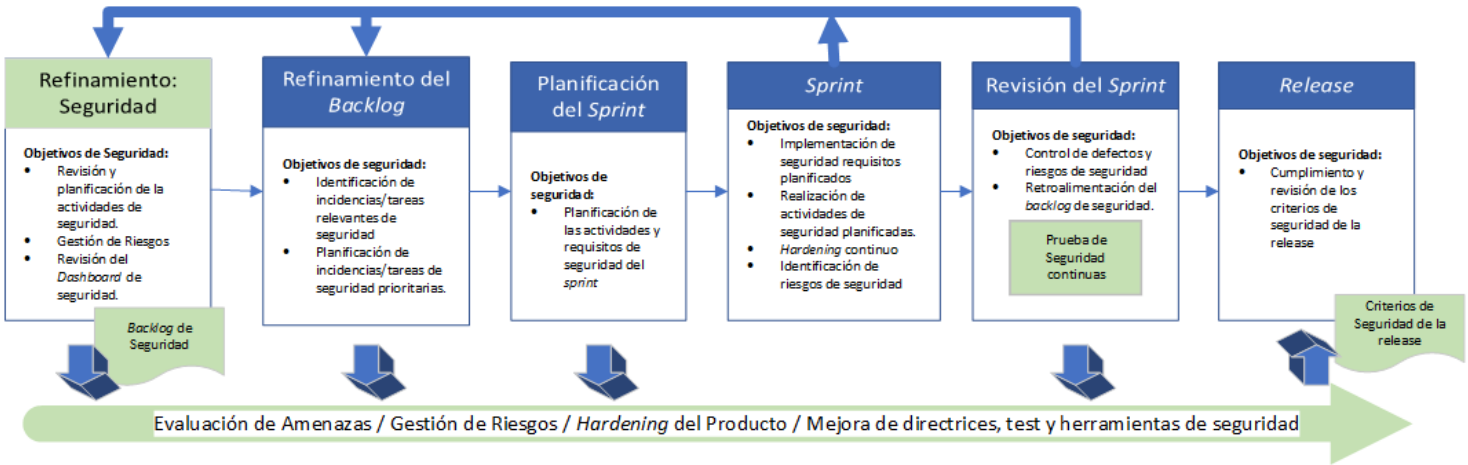

Figure 5. Propuesta de Scrum Seguro

Al ser una aproximación a *Scrum* nos centraremos en la definición de historias de usuario y en la inclusión de la seguridad en ellas, para ello:

1. Refinamiento: A la hora de refinar las historias de usuario se tendrá en cuenta la seguridad, así como la gestión de riesgos. El refinamiento tendrá lugar en cada

iteración y se revisarán y adaptarán las historias de usuario según proceda. El refinamiento se hará en dos *backlogs*:

a. de seguridad: con tareas y actividades genéricas del proyecto, además de implicaciones de seguridad más complejas especificas del proyecto. Y revisión del Modelo de Amenazas y el Plan de Gestión de Riesgos.
b. de producto: historias de usuario específicas de la aplicación. Cuando se haga el refinamiento del *backlog* de producto, se deberá tener en cuenta las consideraciones de seguridad necesarias que irán enriqueciendo las historias.

El Backlog de Seguridad incluirá también tareas en el backlog de producto por prioridad, de forma en los *sprints* podrán incluir además de historias de usuario, tareas de seguridad.

2. Planificación del Sprint: En la planificación del *Sprint* se decidirá qué entra en el siguiente *Sprint*, teniendo en cuenta los objetivos de seguridad que están definidos en el proyecto y en el equipo.
3. Sprint: Los *sprints* se llevarán a cabo teniendo en cuenta los objetivos y directrices de seguridad marcados por el equipo. Cada una de las tareas e historias de usuario considerará los criterios de aceptación necesarios para completarlas. Cada *Sprint* incluirá:
    a. Implementación de los requisitos de seguridad planificados: la implementación también incluirá el desarrollo de pruebas automatizadas (pruebas unitarias, pruebas funcionales y pruebas de integración).
    b. Realización de actividades de seguridad planificadas.
    c. Identificación de riesgos de seguridad.
    d. Revisiones de código: automáticas y por compañeros
    e. Integración continua y despliegue continuo (CI/CD): escaneos de seguridad automáticos, pruebas automáticas, pruebas de seguridad usando herramientas especializadas.
4. Revisión del *sprint*: Cuando se revise cómo fue el *sprint* se comprobará que se hayan cumplido los objetivos de seguridad y en caso de defectos o casos complejos, se añadirán al *backlog* de seguridad para su revisión.
5. Retrospectiva: el equipo evaluará cómo ha ido el *sprint*, si considera que se podría mejorar de alguna forma y si es necesario aplicar cambios sobre el proceso.

6. *Release*: dependiendo del proceso el despliegue del código en producción será cada *sprint* o cada varios *sprints*. Antes de pasar a producción, el código nuevo deberá pasar una serie de pruebas para comprobar que todo funciona perfectamente y que se cumple los criterios de seguridad de la release. Esta fase incluye:
    a. Automatización del empaquetado y escaneo de librerías de terceros.
    b. Entornos de integración y pre-explotación
        i. Pruebas de regresión: que incluirá pruebas de seguridad.
        ii. Pruebas de intrusión
        iii. *Checklist* de seguridad: esta *checklist* tendrá en cuenta los objetivos de seguridad del equipo y del proyecto, así como los criterios de seguridad definidos por el equipo.
        iv. *Hardening* del producto y sistema

El siguiente diagrama permite ver observar cómo sería el proceso de desarrollo de software de una iteración, es decir, desde que se tienen los requisitos que van en el sprint, hasta que se despliega en producción:

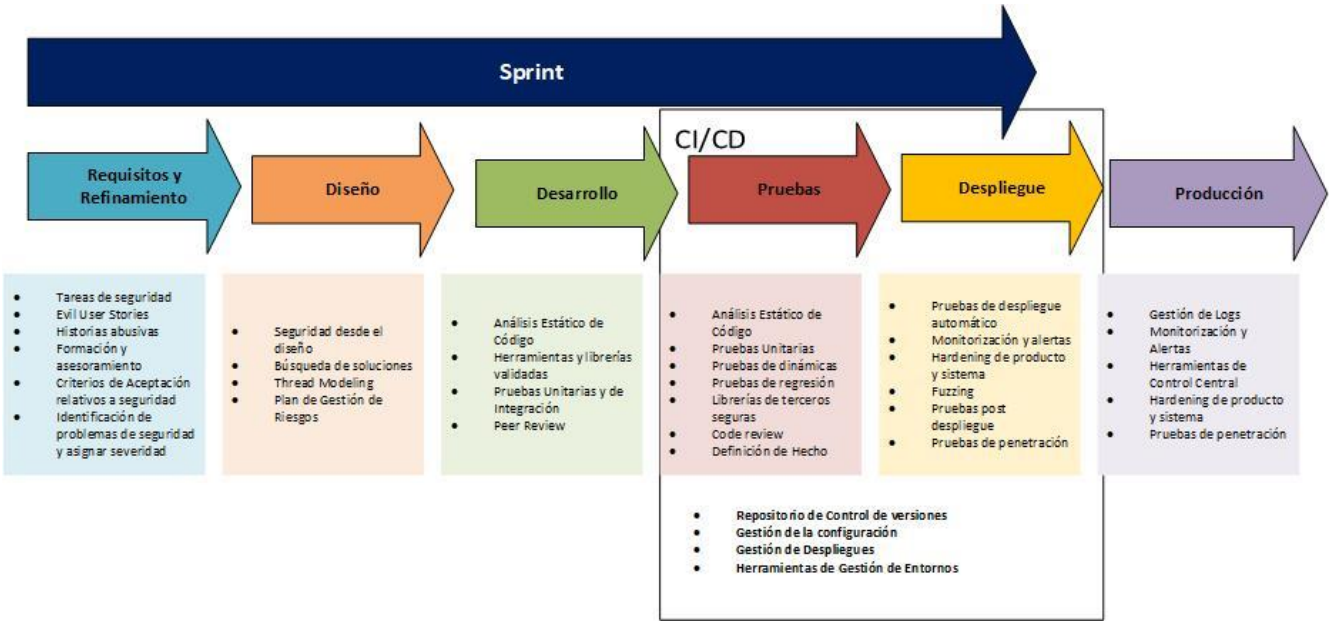

Figure 6. Security in the agile lifecycle

Fase	Tareas y consideraciones
Requisitos y refinamiento	• Tareas de seguridad • Evil User Stories • Historias Abusivas

	• Formación y asesoramiento • Criterios de aceptación relativos a seguridad • Identificación de problemas de seguridad y asignación de severidad.
Diseño	• Seguridad desde el diseño • Búsqueda de soluciones • Thread Modelling • Plan de Gestión de Riesgos
Desarrollo	• Análisis Estático de Código • Herramientas y librerías validadas • Pruebas Unitarias y de Integración • Peer Review
Pruebas	• Análisis Estático de Código • Pruebas Unitarias • Pruebas Dinámicas • Pruebas de Regresión • Librerías de terceros seguras • Code Review • Definición de hecho
Despliegue	• Pruebas de Despliegue Automático • Monitorización y Alertas • Hardening de producto y sistema • Fuzzing • Pruebas post despliegue • Pruebas de penetración
Producción	• Gestón de Logs • Monitorización y Alertas • Herramientas de Control Central • Hardening de Servidor

	• Pruebas de penetración

Es necesario tener en cuenta que, para poder llevar a cabo esta metodología de forma efectiva y que una metodología de desarrollo Ágil siga siento rápida y flexible, es necesario incluir prácticas de DevOps en el proceso de desarrollo. Esto permitirá una mayor colaboración con los compañeros de sistemas, una mayor automatización de las tareas de integración continua y despliegue, así como una mejor concienciación sobre la seguridad.

Conclusiones

El mercado cambiante y global requiere de metodologías de desarrollo rápidas y flexibles, pero que a su vez construyan software seguro que sepa mantener la información de los usuarios a salvo y asegurar la disponibilidad de los sistemas.

Se han estudiados varias iniciativas, normas, estándares y herramientas que velan por ello y presentan propuestas para crear software seguro. De este análisis se ha deducido que:

- Para desarrollar software seguro es clave la formación y la concienciación en términos de seguridad.
- Es necesario incluir la seguridad desde el comienzo del proyecto y a lo largo de toda la vida de este.
- La evolución del desarrollo de software va orientada a la colaboración entre Operaciones y Desarrollo, permitiendo así crear un mejor software que incluya mayor automatización.
- Incluir pruebas de seguridad y herramientas de automatización de pruebas es necesario en entornos Ágiles.
- Cuando antes se detecten amenazas, vulnerabilidades o fallos más fácil será corregirlos y menor será el coste asociado y el impacto.
- El Modelado de Amenazas y el Plan de Gestión de Riesgos no debe dejarse de lado en las metodologías ágiles y debe estar vivo y actualizarse cuando sea necesario con cada iteración.
- La mejor metodología es aquella que mejor se adapta al contexto, el éxito y los buenos resultados que pueda tener una metodología, depende del producto a fabricar y del equipo. La continua revisión de procesos y adaptación mejorará los resultados.

Líneas futuras

Este proyecto ha cubierto la parte teórica sobre buenas prácticas y cómo integrar buenas prácticas dentro de la metodología Ágil. Como trabajos futuros se podría continuar en las siguientes líneas de trabajo:

- Implementar esta propuesta a un entorno de desarrollo y ver los resultados. El implementar esta propuesta en un entorno de desarrollo nos permitiría adaptar la metodología propuesta a las necesidades del proyecto.
- Análisis más en profundidad del mundo de DevOps y SecDevOps, ver hacía dónde va esta filosofía de trabajo y hacer un estudio de herramientas y asignarlas con las distintas fases del Ciclo de Vida de Desarrollo Software, estudiando cómo podría integrarse en diferentes metodologías ágiles.
- Estudio cuantitativo de los resultados obtenidos usando esta propuesta de metodología *Scrum* Segura en diferentes proyectos: comparar tiempos, costes, métricas de seguridad, etc. con otras metodologías de desarrollo software para así poder tener datos tangibles sobre las mejoras introducidas en comparación con otras.
- Gestión de riesgos y Modelado de amenazas en entornos ágiles.

Figuras

Bibliografía

Uribe, E., 2007. *Del manifiesto ágil sus valores y principios.* Pereira: Scientia Et Technica.
Project Management Institute, 2017. *A Guide to the Project Management Body of Knowledge (PMBOK Guide).* 6th ed ed. Newtown Square, Pennsylvania: Project Management Institute.
Beck, K. e. a., 2001. *Manifesto for Agile Software Development.* [Online]
Available at: http://agilemanifesto.org/
[Accessed June 2021].
Figueroa, V., 2016. *Secure Software Development Life Cycle.* [En línea]
Available at: https://www.owasp.org/images/9/9d/OWASP-LATAMTour-Patagonia-2016-rvfigueroa.pdf
[Último acceso: June 2021].
Amber, S. & Associates, 2013. *IT Project Success Rates Survey Results.* [En línea]
Available at: http://www.ambysoft.com/survey/success2013.html
[Último acceso: June 2021].
Mersino, A., 2018. *Vitality Chicago.* [En línea]
Available at: https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects
[Último acceso: June 2021].
Djurovic, A., 2020. *Go Remotely.* [En línea]
Available at: https://goremotely.net/blog/agile-adoption/
[Último acceso: June 2021].
Beck, K., 2001. *Agile manifesto.* [En línea]
Available at: https://agilemanifesto.org/iso/es/manifesto.html
[Último acceso: June 2021].
DSMD Consortium, 2015. *What is DSMD? Retrieved from DSDM web site:.* [En línea]
Available at: http://www.dsdm.org/content/what-dsdm
[Último acceso: June 2021].
Schwaber, K. & Sutherland, J., 2020. La Guía Scrum. *Scrum Guide Spanish European.*
Scrum.org, s.f. *Scrum Glossary.* [En línea]
Available at: https://www.scrum.org/resources/scrum-glossary
[Último acceso: June 2021].
Alotaibi, M., 2016. *EThOs.* [En línea]
Available at: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.693486
[Último acceso: June 2021].
Sullivan, B., 2019. *Agile SDL-Streamline Security Practices For Agile Development.* [En línea]
Available at: https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/november/agile-sdl-streamline-security-practices-for-agile-development
[Último acceso: June 2021].
Chandra, P., 2013. *Software Assurance Madurity Model.* San Francisco, OWASP.
McGraw, G., 2006. *Software Security: Building Security.* s.l.:Pearson Education.
CWE, 2021. *Common Weakness Enumeration.* [En línea]
Available at: https://cwe.mitre.org/
[Último acceso: July 2021].
CERT Coding Standards, 2021. *SEI CERT Coding Standards.* [En línea]
Available at:
https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
[Último acceso: July 2021].

CVE, 2021. *CVE.* [En línea]
Available at: https://cve.mitre.org/index.html
[Último acceso: July 2021].
NVD, 2021. *National Vulnerability Database.* [En línea]
Available at: https://nvd.nist.gov/
[Último acceso: July 2021].
ENS, 2015. *Esquema Nacional de Seguridad.* [En línea]
Available at:
https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/pae_Seguridad_Inicio/pae_Esquema_Nacional_de_Seguridad.html#.YNYSIpMzaX0
[Último acceso: June 2021].
Assal, H. & Chiasson, S., 2018. Security in the Software Development Lifecycle. *Usenix: The advanced computing systemss association,* August, pp. 281-295.
Green, M. & Smith, M., 2016. Developers are not the enemy! The need for security apis. *IEEE Security Privacy,* Volumen 14, pp. 40-46.
Levison, M., 2019. *Agile Pain Relief - Antipattern Hardening Sprint.* [En línea]
Available at: https://agilepainrelief.com/blog/antipattern-hardening-sprint.html
[Último acceso: June 2021].
Wikipedia, s.f. *Fuzzing.* [En línea]
Available at: https://es.wikipedia.org/wiki/Fuzzing)
[Último acceso: June 2021].
Kissel, R., 2013. *Glossary of Key Information Security Terms. Retrieved from National Institude or Standards Tecnology.* [En línea]
Available at: http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf
[Último acceso: July 2021].
Nguyen, T., 2015. *INSF-6840 - Integrating Security into Agile Software Development Methods.* [En línea]
Available at:
https://www.umsl.edu/~sauterv/analysis/F2015/Integrating%20Security%20into%20Agile%20methodologies.html.htm
[Último acceso: July 2021].
Securosis, 2013. *Secure Agile Development.* [En línea]
Available at:
https://securosis.com/assets/library/reports/SecureAgileDevelopment_Nov2014_FINAL.pdf
[Último acceso: July 2021].
Schwaber, K. & Sutherland, J., 2013. *The Definitive Guide to Scrum: The rules of the Game..* [En línea]
Available at: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100
[Último acceso: July 2021].
Mead, N., Viswanathan, V. & Zhan, J., 2008. Incorporating Security Requirements Engineering into Standard Lifecycle Processes. *International Journal of Security and its Applications,* Volumen 2(4), pp. 67-79.
Chinnaraju, A., 2017. *Agile Security Manifesto.* [En línea]
Available at: https://es.slideshare.net/ArunChinnaraju/agile-security-manifesto
[Último acceso: July 2021].
Pohl, C. & Hof, H., 2015. Secure Scrum: development of Secure Software with Scrum. *arXiv.*
Encinar, J., 2018. *Ciclo de vida de desarrollo de software (S-SDLC).* [En línea]
Available at: https://www.boomernix.com/2018/03/ciclo-de-vida-de-desarrollo-se-

software.html
[Último acceso: July 2021].
OWASP, 2021. *OWASP.* [En línea]
Available at: https://owasp.org/
[Último acceso: July 2021].
AEPD, 2019. *Guía de Privacidad desde el Diseño.* [En línea]
Available at: https://www.aepd.es/sites/default/files/2019-11/guia-privacidad-desde-diseno.pdf
[Último acceso: July 2021].
Lohar, P., 2018. *DevOps & Agile Faster Releases.* [En línea]
Available at: https://medium.com/@prasanna.lohar/devops-agile-better-builds-faster-releases-8fa5c4943d26
[Último acceso: June 2021].
López-Rodríguez, S. A., 2020. Metodologías de desarrollo de software seguro con propiedades ágiles. *Polo del conocimiento,* 5(10), pp. 1027-1046.
Rindell, K. y otros, 2021. Security in agile development: A practitioner survey. *ScienceDirect.*
Heusser, M., 2021. *43 free and open-source tools that put te Ops in DevOps.* [En línea]
Available at: https://techbeacon.com/devops/43-free-open-source-tools-put-ops-devops
[Último acceso: June 2021].
Belmer, C., 2019. *Integrating Security with Agile Development.* [En línea]
Available at: https://nullsweep.com/integrating-security-with-agile-development/
[Último acceso: July 2021].