TESIS DOCTORAL

Implementación de un Modelo de Desarrollo de Software Seguro

JOSÉ CARLOS SANCHO NÚÑEZ

PROGRAMA DE DOCTORADO EN TECNOLOGÍAS INFORMÁTICAS (TIN)

2020

UNIVERSIDAD DE EXTREMADURA

TESIS DOCTORAL

Implementación de un Modelo de Desarrollo de Software Seguro

JOSÉ CARLOS SANCHO NÚÑEZ

PROGRAMA DE DOCTORADO EN TECNOLOGÍAS INFORMÁTICAS (TIN)

2020

Con la conformidad de los directores:

Fdo. Dr. D. Andrés Caro Lindo _____

Fdo. Dr. D. Pablo García Rodríguez _____

La conformidad del director/es de la tesis consta en el original en papel de esta Tesis.

Tesis Doctoral presentada por el doctorando José Carlos Sancho Núñez en el Departamento de Ingeniería de Sistemas Informáticos y Telemáticos de la Universidad de Extremadura para la obtención del título de Doctor por la Universidad de Extremadura en el Programa de Doctorado en Tecnologías Informáticas (TIN).

Finalizada en Cáceres el 30 de octubre de 2020.

Título:

Implementación de un Modelo de Desarrollo de Software Seguro

Doctorando:

José Carlos Sancho Núñez (jcsanchon@unex.es)
Departamento de Ingeniería de Sistemas Informáticos y Telemáticos
Escuela Politécnica de Cáceres
Universidad de Extremadura

Directores:

Andrés Caro Lindo (andresc@unex.es)
Pablo García Rodríguez (pablogr@unex.es)

Esta Tesis Doctoral ha sido realizada dentro del grupo de investigación GIM (Grupo de Ingeniería de Medios, grupo TIC012 del catálogo de grupos de investigación reconocidos por el Sistema Extremeño de Ciencia, Tecnología e Innovación) como parte de las actividades englobadas en la Cátedra de Patrocinio "Seguridad y Auditoría de Sistemas Software" entre la Universidad de Extremadura y la empresa Viewnext S.A.

Agradecimientos institucionales

Me gustaría agradecer a las siguientes instituciones su apoyo durante la realización de esta tesis:

- Al Grupo de Investigación de Ingeniería de Medios (GIM) el cual ha permitido mi desarrollo investigador con un apoyo incondicional.

- A la empresa Viewnext S.A. por su apuesta en la investigación con la Universidad de Extremadura a través de la Catedra de Patrocinio "Seguridad y Auditoría de Sistemas Software". Este convenio firmado en el año 2015 ha posibilitado trabajar en un escenario con datos reales que garantizan la transferencia directa de conocimiento desde un centro de investigación a una empresa como Viewnext S.A.

- Al Grupo de Investigación de Análisis, Seguridad y Sistemas (GASS) de la Universidad Complutense de Madrid donde tuve la oportunidad de pasar un pequeño tiempo de estancia y aprender de un equipo de personas que aprecian la investigación como forma de vida.

- Al Departamento de Ingeniería de Sistemas Informáticos y Telemáticos el cual, desde mi inicio como docente, me ha brindado su apoyo y facilidades en todos los recursos que están a su disposición.

- A la Escuela Politécnica, el lugar donde, sin duda, he pasado gran parte de mi tiempo. Centro al que llegué en el año 2005 como estudiante, en el que crecí profesionalmente como Personal de Administración y Servicio (2011-2017) y en el que desde el año 2018 tengo la oportunidad de trasmitir conocimiento como Personal Docente Investigador.

Agradecimientos personales

Siempre pensé que el apartado de agradecimientos personales eran el verdadero resumen de una Tesis Doctoral, el lugar donde se referencia lo que realmente ha sido importante en el proceso y que toma el mismo valor, incluso, que las conclusiones de la investigación.

Mis agradecimientos toman la estructura de "*El viaje del héroe*", un patrón descrito por el mitógrafo Joseph Campbell en su libro "*El héroe de las mil caras"*. Este patrón es la estructura que siguen multitud de historias y que a día de hoy está presente en cine, literatura, videojuegos y en esta Tesis Doctoral.

La historia comienza cuando el héroe está en la zona de confort de su mundo ordinario y llega la llamada a la aventura. Este momento, coincide con el año 2015 cuando trabajando fuera de la Universidad de Extremadura, un antiguo profesor de la titulación de Ingeniería Informática, que a lo postre sería trascendental, me cuenta un proyecto y la posibilidad de hacer una Tesis Doctoral.

En este momento, resulta decisivo el papel de un mentor que reta al héroe a adentrarse en la aventura y le guía en la misma. Por ello, agradezco infinitamente a mis mentores Andrés Caro y Pablo García el trato que me han dado estos años. De quienes he aprendido mucho y espero heredar su profesionalidad, capacidad de trabajo y calidad humana. Gracias por toda vuestra ayuda y consejos.

En la siguiente fase, el héroe cruza un umbral y se adentra en el mundo extraordinario, una zona desconocida (Viewnext) que tendrá que explorar. Momento de agradecer a José Andrés Félix de Sande y Ángel Quesada, miembros de la empresa Viewnext S.A. el apoyo que me han brindado consiguiendo que avanzase y superarse obstáculos cuando algo se complicaba.

En un momento dado de la historia, el héroe cae, es el instante en que está más lejos de alcanzar su propósito y siente que no lo va a lograr. Es la oportunidad de agradecer a Gemma Villegas, mi compañera de viaje, la persona que más apoyo personal, cariño y paciencia me ha dado cuando más lo he necesitado. Gracias por mantener mi equilibro y hacer que la etapa final fuese fácil, espero poder recompensarte con más tiempo y más cariño.

Entonces, el héroe se levanta de su caída y alcanza un estado superior, más conocimiento, más decisión, más valor, más compasión… siendo el momento de la batalla final en la que pone en juego todo lo aprendido en el camino. El camino de la humildad y la constancia, valores de una familia de colonos agricultores que han sabido aprovechar los pocos recursos de los que disponían para dar la mejor educación posible a sus hijos. Doy las gracias a mis padres Antonio y Paula, y a mi hermana Ana, por enseñarme que lo importante no es lo que tenemos si no con quién lo compartimos.

Por último, el personaje regresa a casa pero transformado. Desde el inicio de esta Tesis Doctoral, que parece que fue hace una eternidad, he conocido sitios nuevos en cada congreso, héroes y villanos (por los revisores), he luchado duro hasta conseguir publicar, he descuidado amigos y he conocido a otros, he sufrido y superado miedos, y ya no soy el mismo. Quiero terminar agradeciendo a mis amigos y amigas más cercanos el tiempo que han compartido conmigo: Laura M., Álvaro, Barky, Rosa, Rebeca, Serna, Lorena, Lupe, Violeta, Cris, Olga y Eli. También, a varios estudiantes que pusieron con su Trabajo Fin de Estudios un granito de arena en la batalla final como M. Luz, Lucio y José Méndez. A otros compañeros de la UEx como Chema, Roberto, Iván, Antonio, José Ceballos, Jesús Torrecilla, Ramón y Maribel. A todas las personas que han pasado por el laboratorio del Grupo de Ingeniería de Medios (GIM) y con los que he compartido momentos inolvidables: Manolo, Mar, Aurora, Miguel, Dani C., JuanPe, Irene, Alberto, Óscar, Javi, Dani M. y Manuel. Y por último, a todos los estudiantes que he tenido la oportunidad de trasmitir conocimiento como docente y de los que aprendo cada día.

Después de este viaje, el héroe puede haber alcanzado o no su propósito, pero es otro porque aprendió algo en el camino.

"Si he visto más lejos es porque estoy sentado sobre los hombros de gigantes"
Isaac Newton

Gracias a todos y todas, aquí mi resultado.

X

Índice General

Índice de Figuras

Índice de Tablas

Lista de Acrónimos

AMS	*Application Management Services*
ASVS	*Application Security Verification Standard*
BSIMM	*Building Security In Maturity Model Framework*
CCN	*Centro Criptológico Nacional*
CERT	*Computer Emergency Response Team*
CLASP	*Comprehensive Lightweight Application Security Process*
CMMI	*Modelo de Madurez y Capacidad Integrado*
CSIRT	*Computer Security Incident Response Team*
CVSS	*Common Vulnerability Score System*
CWE	*Common Weakness Enumeration*
FIPS	*Federal Information Processing Standard*
HTTPS	*Hypertext Transfer Protocol Secure*
IBM	*International Business Machines*
IEC	*International Electrotechnical Commission*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
NIST	*National Institute of Standards and Technology*
OSSA	*Oracle Software Security Assurance*
OWASP	*Open Web Application Security Project*

RTC	*Rational Team Concert*
SAMM	*Software Assurance Maturity Model*
SANS	*SysAdmin Audit, Networking and Security*
SDL	*Software Development Lifecycle*
SDLC	*Software Development Life Cycle*
SEI	*Software Engineering Institute*
SMB	*Server Message Block*
SOA	*Service Oriented Architecture*
SSDLC	*Secure Software Development Life Cycle*
SSL	*Secure Sockets Layer*
STRIDE	*Spoofing, Tampering, Repudiation, Information disclosure, Denial of services, Elevation of privileges*
TCP	*Transmission Control Protocol*
TSP	*Team Software Process*
TSP-Secure	*Team Software Process Secure*
URL	*Uniform Resource Locator*
VDI	*Virtual Desktop Infrastructure*
WASC	*Web Application Security Consortium*
XSS	*Cross-site scripting*
ZAP	*Zed Attack Proxy*

Resumen

La investigación realizada en esta Tesis Doctoral se engloba dentro de las actividades desarrolladas en el ámbito de la Cátedra de Patrocinio "Seguridad y Auditoría de Sistemas Software" entre la Universidad la Universidad de Extremadura y la empresa Viewnext S.A.

La necesidad producida por las graves consecuencias que sufren los sistemas informáticos debido al incremento del número de ciberataques con éxito en la última década ha generado un cambio de paradigma con respecto a la seguridad en los procesos de desarrollo del software. La aparición de vulnerabilidades por la falta de controles de seguridad es una de las causas por las que se demandan nuevos marcos de trabajo que produzcan software seguro de forma predeterminada.

Esta Tesis Doctoral presenta una revisión de los modelos de software seguro más utilizados o conocidos y propone un nuevo modelo de desarrollo – modelo Viewnext-UEx – que incorpora prácticas de seguridad de forma preventiva y sistemática en todas las fases del proceso de ciclo de vida del software. El propósito de este nuevo modelo es anticipar la detección de vulnerabilidades aplicando la seguridad desde las fases más tempranas, a la vez que se optimizan los procesos de construcción del software. Para ello, se realiza un caso de estudio donde los datos corresponden a la empresa de servicios tecnológicos Viewnext S.A.

El caso de estudio plantea una novedosa comparación de resultados en dos escenarios de desarrollo reales. El proyecto de software elegido para el experimento pertenece a una compañía del sector eléctrico, considerado crítico en el ámbito de la seguridad. Aunque los escenarios siguen metodologías distintas para producir el software, se ejecutan en el mismo proyecto de software y por el mismo equipo de trabajo.

El primer escenario mantiene la orientación de seguridad tradicional donde, de manera reactiva, se detectan las vulnerabilidades al finalizar el software. El segundo escenario, cuyo enfoque es preventivo, implanta el modelo Viewnext-UEx propuesto en esta tesis, de modo

que en el desarrollo se aplica la seguridad por defecto en todas las fases del ciclo de vida del software.

Los resultados obtenidos indican que en el escenario donde se aplica el modelo Viewnext-UEx se produce una reducción considerable del número vulnerabilidades, de su criticidad y del impacto temporal en la resolución de los fallos de seguridad, frente al escenario reactivo.

Por tanto, se concluye que la falta de controles de seguridad durante el proceso de construcción del software incrementa el número de vulnerabilidades y produce mayor dedicación en la fase de corrección. Por el contrario, la aplicación del modelo Viewnext-UEx anticipa la detección de vulnerabilidades y minimiza su tiempo de corrección.

Bloque I

Contexto de la investigación

"El propósito es el punto de partida de todo logro."

W. Clement Stone

Capítulo 1

1. Introducción

La investigación que se presenta propone el Modelo de Desarrollo de Software Viewnext-UEx, que tiene como objetivo reducir vulnerabilidades en la creación del software, optimizar los tiempos de implementación del desarrollo y construir software más fiable y seguro.

Esta Tesis Doctoral se enmarca y desarrolla dentro de la Cátedra de Patrocinio sobre "Seguridad y Auditoría de Sistemas Software", firmada en el año 2015 y vigente en la actualidad (año 2020), entre investigadores del grupo de Ingeniería de Medios (GIM) de la Universidad de Extremadura (UEx) y Viewnext S.A., empresa de Servicios de Tecnología de la Información del Grupo IBM España. Este acuerdo tiene como principal finalidad la transferencia y aplicación del conocimiento generado por la UEx a través de investigaciones sobre nuevas metodologías de desarrollo de software seguro en proyectos reales, implementados en la empresa Viewnext S.A. De este modo, queda garantizada en esta tesis la transferencia directa de conocimiento entre universidad y empresa y la aplicación de la presente investigación a un contexto real, considerándose así uno de sus puntos fuertes.

Estructurada en 7 capítulos, directamente relacionados, cada uno de ellos proporciona conclusiones parciales, que en su conjunto, conforman los resultados de investigación esperados.

En la primera parte de la tesis se hace una revisión de los diferentes conceptos técnicos relacionados con la seguridad del software y se identifican los modelos o marcos de trabajo mejor considerados dada su especialización en la seguridad en el desarrollo de software.

Estos modelos, en su mayoría propietarios, integran por defecto la seguridad de forma planificada durante todo el proceso o fases del ciclo de vida del desarrollo del software. Esta parte preliminar permite conocer los límites investigados en este ámbito y ofrece las primeras conclusiones del "Estudio del Arte", que se presentan mediante una comparativa en profundidad de los modelos de desarrollo seguro analizados.

Posteriormente, se presenta el modelo Viewnext-UEx, identificando y proponiendo nuevas actividades de seguridad con el objetivo de suplir las carencias detectadas en los modelos estudiados. Las actividades seleccionadas se categorizan en áreas de desarrollo y se ejecutan de forma sistemática en las fases del ciclo de vida del software.

El nuevo modelo debe adaptarse a las necesidades de cualquier proyecto de software. Por lo tanto, se aporta además una metodología de implantación empresarial para facilitar la integración de las prácticas y controles de seguridad destinados a mejorar la seguridad de los desarrollos.

De esta forma, los avances presentados tomarán dos enfoques fundamentales: el primero de ellos, contempla la evolución del nuevo Modelo de Desarrollo Seguro con el objetivo de mejorar los procesos de seguridad que permitan conseguir software más fiable; y el segundo, engloba la implantación empresarial encargada de medir los costes y nivel de seguridad tras aplicar los procesos de seguridad definidos en el modelo propuesto.

Con el fin de obtener resultados concluyentes, esta Tesis Doctoral se enmarca en un proyecto de software real, que implementa un módulo siguiendo una metodología estándar, para poder comparar sus costes temporales y nivel de seguridad con los obtenidos en otro módulo desarrollado siguiendo el modelo Viewnext-UEx .

Los resultados obtenidos tras la evaluación de ambos módulos, desarrollados por el mismo equipo de trabajo pero empleando distintas metodologías, proporcionan unas conclusiones determinantes y permiten valorar el recorrido de la investigación.

Si bien no debe olvidarse el reto de que el software que hoy se considera seguro no tiene por qué serlo mañana, lo que plantea describir en esta Tesis Doctoral las líneas futuras de trabajo que, a corto plazo, se esperan emprender.

1.1 Motivación y delimitación del tema

El número de ciberataques se ha ido incrementando considerablemente cada año. Así lo indican los informes de ciberamenazas y tendencias que recogen los ciberincidentes gestionados por el Centro Criptológico Nacional (CCN-CERT) de los años 2016 (20.940), 2017 (26.500) y 2018 (38.029) [1]–[3]. Un ataque cibernético es la acción ilícita y malintencionada realizada contra los sistemas informáticos para consultar, extraer, cifrar o destruir información privada, con la finalidad de pedir a cambio una prestación económica o de inhabilitar los servicios que ofrecen dichos sistemas informáticos.

Los fallos de seguridad explotados por los ataques cibernéticos genera un gran impacto en las empresas [4]. La alta sofisticación de los ataques provoca consecuencias alarmantes, como grandes pérdidas económicas o retrasos temporales en la ejecución y culminación de proyectos. Sumado al deterioro en la imagen corporativa de las empresas e incluso incumplimientos legislativos de la protección de datos de clientes. Todo ello ha hecho que la seguridad informática se haya convertido en un factor determinante para la supervivencia y reputación de las empresas.

El aumento de ciberataques con éxito y la aparición creciente de vulnerabilidades ha puesto el foco del problema en el proceso de desarrollo del software. Tradicionalmente, las empresas de desarrollo han dado escasa importancia a la seguridad del software. Además, han enfocado la implementación y el diseño del software únicamente a la funcionalidad del mismo, sin tener en cuenta prácticas o actividades de seguridad durante las fases de construcción del software. Pese a que en los últimos años este paradigma ha ido cambiando, todavía las empresas desarrolladoras esperan hasta la fase de pruebas para identificar y solventar vulnerabilidades, es decir, trabajan bajo un modelo reactivo. Sin embargo, Sodiya y otros, clarifican en [5] los problemas de conseguir productos de software seguros.

Se demuestra que retrasar la solución de fallos de seguridad hasta la etapa de pruebas tiene el inconveniente de aumentar exponencialmente el coste de esta fase. Además, podría ocasionar un rediseño y excesivos cambios en la implementación del software. Así, el Instituto Nacional de Estándares y Tecnología (NIST) cuantifica en [6] que es 30 veces más costoso solucionar una vulnerabilidad durante la fase de post-producción que durante la etapa de diseño, la toma de requisitos y la definición de la arquitectura. La Figura 1.1 refleja el coste incremental al resolver vulnerabilidades en fases finales del desarrollo del software con respecto a su detección en fases más tempranas.

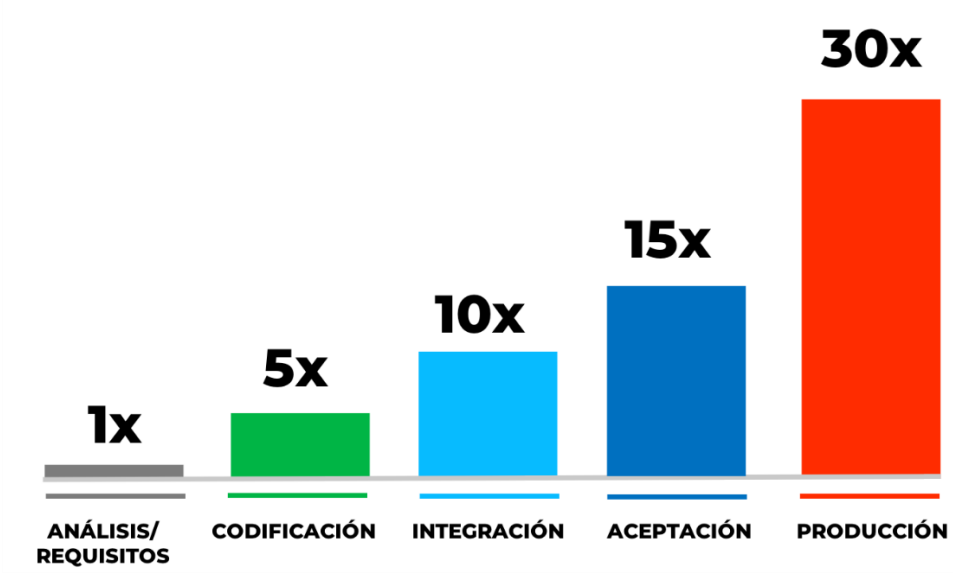


Figura 1.1. Coste de resolución de fallos de seguridad en función de la fase del ciclo de vida donde se detecta. *Fuente: NIST.*

De esta forma, la necesidad de ajustar los tiempos de desarrollo mediante la reducción del riesgo de exposición y de los gastos extras generados, ha obligado a las factorías de software a introducir procesos de seguridad durante las primeras fases del desarrollo del software.

Planteada la necesidad de considerar la seguridad en todas las tareas de las fases del ciclo de vida del software, esta investigación pretende evolucionar el paradigma del desarrollo de software *reactivo* conforme a la seguridad y proponer una nueva metodología de desarrollo seguro de software *preventiva* y particularizada en la empresa Viewnext S.A. Esta nueva metodología debe incorporar prácticas de seguridad planificadas en el ciclo de vida de desarrollo y auditoría del software. También debe recoger el proceso de implantación empresarial del nuevo modelo propuesto, aplicándolo experimentalmente en proyectos de software reales.

Esta Tesis Doctoral busca que la seguridad del software no afecte negativamente a las empresas desarrolladoras, de forma que se optimicen los tiempos de implementación y se construya software con mayor seguridad que el realizado tradicionalmente siguiente los procesos reactivos. Este enfoque de mayor seguridad debe evitar posibles pérdidas de confianza por parte de los clientes y, como consecuencia, impedir posibles disminuciones en sus cuotas de negocio.

1.2. Objetivos

El principal objetivo de esta Tesis Doctoral es el diseñar e implantar un nuevo modelo de desarrollo de software seguro para mejorar la seguridad y para reducir las vulnerabilidades de los proyectos de software, estableciendo el uso de prácticas y controles de seguridad durante todas las fases del ciclo de vida del software bajo un enfoque preventivo.

Para abordar el objetivo general se establecen los siguientes objetivos específicos:

- **Objetivo 1**: elaborar un estudio del estado del arte para buscar las conclusiones de las investigaciones más relevantes que tratan los marcos de trabajo o metodologías especializadas en el desarrollo de software seguro.

- **Objetivo 2**: identificar el conjunto de actividades de seguridad que son consideradas fundamentales y resultan de interés para incluirlos en un nuevo modelo.

- **Objetivo 3**: proponer un nuevo marco de trabajo que incorpore y ejecute de forma sistemática prácticas de seguridad en el proceso de ciclo de vida de desarrollo del software.

- **Objetivo 4**: establecer un procedimiento empresarial óptimo que facilite la implantación del nuevo modelo propuesto y se adapte a las diferentes tipologías de proyectos software de una empresa de desarrollo.

- **Objetivo 5**: comparar las métricas de seguridad y de costes temporales en proyectos de software que han seguido para su desarrollo metodologías estándar, con otros que han utilizado la nueva metodología propuesta.

- **Objetivo 6**: transferir los resultados obtenidos en estas investigaciones a la comunidad científica y a las empresas del sector, para que los avances puedan ser aplicados en factorías de software en particular, y redunden también en la sociedad en general.

1.3 Resumen de las Contribuciones

Los resultados de esta Tesis Doctoral se han publicado en revistas internacionales indexadas en el Journal Citation Reports y en congresos especializados de prestigio. Las publicaciones se enmarcan en el desarrollo seguro del software a nivel empresarial.

Las actividades de seguridad que se han identificado en los modelos de desarrollo de software más populares y especializados para construir software de forma segura por defecto [7] se presentan en el Capítulo 3, siendo consideradas actividades de seguridad recurrentes y, en cierto modo, imprescindibles. El diseño del modelo de desarrollo de software seguro Viewnext-UEx [8], formado por 14 actividades categorizadas en 4 áreas de desarrollo [9], y sus puntos fuertes se exponen en el Capítulo 4. Diversas mejoras del modelo, como la herramienta de entrenamiento de formación [10], la trazabilidad del desarrollo [11], la validación de requisitos [12] y la detección de amenazas [13][14] también son presentadas en el Capítulo 4. La metodología de implantación empresarial del modelo Viewnext-UEx [15] y el ecosistema de herramientas que ayudan a su enfoque preventivo [16] se describen en el Capítulo 5. Los resultados, a modo de comparativa de las métricas de seguridad, y los costes temporales del desarrollo con el método tradicional (enfoque reactivo), comparado con otro desarrollo que aplica el modelo Viewnext-UEx (enfoque preventivo) [17][18] se presentan en el Capítulo 6. La Figura 1.2 muestra un resumen gráfico de las contribuciones mencionadas y que conforman los avances de esta Tesis Doctoral.

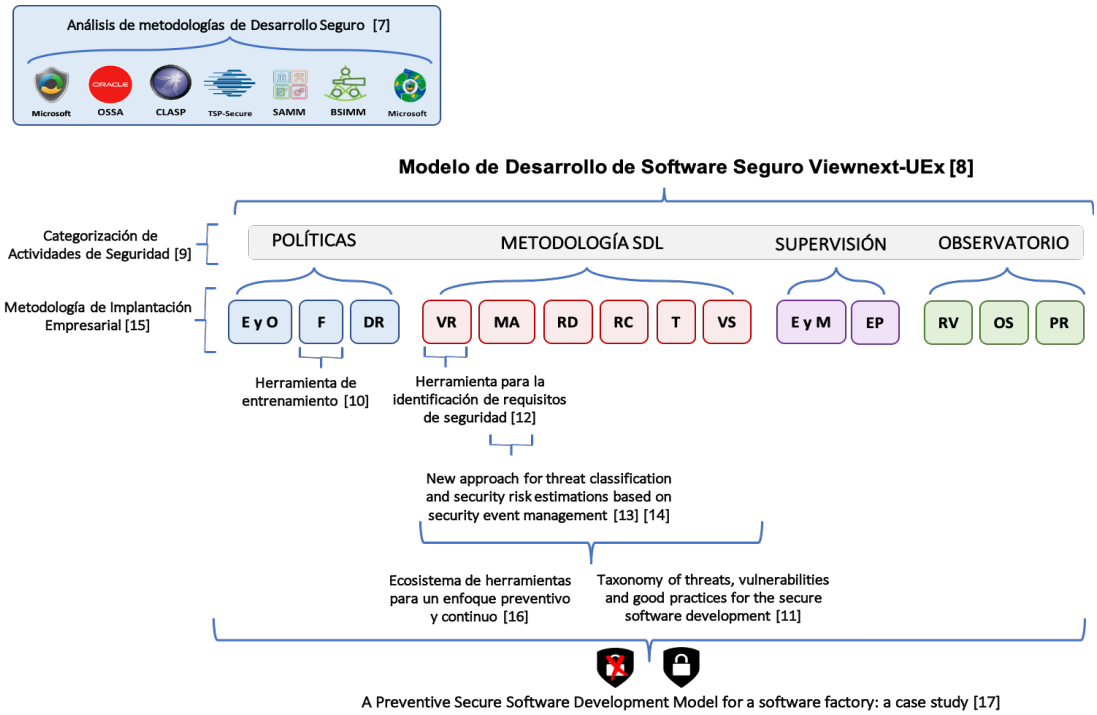


Figura 1.2. Resumen gráfico de las contribuciones de esta Tesis Doctoral. *Fuente: propia.*

1.4 Cronología

Este apartado describe cronológicamente los hitos más reseñables de esta Tesis Doctoral.

- **Abril de 2015.** Se firma la Cátedra de Patrocinio sobre "Seguridad y Auditoría de Sistemas Software" entre la Universidad de Extremadura (UEx) y Viewnext S.A., empresa de Servicios de Tecnología de la Información del grupo IBM España.

- **Noviembre de 2015**. Comienzan los trabajos de investigación dentro del convenio anteriormente mencionado y con ello, la Tesis Doctoral con la firma del documento de Compromiso de Supervisión por la directora de Escuela Internacional de Posgrado de la Universidad de Extremadura Dña. María Jesús Lorenzo Benaya, el coordinador del Programa de Doctorado en Tecnologías Informáticas (TIN) D. Juan Hernández Núñez, el tutor de la Tesis Doctoral D. Fernando Sánchez Figueroa y los directores de la Tesis Doctoral D. Andrés Caro Lindo y D. Pablo García Rodríguez.

- **Enero de 2016**. Las primeras tareas permiten conocer el funcionamiento de la empresa Viewnext S.A. donde se identifican la segmentación de sus servicios, tipología y particularidades de los proyectos de software, así como su departamentalización y filosofía de la empresa.

- **Junio – diciembre de 2016**. Se publica en forma de comparativa el estudio teórico en profundidad de los modelos de desarrollo seguro analizados y la propuesta del modelo Viewnext-UEx ante la compañía.

- **Junio de 2017**. Se diseña la metodología de implantación empresarial del nuevo Modelo de Desarrollo Seguro Viewnext-UEx y comienza a diseñarse el experimento comercial para estudiar la eficacia del modelo.

- **Julio de 2017**. Se realiza una estancia de investigación de un mes de duración, en el Grupo de Investigación "Análisis, Seguridad y Sistemas" (GASS) de la Universidad Complutense de Madrid dirigido por D. Luis Javier García Villalba.

- **Noviembre de 2017**. Se realiza evaluación de seguridad del escenario reactivo desarrollado con la metodología tradicional.

- **Enero de 2018**. Comienza la implantación del nuevo modelo propuesto en el proyecto seleccionado y con ello la formación del equipo de desarrollo en materia de buenas prácticas de desarrollo seguro.

- **Febrero de 2018**. Se desarrolla el escenario preventivo utilizando el modelo Viewnext-UEx y consecutivamente, la evaluación de seguridad de este escenario. También comienza la corrección de los fallos de seguridad en ambos escenarios.

- **Septiembre de 2018.** Se realiza la admisión del proyecto de Tesis Doctoral por parte de la Comisión Académica del Programa de Doctorado en Tecnologías Informáticas (TIN).

- **Octubre de 2018.** Se acepta la inscripción de esta Tesis Doctoral por parte de la Comisión Académica de Doctorado de la Universidad de Extremadura.

- **Enero de 2019**. Se recolectan los datos del experimento realizado y se realiza la comparativa entre el desarrollo realizado siguiendo la metodología reactiva y el modelo implantado en Viewnext S.A.

- **Junio de 2019**. Con la comparativa de ambos módulos desarrollados con distintas metodologías se obtienen las conclusiones determinantes del trabajo de tesis.

- **Noviembre de 2019**. En las Jornadas Nacionales de Investigación en Ciberseguridad de 2019, el Instituto Nacional de Ciberseguridad (INCIBE) premia la contribución que plasma el recorrido y principales conclusiones de esta Tesis Doctoral con su presentación en el evento CyberCamp. Este evento, aun no siendo de ámbito académico, es el acontecimiento más popular de ciberseguridad a nivel nacional.

- **Febrero de 2020**. Comienza la redacción de la Tesis Doctoral.

- **Mayo - julio de 2020.** Se publican en revistas internacionales indexadas en el Journal Citation Reports los resultados más importantes de esta Tesis Doctoral.

- **Octubre de 2020**. Finaliza la redacción de la Tesis Doctoral y se procede a su depósito ante las distintas comisiones de Doctorado de la Universidad de Extremadura.

A los hitos anteriores se suma la presentación anual del plan de investigación y actividades obligatorias descritas en la memoria verificada por la Agencia Nacional de Evaluación de la Calidad y Acreditación (ANECA) del Programa de Doctorado en Tecnologías Informáticas.

1.5 Estructura de la Tesis

Con el propósito de proporcionar una visión completa y contextualizar la memoria de esta Tesis Doctoral, se describe a continuación cada capítulo a modo de resumen:

- **Capítulo 2**. Define los conceptos básicos relacionados con la seguridad del software. Describe las técnicas o enfoques de seguridad utilizados para mitigar vulnerabilidades o anticipar la detección de fallos de seguridad en función de la fase del ciclo de vida del software.

- **Capítulo 3.** Realiza una revisión sobre el estado del arte de las metodologías de desarrollo de software seguro [7]. Se analizan en profundidad las actividades de cada modelo. La comparativa entre los modelos se convierte en la base teórica que permitirá particularizar un modelo de desarrollo de software seguro propio.

- **Capítulo 4.** Presenta el modelo de desarrollo de software seguro Viewnext-UEx [8]. Se detalla la estructura [9] y la descripción de cada actividad de seguridad del modelo. También expone las características que definen el modelo propuesto y sus puntos fuertes [10]–[12].

- **Capítulo 5.** Explica el proceso de implantación empresarial del modelo Viewnext-UEx [15]. Detalla las fases planteadas para optimizar el proceso de transformación laboral. Asimismo, describe las características comunes en los proyectos de software, las herramientas de referencia para mejorar la seguridad construyendo un ecosistema [16], los indicadores de rendimiento de seguridad y los costes temporales para medir los desarrollos.

- **Capítulo 6.** Detalla el diseño del caso de estudio llevado a cabo para medir la eficacia del modelo Viewnext-UEx. Presenta los materiales y métodos del caso de estudio y, por último, presenta los resultados de seguridad y de productividad correspondiente a los escenarios dibujados en el caso de estudio propuesto [17].

- **Capítulo 7.** Expone las principales conclusiones de la tesis y las limitaciones encontradas en el desarrollo de este. De igual forma, define las líneas de investigación futuras y en las que se continúa trabajando.

Además, cada capítulo cuenta con una breve introducción que lo contextualiza y una síntesis final que lo resume.

"El primer paso para crear es conocer."

Capítulo 2

2. Seguridad en el software

Este capítulo hace una revisión de los diferentes aspectos técnicos relacionados con la seguridad del software. En él se explican diversos conceptos que aparecen en otros capítulos de esta Tesis Doctoral. Se presenta la definición de seguridad informática y las características que debe cumplir un sistema seguro. Seguidamente, se define el concepto de vulnerabilidad y se listan las organizaciones u organismos más relevantes que recogen distintas clasificaciones de vulnerabilidades. Posteriormente, se introducen las fases que conforman el ciclo de vida del software y se describen las técnicas o enfoques de seguridad más utilizados para mitigar o detectar fallos de seguridad en cada fase. Por último, se presenta una breve síntesis de lo acontecido en el capítulo.

2.1 Concepto de seguridad informática

El estándar para la seguridad de la información ISO/IEC 27001 [19], publicado en 2005 por la *International Organization for Standardization* (ISO) y por la comisión *International Electrotechnical Commission* (IEC), define la seguridad informática como:

"…un conjunto de medidas técnicas destinadas a preservar la confidencialidad, la integridad y la disponibilidad de la información, pudiendo, además, abarcar otras propiedades, como la autenticidad, la responsabilidad, la fiabilidad y el no repudio."

De esta forma, la seguridad del software viene definida por el cumplimiento de las características que definen a un sistema seguro [19]: integridad, confidencialidad,

disponibilidad, autenticación y no repudio. Por lo tanto, la forma de conseguir el cumplimiento de estas características en el software es a través de la implementación de medidas que preserven su seguridad durante el ciclo de vida de desarrollo del software.

En la motivación de esta Tesis Doctoral se expusieron los datos relativos al incremento de ataques cibernéticos que aprovechan la falta de medidas en el proceso de desarrollo del software para explotar las vulnerabilidades y conseguir su objetivo malicioso. En este sentido, es importante conocer en profundidad las vulnerabilidades más explotadas en la actualidad y las medidas de seguridad a implementar dentro del ciclo de vida del software.

2.2 Vulnerabilidades en el software

Una vulnerabilidad puede considerarse, en términos de informática, al fallo o debilidad que pone en riesgo el funcionamiento correcto de un sistema informático y compromete las, ya mencionadas, características que definen a un sistema seguro.

A continuación, se listan las organizaciones u organismos más relevantes que recogen distintas clasificaciones de vulnerabilidades en los sistemas informáticos:

- ***Open Web Application Security Project*** **(OWASP)**: fundación sin ánimo de lucro, que busca fomentar el desarrollo seguro, la seguridad informática y en especial la seguridad web. *OWASP* presenta periódicamente (normalmente cada 3 años) una clasificación con los 10 tipos de vulnerabilidades más explotadas en aplicaciones web [20]. Esta es la clasificación más utilizada por las empresas desarrolladoras para proteger el código fuente de los proyectos de software.

- ***National Institute of Standards and Technology*** (**NIST)**: agencia que pertenece a la Administración de Tecnologías del Departamento de Comercio de los Estados Unidos. Esta organización publica [21] un listado de 14 vulnerabilidades seleccionadas por su alta probabilidad a ser explotadas junto con un conjunto de herramientas para la detección y mitigación de estas vulnerabilidades.

- ***The Web Application Security Consortium*** **(WASC)**: organismo sin ánimo de lucro compuesto por un grupo internacional de expertos, empresas y profesionales de la seguridad, que publican una clasificación con 49 amenazas en aplicaciones web, organizadas en amenazas y debilidades [22].

- **Instituto SANS**: institución sin ánimo de lucro con el objetivo de reunir información y facilitar la capacitación en el ámbito de la seguridad informática. Junto con MITRE construyeron el *CWE/SANS TOP 25 Most Dangerous Software Errors* [23] donde se recogen 25 errores de software considerados muy peligrosos.

- **Corporación MITRE**: organismo sin ánimo de lucro que desarrolla y mantiene un listado con más de 1000 debilidades comunes en el software denominado *Common Weakness Enumeration* (CWE) [24]. En el año 2020 presentan el CWE Top 25 [25] compuesta por una lista con las 25 debilidades de software consideradas las más peligrosas que se han recogido en los dos últimos años.

Aunque puedan existir otros organismos o instituciones que trabajen iniciativas similares, se han recogido las más conocidas y utilizadas por las empresas desarrolladoras. Muchas de las vulnerabilidades que presentan estos organismos son coincidentes. Por ello y debido a su continua actualización, tomaremos como referencia las categorías estipuladas en el Top 10 de OWASP [20].

2.3 Ciclo de vida del software

El proceso de construcción o desarrollo del software se denomina ciclo de vida (*SDLC - Software Development Life Cycle*). Este proceso se compone de un conjunto de fases perfectamente delimitadas. Según el orden creciente estas fases son la toma de requisitos, el análisis, el diseño, la implementación o codificación, las pruebas, el despliegue y el mantenimiento. La Figura 2.1 muestra las fases o etapas que, generalmente, componen el ciclo de vida del software que, como se visualiza, se encuentran retroalimentadas entre sí.

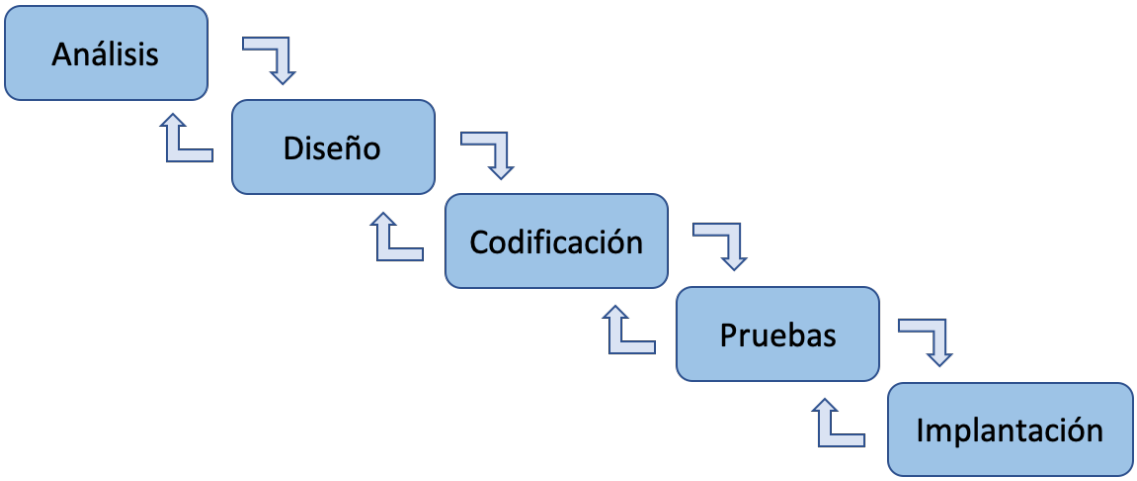


Figura 2.1. Fases del ciclo de vida del desarrollo del software. *Fuente: propia.*

Existen varios tipos de modelos con enfoques diferenciados en función de las actividades y el momento en el que tienen lugar durante el proceso de vida del software. De manera general, un *SDLC* busca planificar, organizar y coordinar el proceso de construcción del software. Los procesos de desarrollo con los paradigmas más conocidos son los siguientes:

- Tradicionales: pensados para dividir el proyecto en fases de manera estructurada. Dentro de los tradicionales son conocidos el modelo en cascada, en el que hasta que no termina una fase no se avanza a la siguiente, o el modelo en espiral, que periódicamente y de manera iterativa gestionan el ciclo del desarrollo.

- Orientado a objetos: cuyo enfoque radica principalmente en la reutilización de software.

- Ágiles: donde el objetivo es prototipar el software y entregarlo lo más pronto posible.

Los paradigmas anteriores no se centran en dedicar especial atención a la seguridad o calidad del software, si no en el proceso de planificación o la entrega del producto al cliente. Así, la falta de integración de prácticas o actividades de seguridad en el proceso de desarrollo puede considerarse una de las causas que originan vulnerabilidades en el software.

En este sentido hay estudios muy conocidos como el de Apvrille y otros en [26], que ponen como referencia considerar la seguridad de forma obligada en todas las tareas y fases que se producen en el ciclo de desarrollo del software, incluidas las fases más tempranas. Del mismo modo, Solinas y otros indican en [27] que la seguridad deja de ser un factor opcional y se hace obligatoria en todo el proceso de construcción.

Siguiendo la base de los estudios anteriores, se decide estudiar diversas técnicas y actividades que apliquen de manera específica la seguridad en cada una de las fases del ciclo de vida del software.

2.3.1 Seguridad en el análisis

Dentro del proceso de análisis se lleva a cabo la toma de requisitos conocido como ingeniería de requisitos. Este proceso recoge la descripción de las características, funcionalidades y comportamiento que va a tener el sistema informático que se quiere desarrollar. En el caso de los modelos ágiles los requisitos se denominan historias de usuario.

Normalmente, en las empresas de desarrollo, los requisitos son divididos en funcionales y no funcionales y son recogidos en un documento de especificación de requisitos que se

pone a disposición de los diseñadores, desarrolladores y *testers* del software. Los requisitos funcionales describen los servicios que presta el sistema y cómo los usuarios van a interactuar con él. Los requisitos no funcionales son los encargados de que el sistema cumplan una serie de propiedades que permitan un comportamiento adecuado. Por lo tanto, los requisitos de seguridad se engloban entre los no funcionales, junto con los de rendimiento o disponibilidad.

La Tabla 2.1 indica, a modo de ejemplo, la identificación de algunos requisitos de seguridad que se extraen a la hora de implementar medidas y consideraciones de protección de autenticación en un sistema de *login* para usuarios, ante ataques automatizados.

Tabla 2.1. Requisitos de seguridad para implementar un sistema de *login*. *Fuente: propia*

Id	Requisito de seguridad
1	Asegurar que las peticiones de *login* se comprueben siempre en el lado del servidor.
2	Implementación de uso de Captchas durante el proceso de *login*.
3	Activar un doble factor de autenticación para operaciones sensibles.
4	Si se supera el número de intentos fallidos, bloquear la cuenta durante un periodo de tiempo (determinar el periodo evitando la denegación de servicio a los usuarios).

Para evitar que la implementación y diseño del software se orienten únicamente a la funcionalidad, como ocurre generalmente, y que las vulnerabilidades comiencen a detectarse en la fase de pruebas, se plantea la introducción de técnicas que faciliten la obtención de requisitos o historias de seguridad. Hay estudios como el de Mellado y otros [28], que se centran en el proceso de ingeniería de requisitos de seguridad para minimizar fallos o posibles ataques maliciosos. En el mismo sentido, Siiskonen y otros [29], presentan cómo conseguir de forma genérica historias de seguridad de usuarios, obtenidas en base a los requisitos funcionales de un sistema informático.

Estos estudios se encuentran muy alineados con el enfoque que se busca en esta Tesis Doctoral anticipándose en la detección de fallos de seguridad desde que el software comienza a pensarse, desde la primera fase del ciclo de vida del software.

2.3.1.1 Técnicas de obtención de requisitos o historias de seguridad

La complejidad en la obtención de requisitos o historias de seguridad ha puesto de manifiesto la creación de distintas técnicas especializadas como las que se exponen a continuación. Estas técnicas facilitan el proceso de ingeniería de requisitos que, lejos de la construcción de lenguajes formales que impiden la implementación en una empresa desarrolladora por su complejidad, presentan enfoques más realista y dinámicos que puedan resultar de interés y aplicación a este tipo de empresas. Se destacan, con este enfoque, las siguientes técnicas o estudios que cumplen el enfoque anterior:

- **OWASP Cornucopia** [30] es una técnica basada en la gamificación para conseguir que los equipos de desarrollo identifiquen requisitos e historias de seguridad mediante el juego de cartas. Se basa en identificar la funcionalidad a realizar y crear un flujo de datos. El juego proporciona la puntuación cuando los desarrolladores comienzan a identificar amenazas para la aplicación. Al final del juego se recogen todas las amenazas y se convierten en requisitos o historias de seguridad.

- **SAFECode** [31] es una guía de seguridad con multitud de ejemplos de requisitos o historias de seguridad que suelen coincidir en las aplicaciones web. Se presentan a la vez la historia de seguridad identificada y las subtareas para llevar a cabo de forma segura su implementación. También recoge del estándar CWE [32] la posible vulnerabilidad a la que se asocia no cumplir la historia de seguridad.

- **IriusRisk** [33] es una herramienta desarrollada por la empresa *Continuum Security* que identifica requisitos y riesgos de seguridad de un sistema informático en base, bien a la contestación de una serie de preguntas que describen funcionalmente la aplicación, o bien, modelando la arquitectura del sistema. Posteriormente, propone las contramedidas que el sistema tiene que tener en cuenta en su proceso de desarrollo.

2.3.2 Seguridad en el diseño

El diseño busca establecer la arquitectura general o las tecnologías que van a dar soporte al sistema informático, definir las estructuras de datos encargadas de gestionar la información que se almacena en el programa y representar tanto las interfaces o pantallas de la aplicación como los algoritmos que definen el comportamiento del sistema informático.

Pasar por alto la seguridad en esta fase podría conllevar que la detección de vulnerabilidades genere un rediseño en el proyecto de software, ocasionando trastornos en la planificación y aumentando los recursos destinados. Por lo tanto, introducir la seguridad en la fase del diseño tiene como objetivo anticipar posibles fallos de seguridad todavía cuando el software se está pensando, antes de comenzar su codificación.

Existen diversas técnicas que permiten abordar la seguridad de manera especializada mientras se realiza el diseño de un sistema informático.

2.3.2.1 Patrones de Diseño Seguro

Desde un punto de vista más teórico es frecuente encontrar en el desarrollo del software diversos patrones introducidos que ayudan a mejorar el producto final. Entre ellos, podemos encontrar modelos creacionales que facilitan la formación o construcción de objetos aplicando encapsulación o abstracción (*Singleton*, *Abstract Factory* o *Factory Method*), estructurales que ayudan a modelar la aplicación en clases (*Wrapper*, *Bridge* o *Facade*) y de comportamiento que buscan relacionar los algoritmos con las clases de la aplicación (*Command*, *Iterator*, *Observer*, o *Template Method*). Sin embargo, estos patrones no prestan atención a la seguridad del software que se produce.

Los patrones de diseño seguro plantean medidas preventivas durante las tareas de diseño del sistema informático con el objetivo de minimizar y reducir la posible aparición de vulnerabilidades. La Tabla 2.2 describe una serie de patrones a disposición de los diseñadores para diseñar una aplicación incluyendo la seguridad de manera genérica, es decir por defecto.

Tabla 2.2. Patrones de diseño seguro y descripción. *Fuente: propia*

Patrón de Diseño	Descripción
Principio del menor privilegio.	Consiste en otorgar a los usuarios los privilegios mínimos necesarios para operar de manera adecuada en una aplicación.
Principios de reducción de superficie de ataque.	Busca limitar el comportamiento de módulos y/o componentes no utilizados en las aplicaciones con el objetivo de reducir las posibilidades de ataques.
Separación de privilegios	Creación de diferentes roles que tienen asignadas un subconjunto de acciones permitidas en función de cada rol.

Criterio de defensa en profundidad (defensa por capas).	Establecer controles de acceso para usuarios en cada uno de los subsistemas de la aplicación, creando varias capas de acceso.
Criterio del "Fallo Seguro".	Controlar las respuestas de los errores y no mostrar información interna de la aplicación que pueda facilitar el objetivo de un atacante.
Simplicidad en el diseño.	Desacoplar los componentes y procesos del software para minimizar las relaciones e interdependencias entre ellos.
Diseño de perfiles y niveles de acceso con listas blancas.	Creación de perfiles con distintos niveles de accesos mediante el uso de una lista de aprobación o *whitelist* que controla los privilegios de cada tipo de perfil.

2.3.2.2 Modelado de amenazas

El proceso de modelado de amenazas utiliza la representación estructurada obtenida a partir de la arquitectura de un sistema informático (modelo) para identificar de forma anticipada los posibles problemas de seguridad (amenazas).

Este modelado refina iterativamente la evolución del modelo abstracto que se obtiene en la fase de análisis con la toma de requisitos. El objetivo es que de manera incremental identifique los posibles vectores de ataque del sistema informático que se va a desarrollar. La Figura 2.2 muestra un ejemplo de caso de uso de una aplicación informática.

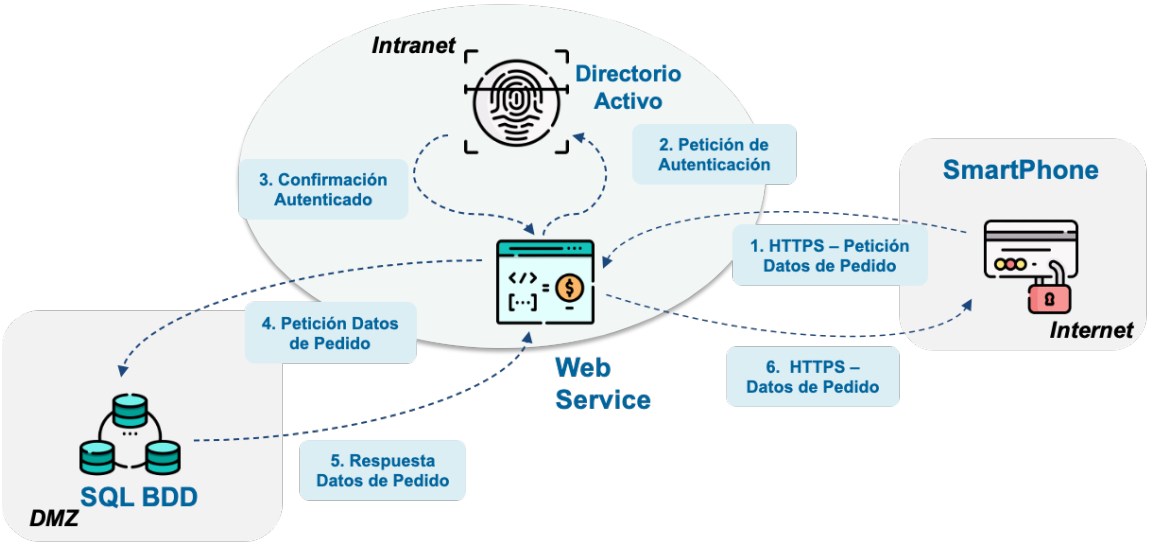


Figura 2.2. Ejemplo de caso de uso preparado para realizar el modelado de amenazas. *Fuente: propia.*

Existen diversos modelos para identificar amenazas siendo los más conocidos y utilizados STRIDE [34] y MITRE ATT & CK [35]. Aunque parecidos, ambos modelos tienen propósitos diferentes. STRIDE modela de manera global el sistema con el objetivo de identificar áreas de vulnerabilidad que un atacante puede explotar, clasificándolas en clases en función del sistema. Por contra, MITRE ATT & CK identifica de manera aislada las diferentes etapas de un ataque, basándose en el flujo general de un ataque, sin tener en cuenta todo el sistema.

En este sentido se encuentran estudios que ya utilizan STRIDE para el modelado de amenazas en aplicaciones. Así, Xin y Xiao-fang [36] se basan en el modelo de amenazas STRIDE en un caso de estudio para analizar la seguridad de una aplicación bancaria. Otros como Venkatasen y Maniproposein [37], usan STRIDE para mejorar una aplicación del gobierno electrónico e Hirano y otros [38] confían en STRIDE para analizar la seguridad y mitigar problemas de análisis forense de almacenamiento en Infraestructura como servicio (IaaS). Por último, Seifert y Rez [39] utilizan STRIDE para determinar posibles problemas de seguridad de la arquitectura de sistemas para asistencia sanitaria.

En base a estos estudios y dado que el modelo STRIDE permite dos enfoques distintos por elemento (obtener un diagrama de elementos potencialmente vulnerables, identificados y categorizados) y por interacción (detección de interacciones vulnerables en un sistema), se considera más ventajoso que MITRE ATT & CK para modelar amenazas en los proyectos de software de una empresa desarrolladora.

La Tabla 2.3 presenta las diferentes categorías del modelo STRIDE junto a un ejemplo y el control de seguridad asociado extraído de las características de un sistema seguro.

Tabla 2.3. Categorización de amenazas del modelo STRIDE y control de seguridad asociado. *Fuente: propia*

Id	Tipo de amenaza	Ejemplo	Control de seguridad
S	Suplantación de Identidad (*Spoofing*)	Ataque para acceder ilegalmente y utilizar credenciales (usuario y contraseñas) de otros usuarios.	Autenticación
T	Manipulación de Datos *(Tampering)*	Ataque que busca modificar maliciosamente datos, como los de una base de datos y la alteración de los datos transmitidos entre varios sistemas.	Integridad

R	Repudio (*Repudiation*)	Ataque para realizar operaciones ilícitas en un sistema que no rastrea operaciones prohibidas y que no se puede garantizar el origen.	No repudio
I	Divulgación de Información *(Information disclosure)*	Amenaza mediante la lectura de un archivo con información confidencial al que, sin vulnerar el sistema, no se tiene acceso.	Confidencialidad
D	Denegación de Servicio (*Denial of service*)	Ataque cuyo objetivo es dejar temporalmente inaccesible o sin servicio a un sistema o aplicación.	Disponibilidad
E	Elevación de Privilegios (*Elevation of privilege*)	Ataque dirigido a obtener acceso privilegiado a recursos inaccesibles por defecto o a información no autorizada del sistema.	Autorización

A continuación, se detallan las tres fases a seguir para realizar el modelado de amenazas. A modo de ejemplo se modela el sistema representado en la Figura 2.2 con la aplicación Microsoft Threat Modeling Tool [40].

1. **Descomposición del sistema y modelado**: esta fase identifica los diferentes elementos y representa la arquitectura del sistema informático. Estos elementos son los actores principales, dependencias externas, zonas de confianza, puntos de entrada, activos, almacenes de datos y flujos o interacciones. La Figura 2.3 representa el modelado del sistema y la arquitectura con los componentes y las relaciones que se producen en la aplicación de la Figura 2.2.

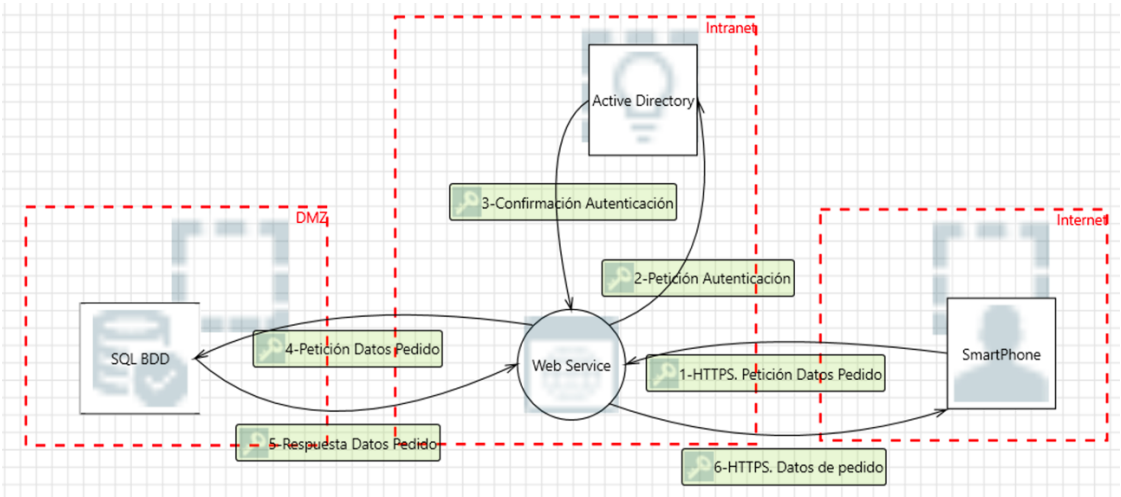


Figura 2.3. Modelado de amenazas del sistema representado en la Figura 2.2. *Fuente: propia.*

2. **Identificación de amenazas**: la aplicación de Microsoft Threat Modeling Tool identifica de manera genérica las potenciales amenazas a las que está expuesto un sistema modelado. La Tabla 2.4 expone, a modo de ejemplo, el comportamiento de algunas de las amenazas del sistema que representa la Figura 2.2 y que ha sido modelado en la Figura 2.3.

Tabla 2.4. Amenazas potenciales del modelado realizado con la Figura 2.3. *Fuente: propia*

Id*	Amenaza	Interacción	Comportamiento
E	Elevación al cambiar el flujo de ejecución en el servicio web	1-HTTPS. Datos Pedido	Un atacante puede pasar datos al servicio web para cambiar el flujo de ejecución del programa dentro del servicio web a elección del atacante.
E	El servicio web puede estar sujeto a la elevación de privilegios mediante la ejecución remota de código	1-HTTPS. Petición Datos Pedido	SmartPhone puede ejecutar código de forma remota para el servicio web.
E	Elevación mediante suplantación	1-HTTPS. Petición Datos Pedido	Web Service puede hacerse pasar por el contexto de SmartPhone para obtener privilegios adicionales.
D	Flujo de datos 1-HTTPS. Petición Datos Pedido potencialmente interrumpido	1-HTTPS. Petición Datos Pedido	Un agente externo interrumpe el flujo de datos a través de un límite de confianza en cualquier dirección.
D	Posible bloqueo del proceso o detención del servicio web	1-HTTPS. Petición Datos Pedido	El servicio web se bloquea, se detiene, o se ejecuta lentamente
R	Posible rechazo de datos por servicio web	1-HTTPS. Petición Datos Pedido	SmartPhone afirma que no recibió datos de un proceso fuera del límite de confianza.
T	El almacén de datos SQL BDD podría estar dañado	4-Petición Datos Pedido	Los datos que fluyen a través de 4-Petición Datos Pedido pueden ser manipulados por un atacante. Esto puede provocar daños en SQL BDD. Es preciso asegurar la integridad del flujo de datos al almacén de datos SQL BDD.

3. **Identificación de medidas de protección**: es la fase encargada de identificar las contramedidas que mitigan las amenazas potenciales antes de que se conviertan en vulnerabilidades. Una muestra de contramedida para el ejemplo desarrollado y las amenazas de la categoría (E-Elevación de privilegios) es la aplicación de patrones de diseño (Tabla 2.2) como el principio del menor privilegio o el criterio de defensa en profundidad.

2.3.2.3 Otras técnicas

Existen otras técnicas que ayudan a modelar las diversas formas que puede utilizar un atacante para alcanzar un objetivo malicioso en un sistema informático. Entre las que destacan los casos de abuso o los árboles de ataque por:

- **Casos de abuso o casos de mal uso**: son una adaptación de los caso de uso pero introduciendo el punto de vista del atacante. Utilizan los casos de uso planteando un mal uso de ellos como si de un ataque se tratase. Hay estudios como el de McDermott y Fox [41] o el de C. Sia [42] que plantean el uso de esta técnica para mejorar la seguridad del software.

- **Árboles de ataque**: es un modelo que, desde el punto de vista del atacante, a través de la representación en árbol de las acciones (nodos del árbol) permite diseñar la forma de ataque en un sistema informático (raíz). La estructura utilizada suele hacer uso del nodo raíz para establecer el objetivo del ataque y el resto de nodos junto con las relaciones de conexión entre ellas diseñan el ataque informático. Estudios como los de Ten y otros [43] o el de Schneier [44] hacen uso de esta técnica con el objetivo de anticipar la detección de vulnerabilidades en el software.

2.3.4 Seguridad en la codificación

El proceso de transformar el diseño de un sistema informático en lenguaje máquina y darle comportamiento según las especificaciones de los requisitos se denomina codificación del software. Durante el proceso de codificación se realizan tareas de depuración cuyo objetivo es eliminar errores, generalmente de semántica, sintáctica y lógica.

La seguridad en la fase de codificación se puede abordar de manera preventiva, mediante el seguimiento de buenas prácticas de codificación segura o de manera reactiva, realizando la revisión manual o automática del código posteriormente desarrollado. A continuación, se explicitan estas técnicas.

- **Buenas prácticas de codificación segura**: se trata de un conjunto de directrices y consejos que un programador debe tener en cuenta para prevenir errores comunes de seguridad. Alguna de estas buenas prácticas son la validación de entradas, manejo de errores, controles criptográficos, cifrado de contraseñas, etc.

- **Revisión manual de código**: una de las formas de identificar vulnerabilidades en el código fuente es realizar la revisión manual de código por parte de otro desarrollador. Sin embargo, esta tarea, además de compleja por el alto nivel que se requiere, duplica la inversión de recursos dedicados en un proyecto de software. Por este motivo, es una técnica es poco utilizada.

- **Revisión automática de código**: se inspecciona y revisa el código fuente de manera automática aplicando una serie de reglas predefinidas que detectan las vulnerabilidades más comunes. Esta técnica precisa que otro desarrollador revise los resultados obtenidos tras el análisis automático para evitar falsos positivos, es decir, vulnerabilidades señaladas que realmente no lo son y terminan siendo descartadas. El uso de herramientas facilita el proceso de revisión de la seguridad del código fuente desarrollado. En este sentido, hay estudios como el de Díaz y Bermejo [45], que se centran en el análisis estático de código fuente, comparando el rendimiento de nueve herramientas (*CBMC, K8-Insight, PC-lint, Prevent, Satabs, SCA, Goanna, Cx-enterprise, Codesona*r). Otro como el de Baca y otros [46] realizan un caso estudio para evaluar el análisis de código estático en software industrial.

2.3.5 Seguridad en las pruebas

Al finalizar la construcción del software se llevan a cabo las pruebas. Esta fase se encarga de verificar el comportamiento correcto del sistema informático según los requisitos establecidos. Tradicionalmente, la fase de pruebas ha sido más utilizada para que, de manera reactiva, se detecten las vulnerabilidades en el software.

Se toma como referencia el estudio de Felderer y otros [47], que repasan diferentes técnicas para hacer *testing* de seguridad, destacando los test de penetración y el análisis dinámico. Al igual que con la revisión del código, las pruebas de seguridad se pueden realizar manualmente, donde un tester simula ser un usuario final y ejecuta distintos casos de usos con el objetivo de vulnerar el sistema y encontrar fallos de seguridad, y automáticamente mediante el escaneo de vulnerabilidades.

- **Test de penetración**: este tipo de test puede ser tanto manual como automático. Se ejecutan simulando un ataque real con la configuración de un usuario externo al sistema informático. El usuario que realiza el intento de penetración tiene únicamente información limitada sobre el sistema y solo interactúa con las partes públicas de la

aplicación. Este tipo de pruebas reactivas son las más utilizadas y son consideradas las más eficaces para encontrar vulnerabilidades en las empresas desarrolladoras de software.

- ***Fuzz testing***: es una técnica dinámica que envía datos aleatorios a un sistema informático con el objetivo de que se comporte de manera errónea. Se considera una técnica eficaz para encontrar vulnerabilidades al conseguir que los sistemas fallen con datos inesperados. Los últimos avances de técnicas de *fuzzing* se basan en mutaciones para la generación de datos.

Con respecto a la selección de herramientas para hacer *testing* de seguridad, en el estudio de Qasaimeh y otros [48] se evalúan la efectividad y precisión de herramientas como: *Acunetix WVS, Burp Suite, NetSparker, Nessus y OWASP ZAP*, centradas en identificar posibles vulnerabilidades de las aplicaciones web para un caso de estudio.

2.3.6 Seguridad en la implantación y mantenimiento

La implantación consiste en poner el software a total disposición para su uso definitivo por parte de los usuarios. El proceso de poner en producción el software es el plan de implantación y es considerado uno de los más complejos debido a la multitud de componentes que intervienen y a la alta probabilidad de que se produzcan fallos en el sistema.

El plan de implantación debe recoger las comprobaciones de medidas adecuadas para garantizar la seguridad del software.

Con respecto al proceso de mantenimiento del software, se debe estar en continua vigilancia y rastreo de nuevas vulnerabilidades desconocidas (*zero day*) consultando fuentes fiables de prestigio. Estas continuas revisiones reducen el tiempo del software inseguro en un proyecto.

Por otro lado, la detección de vulnerabilidades en software en producción debería articular un plan de respuesta a incidentes de manera que se ofrezca una respuesta rápida y organizada que mitiguen el incidente de seguridad y minimicen el tiempo de respuesta hasta la resolución del problema. Por ello, lo ideal es contar con una metodología de pasos definidos y personas involucradas para aplicar desde el momento que se encuentre el fallo hasta que se solucione.

2.4 Síntesis del capítulo

En este capítulo se han explicado los conceptos de seguridad informática y de vulnerabilidad en el software. También, se han expuesto varias clasificaciones de vulnerabilidades cuyo estudio ayudan a mejorar la seguridad en el proceso de desarrollo. Seguidamente, se han listado las fases que conforman el ciclo de vida del software y, para cada una de ellas, se han detallado técnicas o enfoques cuyo objetivo principal es prevenir vulnerabilidades en el software y detectar anticipadamente posibles fallos de seguridad.

Bloque II

Metodologías

"El conocimiento se obtiene buscando la diferencia."

Capítulo 3

3. Modelos de Desarrollo de Software Seguro

El objetivo general de este capítulo es conocer el enfoque y las actividades que recogen diversos marcos de trabajo especializados en el desarrollo de software seguro. En primer lugar, se introduce el concepto de Ciclo de Vida del Desarrollo del Software Seguro, centrado en incorporar actividades de seguridad específicas en el desarrollo de aplicaciones. Seguidamente, se analizan en profundidad estos marcos de trabajo, ampliamente conocidos en el mundo empresarial. El capítulo termina con una breve síntesis que expone cuáles son las actividades que se consideran fundamentales a tener en cuenta para para el diseño de cualquier nuevo modelo seguro.

3.1 Análisis de metodologías de desarrollo seguro

El concepto de Ciclo de Vida de Desarrollo de Software Seguro (*SSDLC - Secure Software Development Life Cycle*) toma importancia en esta investigación, debido a que este tipo de metodología introduce actividades y controles específicos para la seguridad durante las fases tradicionales del ciclo de vida del software. El enfoque preventivo y de anticipación que plantean evita que se propaguen fallos de seguridad durante todas las fases de desarrollo, igualmente, impide que puedan multiplicarse sus efectos perniciosos y, por ende, el incremento de los costes temporales o económicos implicados en su resolución.

A continuación, se analizan los principales modelos de ciclo de vida de desarrollo de software seguro porque, como se observará en próximos apartados, son pocos los estudios

fiables y publicados en revistas de alto impacto que analizan metodologías específicas de seguridad y las comparan entre sí.

3.1.1 Microsoft Security Development Lifecycle

El *Microsoft SDL* [49] es el modelo de ciclo de vida de desarrollo de software seguro de la multinacional estadounidense Microsoft. Considerado el más popular, es utilizado por algunas empresas tecnológicas.

Precedido en el año 2002 por la iniciativa *Trustworthy Computing*, fue convertido en 2004 en una directiva obligatoria del proceso integral del desarrollo de software de Microsoft. En la actualidad, sigue desempeñando un papel trascendental en la integración de la seguridad y de la privacidad de los productos de Microsoft, con la finalidad de reducir el número y la gravedad de las vulnerabilidades del software que desarrolla la compañía. Es utilizado para asegurar los productos con ciclos de desarrollos más largos, como el sistema operativo Windows o el paquete Microsoft Office.

Parte de un enfoque dividido en cinco áreas de capacidades alineadas con las fases del ciclo de vida de desarrollo de software, y cuatro niveles de madurez para los procedimientos: básico, estandarizado, avanzado y dinámico.

En la Figura 3.1 se visualizan el conjunto de las diecisiete actividades de seguridad que forman el *Microsoft SDL*. Estas se ejecutan siguiendo el orden de las fases del ciclo de vida de desarrollo de software tradicional, manteniendo, a su vez, cierta flexibilidad para agregar otras directivas y/o técnicas que permitan crear una metodología de desarrollo de software única para una organización.

Figura 3.1. Actividades de seguridad del modelo *Microsoft SDL*. *Fuente: Microsoft.*

3.1.2 Agile Microsoft Security Development Lifecycle

En el año 2012, ante el crecimiento de las metodologías ágiles, Microsoft adapta el proceso de seguridad para aplicaciones más ligeras publicando el *SDL-Agile* [50]. Las fases del proceso sufren una reorganización con respecto al modelo tradicional. De esta forma Microsoft propone mapear las tareas en función del ciclo de liberación del software. La Figura 3.2 proporciona esta visión del proceso.

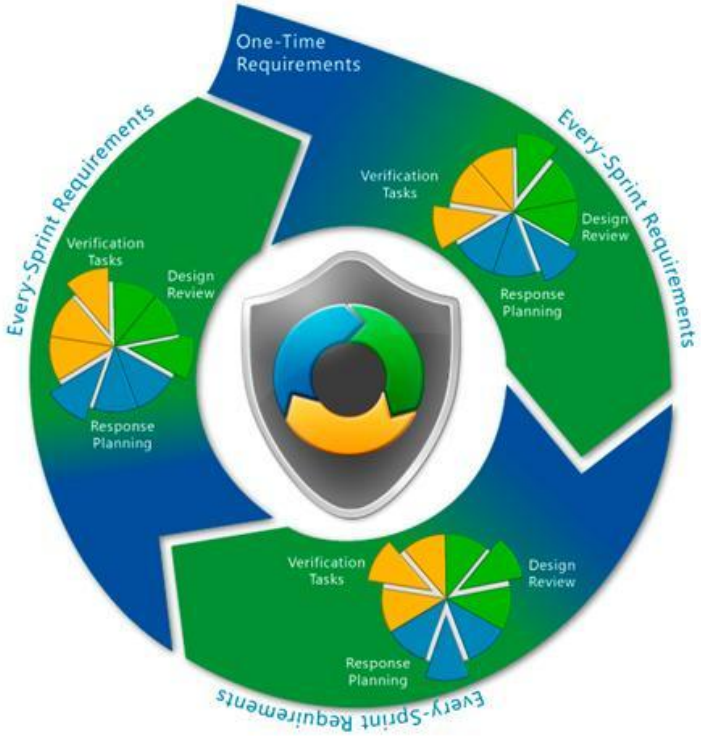


Figura 3.2. Proceso del *SDL-Agile*. *Fuente: Microsoft*.

La diferencia fundamental es que, en cada *sprint*, los requisitos se presentan como tareas en tres categorías: requisitos esenciales para la seguridad, requisitos o tareas regulares obligatorios durante el ciclo de vida y requisitos únicos, generalmente, fáciles de implementar. Las tareas regulares a su vez se dividen en tareas de verificación, de diseño y de planificación.

3.1.3 Oracle Software Security Assurance

La compañía norteamericana *Oracle Corporation* integra la seguridad en el proceso de desarrollo y mantenimiento de la seguridad de sus productos a través del modelo *Oracle Software Security Assurance* (OSSA) [51]. El modelo incluye actividades de seguridad en el diseño, implementación, pruebas y el mantenimiento. La Figura 3.3 resume su enfoque:

Figura 3.3. Organización del modelo OSSA. *Fuente: Oracle.*

Como se puede visualizar en la Figura 3.3, el modelo OSSA se estructura en las siguientes etapas:

- ***Secure Development Standards***: es la fase núcleo de OSSA, abarca las reglas y principios de codificación segura, ejemplos de mal uso, requisitos de seguridad para el diseño y las funciones complejas. También engloba la formación en seguridad de sus desarrolladores.

- ***Product Definition***: establece los requisitos de seguridad origen de la codificación segura y los criterios de seguridad que debe cumplir el producto, siendo revisados estos en cada paso de los procesos de implementación y liberación del producto.

- ***Product Development***: en esta etapa se producen continuas revisiones que validan el cumplimiento de las normas de codificación segura. Además, se realiza una revisión adicional en la seguridad del diseño y se ejecutan pruebas de seguridad, tanto reactivas como destructivas, con el objetivo de conocer la calidad del producto.

- ***Ongoing Assurance***: es el ciclo encargado de realizar, de manera obligatoria y antes de la implantación, las pruebas de seguridad automatizadas en la fase de pre-producción. Seguidamente, la fase de post-producción evalúa la resistencia de nuevas amenazas y el cumplimiento de los *Common Criteria*, mediante la ISO-15408 y FIPS 140-2.

3.1.4 Comprehensive Lightweight Application Security Process

La metodología *Comprehensive Lightweight Application Security Process* (CLASP) [52] trata la construcción de software como un conjunto de piezas especializadas en seguridad. Busca integrarse de manera sencilla con el resto de los procesos de desarrollo. Destaca por la adaptación y conexión entre los procedimientos de construcción del software.

CLASP se estructura en 5 vistas y 24 tareas de seguridad, que divide en 7 categorías de buenas prácticas y 104 vulnerabilidades o fallos de seguridad. En la Figura 3.4 se visualiza la relación entre estos elementos:

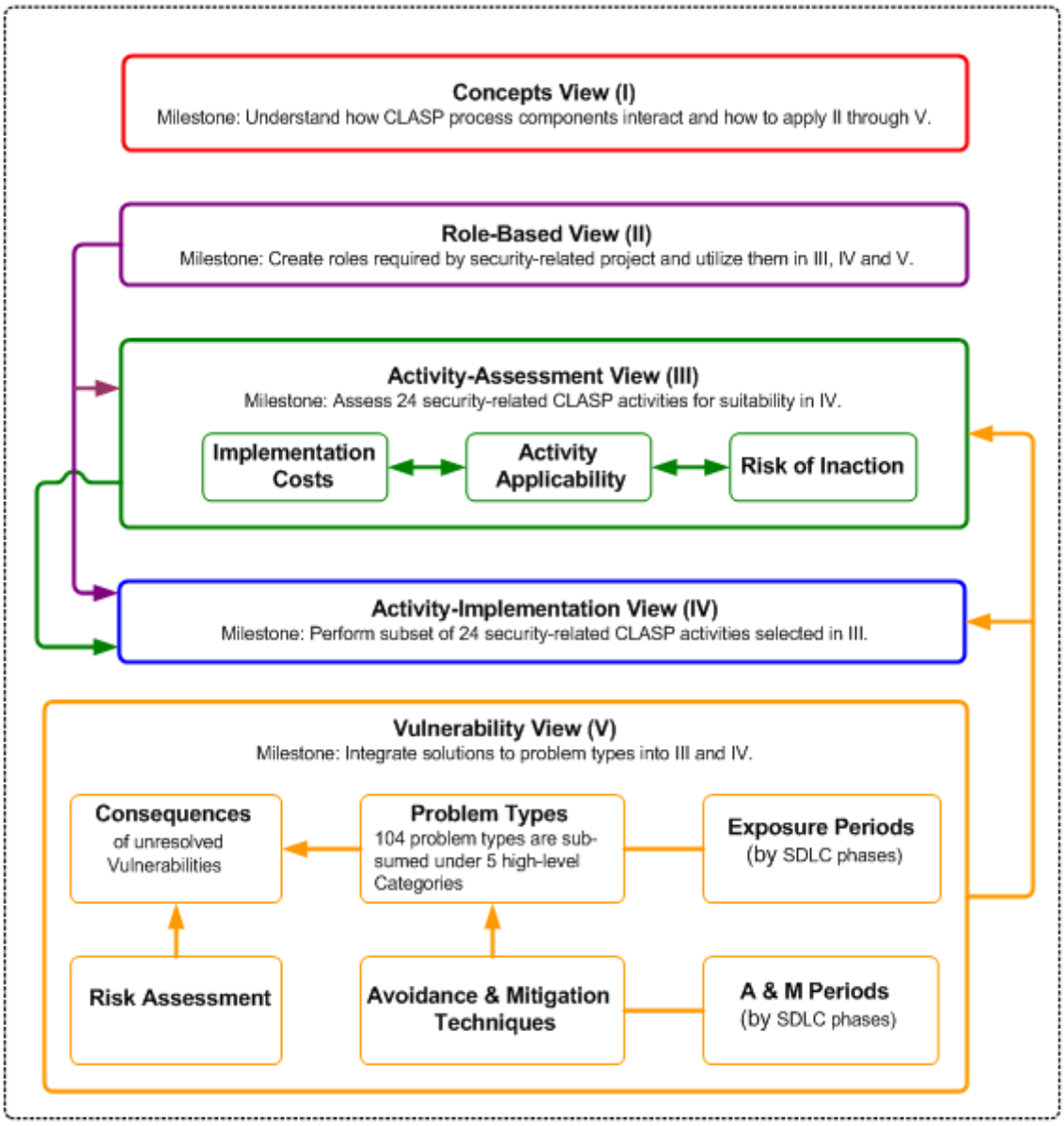


Figura 3.4. Organización del modelo CLASP. *Fuente: OWASP.*

Las cinco vistas descritas a alto nivel son las siguientes:

- ***Concepts view***: proporciona la introducción al modelo y detalla la interacción y conexión con el resto de vistas. También contiene los pasos a seguir para adoptar la metodología.
- ***Role-Based view***: congrega las acciones que optimizan la organización del conjunto de los grupos implicados en una metodología de seguridad. Los roles que se tienen en cuenta son: gerente de proyecto, arquitecto, ingeniero de requisitos, diseñador, programador (equipos de desarrollo) y auditor de seguridad.
- ***Activity-Assesment view***: realiza la selección entre las 24 actividades de seguridad teniendo en cuenta sus costes, la aplicabilidad y el riesgo de inacción. También se mide la capacidad de integración en el proceso de construcción del software.
- ***Activity-Implementation view***: vista encargada de ejecutar las actividades de seguridad que han sido seleccionadas en la vista anterior.
- ***Vulnerability view***: vista que contiene un catálogo de 104 vulnerabilidades, agrupadas en 5 categorías, que forma la base de los errores más comunes identificados con anterioridad sobre incumplimientos de codificación segura. El objetivo de esta vista es tener una base de conocimiento que sirva de consulta.

3.1.5 Team Software Process Secure

El *Team Software Process Secure* (TSP-Secure) [53] tiene el propósito de mejorar el nivel de calidad y productividad de proyectos de desarrollo de software. Es un marco de trabajo gestionado por el Equipo de Procesos de Software (TSP) y el Equipo de Respuesta ante Incidencias de Seguridad Informática (CERT-CSIRT), ambos ligados al Instituto de Ingeniería del Software (SEI).

El *TPS-Secure* está formado por un conjunto de procesos y métodos que aplican los principios fundamentales de ingeniería del software, teniendo presente el impacto de la seguridad. Esta metodología de desarrollo seguro integra técnicas de codificación segura y proporciona los medios para realizar una planificación que garantice la seguridad del software. También, favorece la gestión de equipos de desarrollo para mejorar su autonomía e incluye formación específica y concienciación en materia de seguridad para los desarrolladores. En la Figura 3.5. se especifican los procesos involucrados:

Figura 3.5. Procesos involucrados en TSP-Secure. *Fuente: SEI.*

3.1.6 Software Assurance Maturity Model

El *Software Assurance Maturity Model* (SAMM) [54] es un marco de trabajo desarrollado por el organismo sin ánimo de lucro *Open Web Application Security Project* (OWASP). El modelo SAMM contempla la seguridad en las aplicaciones implicando a las personas que deciden sobre los riesgos de seguridad de las aplicaciones.

Una de sus principales características es la flexibilidad. Ha sido diseñado para aplicarse en diversos escenarios: organizaciones completas, en una sola línea de negocio o incluso en un proyecto en particular, entidades pequeñas, medianas o grandes. Además, es posible adaptarla a cualquier estilo de desarrollo.

SAMM se basa en cuatro funciones de negocio (gobierno, construcción, verificación e implementación) que se conectan con los procesos de desarrollo de software y 12 actividades de seguridad específicas. En la Figura 3.6 se define la estructura del modelo:

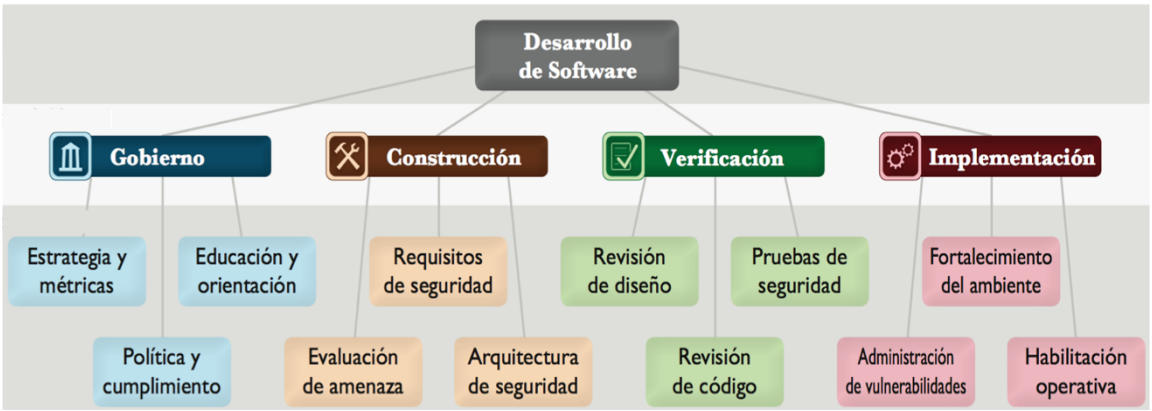


Figura 3.6. Funciones de negocio y prácticas de seguridad del modelo SAMM. *Fuente: OWASP.*

Las prácticas de seguridad concretan tres niveles de madurez como cumplimiento de los objetivos de seguridad fijados por cada una. Cada nivel de madurez persigue la consecución de un objetivo más sofisticado y con mayor exigencia. El desempeño de cada actividad se determina en función de indicadores específicos y métricas de éxito. Asimismo, el modelo SAMM dispone de planes de mejora y especialización para cada práctica de seguridad.

3.1.7 Building Security in Maturity Model Framework

El *Building Security In Maturity Model Framework* (BSIMM) [55] nace en el año 2008 de un estudio encargado de cuantificar las prácticas de seguridad existentes en multitud de organizaciones. El estudio se ha convertido en el reflejo del estado de la seguridad del software en las organizaciones. En la décima versión han participado un total de 122 organizaciones industriales.

El modelo BSIMM se organiza en 12 actividades de seguridad englobadas en 4 dominios: gobernanza, inteligencia, puntos de contacto con el SSDL y despliegue.

Modelo de Referencia para la Seguridad del Software (MRSS)			
Gobernanza	**Inteligencia**	**Puntos de Contacto con el SSDL**	**Despliegue**
Estrategia y Métricas	Modelos de Ataque	Análisis de la Arquitectura	Pruebas de Penetración
Cumplimiento y Política	Características de Seguridad y Diseño	Revisión de Código	Entorno del Software
Capacitación	Normas y Requisitos	Pruebas de Seguridad	Gestión de Configuración y Gestión de Vulnerabilidades

Figura 3.7. Estructura modelo BSIMM. *Fuente: BSIMM.*

Así, cada dominio y cada práctica tienen una meta de alto nivel. El dominio de gobernanza busca mejorar la transparencia y cumplimiento del nivel de seguridad del modelo. El de inteligencia crear recursos propios para pensar como un atacante, para crear patrones de seguridad y para la creación de normativas de seguridad propias. Los puntos de contacto incluyen las prácticas del SDLC con el objetivo de controlar la calidad el software desarrollado. Por último, el despliegue también tiene como meta el control de la calidad a través de pruebas de penetración y del manejo de incidentes.

3.2 Comparativa de marcos de trabajo seguros

Los marcos de trabajo analizados en los apartados anteriores tienen en común el enfoque *preventivo* sobre la seguridad en la construcción del software. Coinciden en controlar la seguridad durante todo el ciclo de vida, incluyendo las fases más tempranas. Todas ellas integran un conjunto de actividades o procedimientos específicos con la finalidad de reducir las posibilidades de ataques o intrusiones, y detectar con rapidez la aparición de constantes vulnerabilidades.

Existen pocos estudios que analicen en profundidad uno o varios marcos de trabajo, metodologías y/o estándares que por defecto apliquen la seguridad durante todo el proceso de desarrollo del software. Destacan [56]–[59], aunque ninguno hace una revisión exhaustiva que permita realizar comparaciones entre los modelos, un objetivo marcado en esta investigación.

El estudio realizado por Mohino y otros en [56], se considera uno de los más profundos en la actualidad. Esto es debido al número de marcos de trabajo de seguridad que tiene en cuenta, la profundidad del análisis y las conclusiones obtenidas. Entre ellas, se plantea la necesidad de priorizar los aspectos de seguridad en el desarrollo del software, y aprovechar los beneficios de los modelos ágiles. Aunque no se realiza un análisis de cada modelo por actividad, se indica el enfoque que cubre cada modelo en función de las fases del ciclo de vida del software y se identifican los modelos que presentan reportes empresariales.

En la misma línea que el anterior, en Grégoire, B. De Win y otros [59] se comparan teóricamente las similitudes de los modelos CLASP y *Microsoft SDL*. Su trabajo se amplía en [58] para incluir en la comparativa el modelo *Touchpoints* [60], agrupando las similitudes entre los procesos según la fase del ciclo de vida tradicional en la cual se realizan.

Otro estudio de interés es el realizando por Williams y otros en [57], donde se analiza el uso que hacen más de 100 empresas de un subconjunto de 113 actividades de seguridad específicas para el software. La evolución de este análisis ha fundamentado las actualizaciones del modelo *Building Security In Maturity Model Framework* (BSIMM).

Por último, destacar el estudio realizado Mohammed y otros en [61], donde a través de un mapeo sistemático de la literatura se recogen diferentes enfoques de seguridad que se aplican en el desarrollo del software. Este estudio identifica 13 enfoques centrados de manera

transversal en la seguridad de todo el proceso de construcción del software. Sin embargo, ninguno de ellos presenta pruebas empíricas de su implantación o aplicación empresarial.

Con todo ello, los estudios encontrados no proponen nuevas actividades de seguridad, ni la creación de nuevos modelos adaptados a las necesidades actuales. Generalmente, estos estudios se limitan a mostrar las similitudes extraídas de los modelos comparados y además, no tienen en cuenta algunos marcos de trabajo como *Microsoft ASDL, TSP-Secure* y *SAMM*, que pasan desapercibidos al no encontrarse estudios que los analicen, a pesar de que son modelos de desarrollo seguro muy conocidos.

Por tanto, identificada la necesidad de realizar un análisis en profundidad de los marcos o modelos de desarrollo seguro, se presenta en [7] el conjunto de las actividades de seguridad de los modelos anteriormente expuestos. Las actividades, analizadas en detalle, se clasifican en función de la fase del ciclo de vida donde se ejecutan. De este modo, la Tabla 3.1 permite visualizar la diferencia entre los modelos estudiados, coincidentes en su aplicación en entornos industriales.

Tabla 3.1. Actividades de seguridad por modelo y fase de ejecución. *Fuente: propia*

Fase/Modelo	Microsoft SDL/ASDL	OSSA	CLASP	TSP-Secure	SAMM	BSIMM
Políticas			• Política de seguridad global • Recursos y límites de confianza • Roles de usuario y capacidades • Especificar el entorno operativo		• Política y cumplimiento	• Cumplimiento y política
Formación	• Formación	• Formación	• Programa de sensibilización sobre seguridad institucional	• Formación	• Educación y orientación	• Formación
Análisis	• Análisis de riesgos de seguridad y privacidad • Requisitos de seguridad	• Definición de riesgos • Requisitos de seguridad	• Análisis superficie de ataques y riesgos • Requisitos de seguridad	• Análisis de riesgos • Requisitos de seguridad	• Evaluación de amenazas • Requisitos de seguridad • Arquitectura de seguridad	• Requisitos y estándares • Análisis de las arquitecturas
Diseño	• Modelos de riesgos • Analizar superficie de ataques • Requisitos de diseño	• Modelado de amenazas • Revisión del diseño • Caso de abuso	• Modelado de amenazas • Guía y principios del diseño seguro • Diagrama de clases de seguridad • Casos de abuso	• Modelado de amenazas • Diseño seguro • Casos de abuso	• Modelado de amenazas • Revisión del diseño	• Modelos de amenazas • Características y diseño de seguridad
Implementación	• Usar herramientas aprobadas • Prohibir funciones no seguras	• Reglas y principios de codificación segura • Revisión de funciones complejas • Revisión de código	• Configuración de seguridad en BD • Análisis estático	• Guía y estándares de desarrollo seguro • Análisis estático	• Revisión de código	• Revisión de código • Entorno del software
Pruebas	• Análisis dinámico • Fuzz testing	• Testing de seguridad	• Testing de seguridad	• Testing de seguridad • Fuzz testing	• Testing de seguridad	• Testing de seguridad • Pruebas de penetración
Pre-Release	• Revisión de seguridad final • Lanzamiento o archivado	• Validación del cumplimiento	• Validación de seguridad			
Post-Release	• Plan de respuesta a incidentes	• Corrección de vulnerabilidades	• Dirección de problemas de seguridad reportados	• Lista de vulnerabilidades con sus posibles mitigaciones	• Gestión de vulnerabilidades • Securización del entorno • Habilitación operativa	• Gestión de configuración y de vulnerabilidades
Métricas	• Umbrales de calidad y límites de errores	• Criterios de seguridad que deberá cumplir el producto	• Monitorizar métricas de seguridad	• Medir métricas de seguridad	• Estrategia y métricas	• Estrategia y métricas
Uso empresarial	X	X	X	X	X	X

3.2.1 Análisis de la comparativa

La comparativa de actividades anterior permite concluir que la forma de construir software seguro y de calidad es realizando desarrollos guiados por normas y estándares que fuercen el uso de prácticas seguras. Los encargados de minimizar las posibilidades de error y vigilar el seguimiento de la evolución del software son los controles de seguridad.

La identificación de actividades de seguridad repetidas de manera recurrente en los modelos estudiados, evidencian la necesidad de considerar estas prácticas de seguridad en cualquier marco de trabajo propio, que comparta el objetivo de producir software seguro.

La Figura 3.8 muestra las semejanzas y diferencias entre los modelos analizados. Para representar cada modelo como un conjunto y detectar de forma sencilla las relaciones entre ellos, se utiliza el diagrama de Venn Euler. En el círculo central (como intersección de todos los modelos) se identifican las actividades de seguridad comunes a los modelos estudiados.

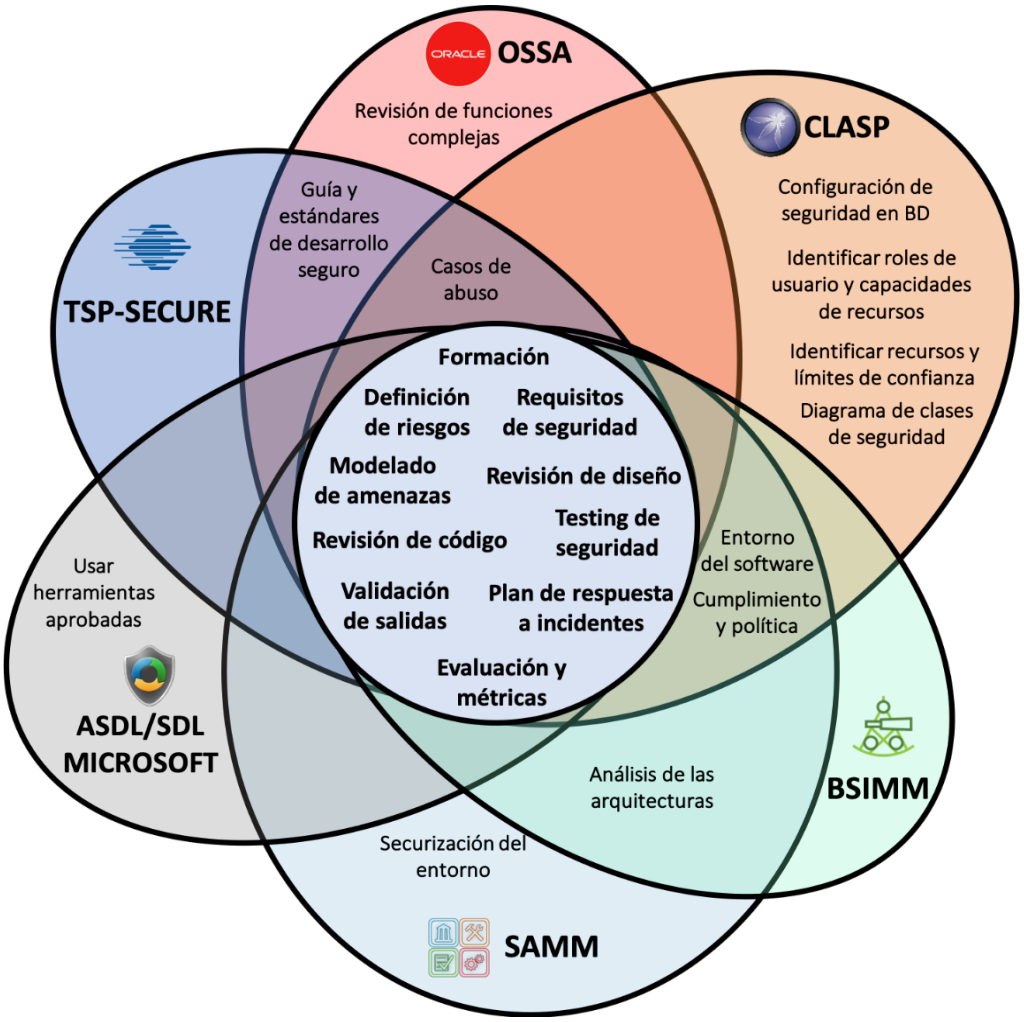


Figura 3.8. Comparativa de los modelos de desarrollo seguro estudiados. *Fuente: propia.*

3.3 Síntesis del capítulo

En este capítulo se han conocido las similitudes y diferencias de varios modelos de desarrollo que se centran en prevenir la seguridad desde las fases iniciales del ciclo de vida del software.

Como conclusión preliminar, se han identificado 10 actividades coincidentes entre los modelos analizados, que han sido representados en la parte central de la Figura 3.8. Dichas actividades, se han considerado de interés obligatorio en la creación o particularización de nuevos modelos de desarrollo de software seguro. Las actividades son las siguientes:

- **Formación**: especialización de todos los miembros de los grupos implicados en el desarrollo del software en materia de análisis de riegos, amenazas y codificación segura.
- **Definición de riesgos**: identificación y definición de los riesgos dependientes del modelo de negocio del proyecto donde implantar un modelo de desarrollo seguro.
- **Requisitos de seguridad**: obtención, definición y validación de los requisitos o historias de seguridad de la aplicación a desarrollar.
- **Modelado de amenazas**: análisis y modelado de la superficie de ataques en función del modelo de negocio del proyecto o cliente.
- **Revisión del diseño**: revisión de la seguridad del diseño analizando los activos, entidades externas y sus relaciones.
- **Revisión del código**: análisis estático del código en busca de posibles vulnerabilidades.
- **Testing de seguridad**: realización de pruebas o test de intrusión para verificar la calidad y seguridad del software.
- **Validación de seguridad:** verificación final del software, previo a su entrega e implantación en explotación de las aplicaciones.
- **Implantación de un plan de respuesta a incidentes:** medida encargada de la respuesta ante la detección de nuevas vulnerabilidades en los proyectos.
- **Evaluación y métricas:** medición y control del desempeño de las actividades de seguridad y del cumplimiento normativo con respecto a la seguridad de los proyectos.

"Después de aprender, el reto está en ver qué puedes mejorar."

Capítulo 4

4. Modelo de Desarrollo Seguro Viewnext-UEx

Este capítulo comienza presentando la estructura del Modelo de Desarrollo de Software Seguro Viewnext-UEx. Seguidamente, se presenta cada actividad de seguridad del modelo con su descripción, su objetivo, las técnicas que aplica y los resultados esperados tras su ejecución. Se exponen también las características principales y los puntos fuertes que describen y avalan al modelo. El capítulo termina con una breve síntesis de lo acontecido en el mismo.

4.1 Modelo de Desarrollo de Software Seguro Viewnext-UEx

El Modelo de Desarrollo de Software Seguro Viewnext-UEx (modelo Viewnext-UEx) se enmarca, como se ha comentado anteriormente, en el convenio firmado en el año 2015 entre la empresa Viewnext. S.A. (filial de IBM) y la Universidad de Extremadura (UEx).

El modelo, siguiendo el paradigma preventivo, ha sido diseñado para incluir actividades específicas de seguridad en todas las fases del ciclo de vida del software, siendo el principal propósito la mejora de la seguridad del software, adelantando la detección de vulnerabilidades y optimizando los tiempos de desarrollo. Con todo ello, se pretende evitar que la aparición de fallos de seguridad desajuste los tiempos y costes de la ejecución de

proyectos de software. La Figura 4.1 presenta un esquema del modelo Viewnext-UEx , su estructura y las actividades de seguridad que engloba.

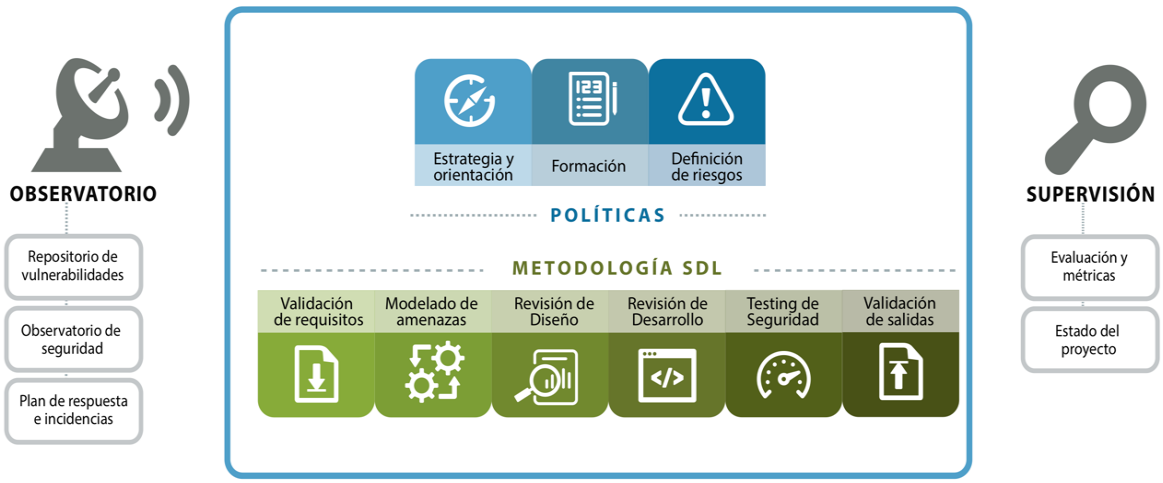


Figura 4.1. Modelo Viewnext-UEx . *Fuente: Viewnext.*

4.2 Estructura del modelo Viewnext-UEx

El modelo Viewnext-UEx se define como un metamodelo basado en una metodología sistemática que se estructura en 4 áreas de desarrollo y 14 actividades de seguridad.

Estructurar el modelo en 4 áreas de desarrollo permite categorizar las actividades de seguridad en relación al objetivo general que debe cumplir cada área, como se recoge en [9]. Esta categorización ayuda a planificar y sistematizar el conjunto de actividades de seguridad durante todo del ciclo de vida del software.

En la Tabla 4.1 se presenta el modelo Viewnext-UEx agrupando las actividades de seguridad según al área de desarrollo al que pertenecen y en función de la fase de ejecución del ciclo de vida en la que se llevan a cabo [8].

Tabla 4.1. Actividades de seguridad del modelo Viewnext-UEx en función del área de desarrollo donde se engloba y la fase de ejecución del ciclo de vida. *Fuente: propia*

Área de desarrollo	Actividad de Seguridad	Fase de ejecución
Políticas	Estrategia y orientación	Todo el ciclo de vida
	Formación	
	Definición de riesgos	

Metodología SDL	Validación de requisitos	Análisis
	Modelado de amenazas	Análisis Diseño
	Revisión de Diseño	Diseño
	Revisión de Desarrollo	Codificación
	Testing de Seguridad	Codificación Pruebas
	Validación de salidas	Pre-release
Supervisión	Estado del Proyecto	Todo el ciclo de vida
	Evaluación y métricas	
Observatorio	Observatorio de seguridad	
	Repositorio de vulnerabilidades	
	Plan de respuesta a incidentes	Post-release

4.2.1. Áreas de desarrollo

El modelo Viewnext-UEx se estructura en 4 áreas de desarrollo: Políticas, Metodología SDL, Observatorio y Supervisión. Cada área de desarrollo abarca los procesos comunes de las actividades que engloba.

En este sentido, se mantiene la línea de otros modelos estudiados que categorizan sus actividades o procedimientos de seguridad en relación a diferentes parámetros. SAMM [54] divide sus 12 prácticas de seguridad en las funciones de negocio a la que afecta su ejecución. Otro ejemplo distinto puede considerarse el modelo de *Microsoft* [49] que ordena las actividades de seguridad en función de la fase de ejecución dentro del ciclo de vida del software tradicional.

A continuación, se listan y describen las 4 áreas de desarrollo señaladas en la Tabla 4.1 indicando las actividades que recogen. Las actividades marcadas con un asterisco (*****) son inexistentes en el resto de modelos estudiados y, por ello, propias del modelo Viewnext-UEx. Precisamente buscan cubrir las carencias detectadas en los modelos.

- **Políticas**: enfocada a definir los principios y directrices globales que garanticen el cumplimiento de los objetivos de seguridad del software en cualquier proyecto donde se aplique el modelo. Las prácticas de seguridad de esta categoría requieren una participación activa de todos los grupos implicados en la ejecución de un proyecto software. La finalidad de este área es unificar las acciones dedicadas a mejorar la seguridad del software en la organización. Las actividades de seguridad englobadas en este área de desarrollo son:

  - Estrategia y orientación.
  - Formación en seguridad.
  - Definición de riesgos.

- **Metodología SDL**: define los procesos y actividades relacionados con la construcción y pruebas de software seguro por defecto. Las actividades de seguridad de esta área son:

  - Validación de requisitos.
  - Modelado de amenazas.
  - Revisión del diseño.
  - Revisión del desarrollo.
  - Testing de seguridad.
  - Validación de salidas.

- **Supervisión**: recoge la información de los procesos y actividades de seguridad definidas en el modelo para obtener una visión global sobre el estado de la seguridad de cada proyecto. Se encarga de la revisión de vulnerabilidades e incidencias operativas durante el desarrollo del software. Realiza la evaluación final del funcionamiento y seguridad del software antes de pasar a la fase de producción. Las actividades de este área son:

  - Estado del proyecto (*).
  - Evaluación y métricas.

- **Observatorio**: área de seguridad encargada de la vigilancia continua para la detección anticipada de nuevas vulnerabilidades en el software. Se encarga de establecer y desarrollar nuevas líneas de investigación que estudien novedosas técnicas de ataques desconocidas y de la definición de un plan de contingencias ante posibles incidentes de seguridad. Las actividades que incluye son:

- Observatorio de seguridad (*).
- Repositorio de vulnerabilidades (*).
- Plan de respuesta a incidentes.

4.2.2. Actividades de seguridad

Descritas las 4 áreas que integran el modelo Viewnext-UEx, se pasa a detallar las 14 actividades de seguridad que las componen y cuyo objetivo es sistematizar las acciones de seguridad que se aplican al software durante todo su ciclo de vida.

Como apoyo a la realización de cada actividad y facilitando la consecución de sus objetivos, en [16] se selecciona un conjunto de herramientas conformando un ecosistema, que en apartados futuros se explicarán en mayor detalle.

A continuación, se presenta a modo de tabla cada una de las actividades que conforman el modelo Viewnext-UEx. En cada una de ellas se indica el nombre, el área de desarrollo en el que se engloban, una descripción específica de la misma, los objetivos de su aplicación, las acciones o técnicas aplicadas y los resultados esperados, argumentando así todas las actividades integradas en el nuevo modelo propuesto, tanto aquellas que ya aparecían en modelos anteriores o aquellas nuevas propias del modelo Viewnext-UEx**.**

4.2.2.1 Estrategia y orientación

Tabla 4.2. Descripción de la actividad "Estrategia y orientación". *Fuente: propia*

Nombre	Estrategia y orientación	**Área**	Políticas
Descripción	Esta actividad se ejecuta de forma transversal a lo largo de todo el progreso y desarrollo de cada proyecto. Aborda la definición de la política del cumplimiento de la seguridad del software dependiendo del sector del proyecto y la orientación sobre la protección e identificación de los proyectos sobre las vulnerabilidades más explotadas en la actualidad [62].		
Objetivo	Cumplir el plan estratégico unificado y las normas específicas de seguridad para el desarrollo de software de un proyecto teniendo en cuenta el sector al que pertenece.		
Técnicas o acciones	Estudiar la normativa sectorial en cuanto a la ley de protección de datos del proyecto de software.		

	Clasificar los datos y aplicaciones basados en el riesgo de negocio del cliente.
Resultados	Política particularizada para el cumplimiento de la seguridad establecida en el proyecto software donde se implanta el modelo.

4.2.2.2 Formación

Tabla 4.3. Descripción de la actividad "Formación". *Fuente: propia*

Nombre	Formación	**Área**	Políticas
Descripción	La formación en materia de seguridad se ejecuta durante el proceso de implantación del modelo Viewnext-UEx.		
Objetivo	Concienciar al equipo de trabajo sobre los efectos perniciosos de los fallos de seguridad y capacitarlo para el desarrollo de software más seguro.		
Técnicas o acciones	Formación sobre análisis de seguridad. ▪ Normativas sectoriales. ▪ Riesgos de seguridad. ▪ Requisitos o historias de seguridad. ▪ Modelado de amenazas. Formación en patrones de diseño seguro. ▪ Reducción de la superficie de ataques. ▪ Defensa en profundidad. ▪ Principio de privilegios mínimos. ▪ Configuraciones predeterminadas seguras. Formación en buenas prácticas de codificación segura. ▪ Top 10 de OWASP. ▪ Validación de entradas. ▪ Administración de sesiones. ▪ Política de control de acceso. ▪ Protección y tratamiento de datos. ▪ Sistemas y arquitecturas de seguridad. ▪ Herramienta de entrenamiento de desarrollo seguro [10].		
Resultados	Conseguir personal formado en desarrollo de software seguro.		

4.2.2.3 Definición de riesgos

Tabla 4.4. Descripción de la actividad "Definición de riesgos". *Fuente: propia*

Nombre	Definición de riesgos	**Área**	Políticas
Descripción	Identificación de los riesgos de seguridad basado en la normativa del sectorial al que pertenece el proyecto software, la funcionalidad del software y la arquitectura del entorno de ejecución. Los riesgos de negocio son distintos dependiendo del cada sector: médico, banca, eléctrico, venta comercial, etc.		
Objetivo	Identificar y comprender los riesgos de seguridad de alto nivel y su impacto, en base a la actividad y modelo de negocio del proyecto. Detectar las partes del proyecto que van a requerir modelos de riesgos, revisiones del diseño de seguridad, pruebas de penetración y alto testeo de seguridad.		
Técnicas o acciones	Estudiar: ▪ Los riesgos del modelo negocio del proyecto. ▪ La normativa específica. ▪ Los activos del proyecto. Realizar una evaluación de los riesgos de seguridad y privacidad de los datos.		
Resultados	Listado de los riesgos de seguridad en base al modelo de negocio del proyecto y de los requisitos del software para la normativa sectorial y la arquitectura del proyecto a desarrollar.		

4.2.2.4 Validación de requisitos

Tabla 4.5. Descripción de la actividad "Validación de requisitos". *Fuente: propia*

Nombre	Validación de requisitos	**Área**	Metodología SDL
Descripción	Actividad preventiva para el desarrollo del software centrada en conocer proactivamente los requisitos o historias de seguridad del proyecto de software.		
Objetivo	Especificar los requisitos o historias de seguridad en base a los requisitos funcionales, riesgos conocidos y la lógica de negocio del proyecto.		
Técnicas o acciones	En esta actividad se pueden usar algunas de las siguientes técnicas: ▪ Validación de requisitos o historias en base a los requisitos funcionales haciendo uso de distintas técnicas [29], [30], [62].		

	▪ Casos de abuso o mal uso consistentes en diseñar escenarios de uso indebido que permiten el análisis de la aplicación desde el punto de vista del atacante. ▪ Prototipado de seguridad que permite extraer los requisitos de seguridad en base a las especificaciones del proyecto y permite evaluar si son inconsistentes y/o están incompletos.
Resultados	Listado de requisitos o historias de seguridad de acuerdo con los riesgos conocidos en la actividad anterior.

4.2.2.5 Modelado de amenazas

Tabla 4.6. Descripción de la actividad "Modelado de amenazas". *Fuente: propia*

Nombre	Modelado de amenazas	**Área**	Metodología SDL
Descripción	Medida preventiva y proactiva en el desarrollo del software enfocada específicamente en conocer las amenazas del proyecto en base a los requisitos o historias de seguridad extraídos de la actividad anterior.		
Objetivo	Reducir drásticamente la superficie de ataques y la posibilidad de vulnerabilidades en el proyecto de software.		
Técnicas o acciones	Dos técnicas básicamente destacan para ser utilizadas en esta actividad: ▪ Analizar la superficie de ataques en base a los riegos obtenidos previamente. ▪ Modelar las amenazas para conocer las partes más del software realizando el diseño arquitectónico y funcional.		
Resultados	Listado de amenazas categorizadas en función de los puntos de entrada, los activos, los módulos y las entidades externas que componen la aplicación. Conjunto de modelos de amenazas construidos, adaptables y reutilizables en el proceso de desarrollo del software.		

4.2.2.6 Revisión del diseño

Tabla 4.7. Descripción de la actividad "Revisión del diseño". *Fuente: propia*

Nombre	Revisión del diseño	**Área**	Metodología SDL
Descripción	Actividad reactiva que analiza y evalúa el diseño y arquitectura del software con el fin de detectar problemas relacionados con la seguridad. Se realiza antes de que se inicie la codificación del software para reducir las futuras posibilidades de rediseños que conlleven incremento en los costes debido a problemas de seguridad.		

Objetivo	Controlar formalmente el proceso de diseño del software y validar la utilización de componentes de seguridad.
Técnicas o acciones	Aplicación de la revisión por pares del diseño siguiendo los criterios del Estándar para la Verificación de Seguridad de Aplicaciones (ASVS) de OWASP [63] : ▪ Principio del menor privilegio. ▪ Principios de reducción de superficie de ataque. ▪ Separación de privilegios (defensa por capas). ▪ Criterio de defensa en profundidad. ▪ Criterio del "Fallo Seguro". ▪ Diseño de autenticación. ▪ Diseño de perfiles y niveles de acceso. ▪ Diseño de protección contra Denial of Service. ▪ Definición de mensajes de error. ▪ Prevención de divulgación de información. ▪ Manejo de información sensible. ▪ Almacenamiento seguro. ▪ Transferencia segura. ▪ Encriptación y Hashes. ▪ Interacción con bases de datos. ▪ Auditoría y Logging. ▪ Manejo seguro de errores.
Resultados	Informe en forma de *checklist* (lista de verificación) sobre los puntos de verificación tras aplicar los patrones de seguridad y las buenas prácticas de diseño seguro.

4.2.2.7 Revisión del desarrollo

Tabla 4.8. Descripción de la actividad "Revisión del desarrollo". *Fuente: propia*

Nombre	Revisión del desarrollo	**Área**	Metodología SDL
Descripción	Actividad reactiva que comienza en el momento de la codificación para analizar y evaluar automáticamente la seguridad del código desarrollado con el propósito de identificar posibles vulnerabilidades que se clasifican según el tipo y la criticidad.		
Objetivo	Detectar posibles vulnerabilidades a nivel del código fuente desarrollado.		
Técnicas o acciones	Esta actividad realiza de forma automatizada un análisis estático del código fuente a bajo nivel en busca de posibles vulnerabilidades. Este análisis permite, además, hacer una revisión escalable siguiendo las directivas de codificación segura, a la vez que resuelven los problemas de		

	calidad y fallos de seguridad encontrados. El análisis y revisión se encuentra en continua ejecución y tiene en cuenta: ▪ Detectar vulnerabilidades en el software. ▪ Detectar fragmentos de código inalcanzable o repetido. ▪ Evitar fragmentos de código de complejidad innecesaria. ▪ Verificar que todos los bucles tienen condición de salida y funcionan de forma correcta. ▪ Evitar errores en las estructuras de datos y en los accesos a las mismas. ▪ Comprobar cualquier operación con datos que pueda ser susceptible de enmascarar un problema. ▪ Etc.
Resultados	Informe del estado de la seguridad del código fuente y listado de las acciones realizadas para resolución de los problemas de seguridad.

4.2.2.8 Testing de seguridad

Tabla 4.9. Descripción de la actividad "Testing de seguridad". *Fuente: propia*

Nombre	Testing de seguridad	**Área**	Metodología SDL
Descripción	Actividad reactiva que comienza cuando concluye la revisión del desarrollo (actividad anterior) y finaliza antes del lanzamiento del software. Se encarga de buscar vulnerabilidades aplicando una metodología mixta (pruebas automatizadas y manuales) para explotar el software desde la visión del atacante. Las vulnerabilidades se clasifican según el tipo y criticidad.		
Objetivo	Realizar pruebas de seguridad basándose en la implementación y los requisitos de seguridad del software. Detectar posibles vulnerabilidades a nivel de la ejecución final de la aplicación.		
Técnicas o acciones	En esta actividad se realizan pruebas de penetración o intrusión, tanto automáticas como manuales. Destacan dos tipos: ▪ Método de caja negra: Intento de intrusión al sistema sin tener conocimientos de este, generando una situación de ataque realista en el ambiente de ejecución del proyecto de software. ▪ Método de caja blanca: Intento de penetración al sistema conociendo su funcionalidad por completo, con la finalidad de probar al máximo los límites de seguridad del software.		
Resultados	Informe del estado de la verificación de la seguridad de la aplicación desarrollada y listado de las acciones realizadas para resolver los fallos de seguridad.		

4.2.2.9 Validación de salidas

Tabla 4.10. Descripción de la actividad "Validación de salidas". *Fuente: propia*

Nombre	Validación de salidas	**Área**	Metodología SDL
Descripción	Actividad que se ubica lo más cerca posible del lanzamiento del software. Realiza una revisión global del resto de actividades de seguridad llevadas a cabo en el área de la Metodología SDL, siguiendo las directrices de seguridad fijadas en las políticas y acordadas con el cliente.		
Objetivo	Garantizar la corrección de los fallos de seguridad y de privacidad identificados en la revisión de seguridad final.		
Técnicas o acciones	Comprobación final antes de terminar el proyecto para garantizar la corrección y mitigación de todos los problemas de seguridad y de privacidad en base a los umbrales de calidad, rendimiento y seguridad determinados con el cliente.		
Resultados	Versión definitiva del software sin errores y sin vulnerabilidades.		

4.2.2.10 Estado del proyecto

Tabla 4.11. Descripción de la actividad "Estado del proyecto". *Fuente: propia*

Nombre	Estado del proyecto (*****)	**Área**	Supervisión
Descripción	Actividad de nueva creación, única en el modelo Viewnext-UEx, encargada de dar a conocer el estado del proyecto actual mediante la revisión continua de los umbrales de seguridad fijados en las políticas de cumplimiento. La gestión de la seguridad de varios productos de un mismo proyecto de software resulta compleja debido al avance en paralelo de cada aplicación.		
Objetivo	Cumplir las implicaciones en cuanto al nivel de seguridad y tiempo de resolución de incidencias de estándares como el Modelo de Madurez y Capacidad Integrado (CMMI).		
Técnicas o acciones	Revisar los cuadros de mandos de las aplicaciones de un proyecto de software para adaptar los recursos destinados a solventar situaciones de seguridad críticas y facilitar la resolución de incidencias y cumplimiento de estándares CMMI.		
Resultados	Conocer de forma inmediata el estado del proyecto de software con respecto al nivel de seguridad, la detección y resolución de vulnerabilidades.		

4.2.2.11 Evaluación y métricas

Tabla 4.12. Descripción de la actividad "Evaluación y métricas". *Fuente: propia*

Nombre	Evaluación y métricas	**Área**	Supervisión
Descripción	Actividad que se desarrolla de manera transversal a todo el desarrollo del proyecto de software. Encargada de medir los indicadores de seguridad y el coste temporal del resto de actividades de seguridad.		
Objetivo	Medir los indicadores de rendimiento del resto de actividades del modelo y garantizar el cumplimiento de las regulaciones a nivel proyecto.		
Técnicas o acciones	Para evaluar la eficacia del modelo Viewnext-UEx se proponen distintas métricas: ▪ Indicadores de seguridad: número de vulnerabilidades, criticidad, tipo y categoría. ▪ Indicadores de productividad: tiempo dedicado a cada actividad del modelo.		
Resultados	Listado de indicadores de rendimiento clasificados correctamente.		

4.2.2.12 Repositorio de vulnerabilidades

Tabla 4.13. Descripción de la actividad "Repositorio de vulnerabilidades". *Fuente: propia*

Nombre	Repositorio de vulnerabilidades (*****)	**Área**	Observatorio
Descripción	Actividad de nueva creación, única en el modelo Viewnext-UEx, que mantiene información actualizada sobre las características de las vulnerabilidades identificadas en el proyecto de software. Traslada esta información de manera proactiva a las fases iniciales del proyecto, produciendo así la retroalimentación y mejora continua. Esta actividad facilita a los desarrolladores el acceso a la detección de fallos en sus propios desarrollos y retroalimenta la futura formación a recibir por parte del equipo de trabajo.		
Objetivo	Mejorar y progresar empíricamente la seguridad de los proyectos de software, trasladando la información generada con las medidas reactivas a las actividades preventivas e iniciales del desarrollo.		
Técnicas o acciones	Gestión de un repositorio de vulnerabilidades actualizado que sirva como base de conocimiento y almacene información sobre sus criticidades, tipologías, tecnologías específicas, usos de servicios y, sobre todo, pasos para resolverlas con éxito.		
Resultados	Mayor agilidad en la mitigación de vulnerabilidades o fallos de seguridad.		

4.2.2.13 Observatorio de seguridad

Tabla 4.14. Descripción de la actividad "Observatorio de seguridad". *Fuente: propia*

Nombre	Observatorio de seguridad (*****)	**Área**	Observatorio
Descripción	Actividad de nueva creación dentro del modelo Viewnext-UEx dedicada al rastreo constante de vulnerabilidades desconocidas (*zero day*) a través de la consulta de fuentes fiables de prestigio. La información de esta actividad se transfiere a toda la compañía desarrolladora para el beneficio de todos los proyectos.		
Objetivo	Identificar mediante una continua vigilancia la aparición de nuevas vulnerabilidades y/o técnicas de ataques sobre el software desarrollado.		
Técnicas o acciones	Consulta continua de fuentes de prestigio que publican vulnerabilidades y técnicas de ataque recientes. Investigación en nuevas técnicas o herramientas que permitan automatizar las labores de generación del software seguro y su validación.		
Resultados	Minimización del tiempo que el software de un proyecto está comprometido por la aparición de nuevas vulnerabilidades.		

4.2.2.14 Plan de respuesta a incidentes

Tabla 4.15. Descripción de la actividad "Plan de respuesta a incidentes". *Fuente: propia*

Nombre	Plan de respuesta a incidentes	**Área**	Observatorio
Descripción	Actividad reactiva que se aplica una vez que el software se encuentra en producción y que se encarga de responder con rapidez y de manera organizada ante un incidente de seguridad. Esto conlleva una metodología, requiriendo para ello un grupo de personas especializado.		
Objetivo	Minimizar el impacto de un incidente de seguridad y reducir el tiempo de respuesta ante su resolución.		
Técnicas o acciones	Esta actividad aplica las siguientes fases: ▪ Localizar al equipo de trabajo especializado e identificar los recursos destinados a la resolución de la incidencia. ▪ Evaluar en detalle la incidencia. ▪ Priorizar, identificar y evaluar los pasos para su resolución. ▪ Restablecer el sistema una vez la incidencia ha sido resuelta.		
Resultados	Secuencia de pasos e identificación de recursos concretos que reducen el tiempo de respuesta ante un posible incidente de seguridad.		

4.3 Características del modelo Viewnext-UEx

El modelo Viewnext-UEx se diseña con un propósito holístico. Busca la integración global en los diversos proyectos de software de una compañía desarrolladora. Para su creación se tienen en cuenta las distintas tipologías de proyectos software que pudiera tener una factoría de software, los enfoques particulares de los modelos que se utilizan (ágil, cascada, etc.), e incluso la fase de desarrollo en la que comienza la intervención de la empresa que desarrolla (toma de requisitos, análisis funcional o diseño).

El modelo propuesto tiene un enfoque preventivo, integrado y flexible, permitiendo la retroalimentación para mejorar su finalidad y la reutilización para incorporarlo y extrapolarlo a otras compañías de software. Las principales características que se desprenden del modelo Viewnext-UEx son las siguientes:

- **Preventivo**: el modelo Viewnext-UEx integra la seguridad en el desarrollo de software desde las fases iniciales del ciclo de vida del software. Realiza la trazabilidad entre los riesgos de seguridad, los requisitos de seguridad y las buenas prácticas de código. De esta forma, las medidas reactivas tomadas durante el proceso del desarrollo del software se convierten en preventivas, consiguiendo la anticipación a nuevos fallos de seguridad.

- **Integrado**: las actividades de seguridad del modelo Viewnext-UEx se encuentran categorizadas y planificadas en el ciclo de vida del desarrollo del software para realizar una ejecución sistemática, en función de la fase del ciclo de vida que corresponda. El equipo de desarrollo adopta las actividades de seguridad, evitando un proceso disruptivo en sus tareas diarias.

- **Flexible**: la forma en la que se diseña el modelo Viewnext-UEx facilita la adaptación a cualquier tipo de proyecto software (nuevo o mantenimiento), enfoques de desarrollo (ágil o tradicional) y equipo de trabajo. Las actividades de seguridad son las mismas independientemente del ciclo de liberación (prolongados o frecuentes) y del enfoque (tradicional o ágil). El modelo ha sido pensado para soportar cambios y ajustes personalizados.

- **Retroalimentado**: el modelo Viewnext-UEx se retroalimenta con la detección de vulnerabilidades y fallos de seguridad encontrados en el proceso de desarrollo. Hace uso de la información extraída en las fases finales para trasladarla a fases más tempranas. Así se evita la reproducción continua de errores y se consigue progresivamente mejorar la

seguridad general del producto final. Se utilizan fuentes de información confiables [64]–[67] para encontrar la información específica sobre vulnerabilidades conocidas.

- **Reutilizable**: Las diversas características que definen al modelo Viewnext-UEx permiten que, con pequeños cambios, se pueda adaptar a cualquier organización o empresa desarrolladora. Las actividades de seguridad complementan desde el punto de vista seguro a otras actividades que se realizan en cualquier tipo de desarrollo de software, lo cual facilita su reutilización.

4.4 Puntos fuertes del modelo Viewnext-UEx

Existen dos cuestiones singulares del modelo Viewnext-UEx que lo diferencian y refuerzan sobre el resto. El primero es el enfoque de la formación en materia de seguridad ofrecida a los desarrollares y auditores. El segundo es plantear un proceso de trazabilidad que ensamble y conecte de manera continua los riesgos de seguridad con requisitos o historias de seguridad y las buenas prácticas de codificación segura. Ambas particularidades fomentan el enfoque preventivo del modelo.

4.4.1 Formación en seguridad

La formación en materia de seguridad se considera fundamental dentro del modelo Viewnext-UEx. El objetivo de la formación es enseñar y concienciar a desarrolladores y auditores sobre los perjuicios que pueden generar los fallos de seguridad en el software. De esta forma, se crea conciencia para seguir unas buenas prácticas y garantizar el cumplimiento de las políticas de seguridad definidas por las empresas desarrolladoras.

Existen diversas herramientas vulnerables [68]–[70] que permiten explotar y comprobar los errores más comunes sobre la seguridad del software. Sin embargo, ninguna de ellas plantea ver los escenarios vulnerables y protegidos a la vez.

Por ello y tomando como base las herramientas anteriores, se construye un nuevo entorno de entrenamiento innovador y particularizado [10] que replica algunas de las vulnerabilidades más explotadas en la actualidad [62]. Como se muestra en la Figura 4.2 se visualizan simultáneamente dos escenarios diferentes: uno vulnerable y otro seguro. Además, la herramienta proporciona una comparativa entre el código de ambos escenarios. De esta forma, los usuarios visualizan los errores y su resolución de manera directa.

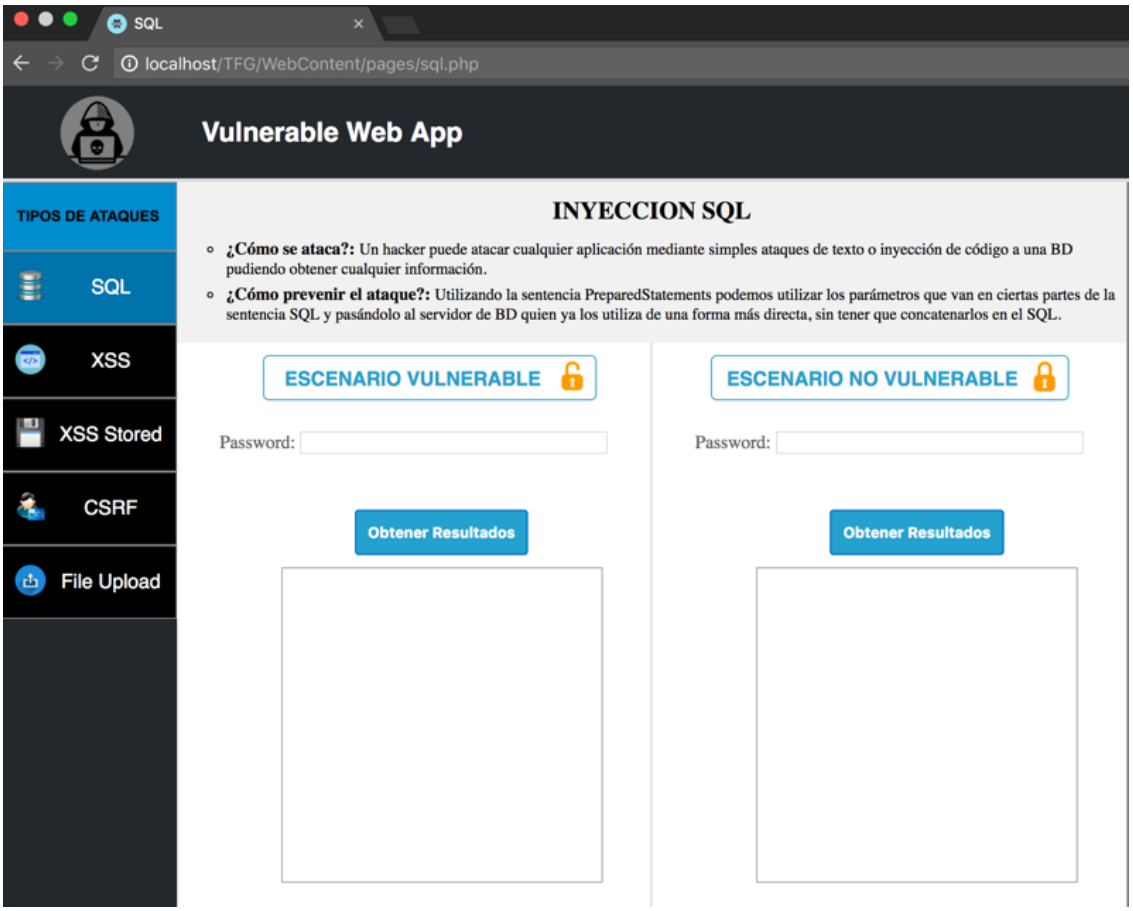


Figura 4.2. Herramienta de entrenamiento para el desarrollo de software seguro. *Fuente: propia.*

Por otro lado, se plantean dos tipos de formaciones con una suma total de 40 horas. La primera pensada para analistas y diseñadores, centrada en la toma de requisitos de seguridad, modelado de amenazas y los patrones de diseño seguro. La segunda, pensada para desarrolladores sobre buenas prácticas de codificación segura.

De esta forma, el modelo Viewnext-UEx suple las carencias existentes de forma generalizada en materia de seguridad. Generalmente, la enseñanza no especializada antepone competencias del software relativas a funcionalidad, usabilidad y escalabilidad, frente a competencias relativas a la seguridad.

4.4.2 Trazabilidad en la metodología de desarrollo

La seña de identidad más reconocida del modelo Viewnext-UEx es la realización de una correcta trazabilidad entre los riesgos y requisitos de seguridad, las amenazas detectadas y los patrones de diseño, para convertir toda la información del proyecto de software en buenas prácticas de codificación segura [11].

Los requisitos de seguridad se extraen, mediante la resolución de un cuestionario automatizado [12], de los aspectos de seguridad que se deducen de los requisitos funcionales de un proyecto de software. Un ejemplo puede ser que la creación del proceso de *login* de una aplicación lleve ligado como requisito de seguridad la protección y control ante intentos automatizados, volver a autenticarse ante operaciones sensibles o la utilización de *captchas*.

El conocimiento generado en la obtención de requisitos (activos, puntos de entradas y entidades externas) permiten modelar posibles amenazas relacionadas con el requisito y categorizarlas según la metodología STRIDE [71]. De esta forma el proceso de *login* de una aplicación es clasificada como una amenaza de la categoría de suplantación de identidad (*Spoofing*). Entre las amenazas destacan los ataques automatizados que comprueban aleatoriamente el usuario y contraseña, el robo de la sesión o de las credenciales del usuario o el intento de modificación del método de autenticación. Así, las comprobaciones aconsejadas consisten en verificar que se controla la autenticación y el acceso en todas las páginas de la aplicación o el envío de credenciales cifrado y su validación en el servidor.

La trazabilidad permite que los requisitos de seguridad ayuden a modelar las amenazas y se conviertan en un libro blanco de buenas prácticas para la codificación segura. Estas buenas prácticas son recomendaciones técnicas a nivel de código, es decir, de cómo realizar la implementación por parte del desarrollador. Siguiendo el ejemplo del *login*, algunas de estas consideraciones a tener en cuenta son la construcción, a través de una política, de los requerimientos de complejidad de la contraseña. Es decir, obligar a introducir contraseñas robustas con combinaciones de caracteres numéricos/alfanuméricos y/o caracteres especiales, impidiendo caracteres considerados peligrosos como: < > " ' % ( ) & + \ / \ ' \ ".

De esta forma, la trazabilidad y su enfoque preventivo evitan que el diseño e implementación de software se oriente únicamente a la funcionalidad. A su vez, permite anticiparse al fallo del software, ya que no se pierde la referencia de los primeros requisitos de seguridad desde que se detectan, pasando por el proceso de revisión de codificación, y testing, hasta la actividad de validación de salidas encargada de la última verificación del software antes de su entrega al cliente.

En la Tabla 4.16 se recoge un ejemplo simplificado para la trazabilidad de un sistema de *login*. En ella, se observa que para el requisito de seguridad seleccionado existen varias amenazas que evitar y que, a su vez, se cuenta con diferentes buenas prácticas de codificación segura para solventarlas.

Tabla 4.16. Ejemplo de trazabilidad en la implementación de un sistema de *login*. *Fuente: propia*

	Implementación de sistema de *login*
Requisito de seguridad	Implementar medidas y consideraciones de protección de autenticación, ante ataques automatizados.
Amenazas	Categoría -> Suplantación de identidad. ▪ Ataque masivo automatizado. ▪ Robo de credenciales por envío sin cifrar. ▪ Robo de sesión por no controlar la caducidad. ▪ Cambio en el método HTTPS de autenticación.
Buenas prácticas de codificación	Consideraciones en la implementación: ▪ Utilización de Captchas. ▪ Controlar un máximo de intentos fallidos (aconsejable 5). ▪ Bloquear cuenta temporalmente (determinar el periodo, evitando la denegación de servicio a los usuarios). ▪ Bloqueo rango de IP intento masivos en poco tiempo. ▪ Registrar los intentos fallidos en un *log*. ▪ Solicitud parcial de la contraseña para evitar k*eystroke logger*. Por ejemplo: "Dígitos primero, tercero y sexto de la contraseña". ▪ Autocompletar deshabilitado. ▪ Doble autenticación en operaciones sensibles. ▪ Política de renovación de credenciales.

4.5 Comparativa del modelo Viewnext-UEx con otros modelos

La Tabla 4.17 recoge una comparativa entre los modelos analizados y el nuevo modelo Viewnext-UEx. De esta forma complementamos la Tabla 3.1, en la que únicamente aparecían el resto de modelos, con la inclusión del modelo propuesto en esta Tesis Doctoral. En esta tabla, las actividades de cada modelo se disponen clasificadas según la fase del ciclo de vida donde se ejecutan. Recordar que todos los modelos que aparecen en la Tabla 4.17 se aplican en entornos industriales.

El modelo Viewnext-UEx se ha diseñado uniendo un subconjunto de actividades coincidentes en otros marcos de trabajo estudiados y se complementado con tres prácticas nuevas que buscan suplir carencias detectadas en los modelos estudiados. Se puede observar cómo el modelo propuesto apunta a ser uno de los más completos, al ser uno de los pocos que integran actividades en todas las fases del ciclo de vida del software. No obstante, serán los apartados de resultados y discusión los que determinen la eficacia del nuevo modelo.

Tabla 4.17. Comparativa de actividades de seguridad de los modelos, incluyendo el modelo Viewnext-UEx. *Fuente: propia*

Fase/Modelo	Microsoft SDL/ASDL	OSSA	CLASP	TSP-Secure	SAMM	BSIMM	Viewnext-UEx
Políticas			• Política de seguridad global • Recursos y límites de confianza • Roles de usuario y capacidades • Especificar el entorno operativo		• Política y cumplimiento	• Cumplimiento y política	• Estrategia y orientación • Definición de riesgos
Formación	• Formación	• Formación	• Programa de sensibilización sobre seguridad institucional	• Formación	• Educación y orientación	• Formación	• Formación
Análisis	• Análisis de riesgos de seguridad y privacidad • Requisitos de seguridad	• Definición de riesgos • Requisitos de seguridad	• Análisis superficie de ataques y riesgos • Requisitos de seguridad	• Análisis de riesgos • Requisitos de seguridad	• Evaluación de amenazas • Requisitos de seguridad • Arquitectura de seguridad	• Requisitos y estándares • Análisis de las arquitecturas	• Validación de requisitos • Modelos de amenazas
Diseño	• Modelos de riesgos • Analizar superficie de ataques • Requisitos de diseño	• Modelado de amenazas • Revisión del diseño • Caso de abuso	• Modelado de amenazas • Guía y principios del diseño seguro • Diagrama de clases de seguridad • Casos de abuso	• Modelado de amenazas • Diseño seguro • Casos de abuso	• Modelado de amenazas • Revisión del diseño	• Modelos de amenazas • Características y diseño de seguridad	• Revisión del diseño
Implementación	• Usar herramientas aprobadas • Prohibir funciones no seguras	• Reglas y principios de codificación segura • Revisión de funciones complejas • Revisión de código	• Configuración de seguridad en BD • Análisis estático	• Guía y estándares de desarrollo seguro • Análisis estático	• Revisión de código	• Revisión de código • Entorno del software	• Revisión del desarrollo
Pruebas	• Análisis dinámico • Fuzz testing	• Testing de seguridad	• Testing de seguridad	• Testing de seguridad • Fuzz testing	• Testing de seguridad	• Testing de seguridad • Pruebas de penetración	• Testing de Seguridad
Pre-Release	• Revisión de seguridad final • Lanzamiento o archivado	• Validación del cumplimiento	• Validación de seguridad				• Validación de salidas
Post-Release	• Plan de respuesta a incidentes	• Corrección de vulnerabilidades	• Dirección de problemas de seguridad reportados	• Lista de vulnerabilidades con sus posibles mitigaciones	• Gestión de vulnerabilidades • Securización del entorno • Habilitación operativa	• Gestión de configuración y de vulnerabilidades	• Observatorio de seguridad • Repositorio de vulnerabilidades • Plan de respuesta a incidentes
Métricas	• Umbrales de calidad y límites de errores	• Criterios de seguridad que deberá cumplir el producto	• Monitorizar métricas de seguridad	• Medir métricas de seguridad	• Estrategia y métricas	• Estrategia y métricas	• Evaluación y métricas • Estado del proyecto
Uso empresarial	X	X	X	X	X	X	X

4.5 Síntesis del capítulo

Este capítulo ha presentado el modelo Viewnext-UEx compuesto por 4 áreas de desarrollo y 14 actividades de seguridad. Se ha expuesto el conjunto de actividades de seguridad del modelo recogiendo los detalles más importantes de cada una y detallando cómo algunas de las actividades específicas en el desarrollo del software tienen un carácter preventivo y otras un enfoque reactivo. El objetivo de las actividades reactivas es retroalimentar con su información a las de sesgo preventivo, produciendo la mejora continua en el modelo, sus actividades y la seguridad final del software. En la Tabla 4.18 se aprecian las actividades y el enfoque que se otorga a cada una de ellas.

Tabla 4.18. Enfoque de las actividades de seguridad del área Metodología SDL. *Fuente: propia*

Área de desarrollo	Actividad de Seguridad	Enfoque
Metodología SDL	Validación de requisitos	Preventivo
	Modelado de amenazas	Preventivo
	Revisión de Diseño	Reactivo
	Revisión de Desarrollo	Reactivo
	Testing de Seguridad	Reactivo
	Validación de salidas	Reactivo

Seguidamente, se han expuesto las características que describen el modelo Viewnext-UEx, donde el enfoque preventivo y retroalimentado se consigue al transferir la información entre las actividades. La integración, reutilización y flexibilidad es debida a que el modelo se ha fundamentado en las 10 actividades más importantes extraídas de otros modelos de seguridad, una actividad (estrategia y orientación) no común en los modelos estudiados pero considerada importante y tres actividades de nueva creación. Las actividades nuevas son el repositorio de vulnerabilidades, estado del proyecto y observatorio de seguridad, creadas con el objetivo de suplir las carencias existentes en otros modelos y de anticipar la detección de vulnerabilidades en el software.

El capítulo ha finalizado con la exposición de los puntos fuertes del modelo Viewnext-UEx, mostrándose en comparativa con el resto de modelos.

"Para avanzar, basta con dar el primer paso."
Chase Neill

Capítulo 5

5. Metodología de implantación empresarial

Este capítulo tiene como objetivo exponer el procedimiento seguido para incorporar el modelo Viewnext-UEx en cualquier compañía de desarrollo. Se comienza describiendo una serie de aspectos relativos a los proyectos de software que se consideran importantes y que afectan al proceso de implantación del modelo. Después, se describen las fases diseñadas para la integración y adaptación de las actividades de seguridad del modelo Viewnext-UEx. El capítulo termina con una síntesis de lo expuesto.

5.1 Características de un proyecto de software

La implantación y adaptación empresarial de un nuevo marco de trabajo para el desarrollo de software es una tarea compleja. Los cambios en este tipo de procesos afectan al entorno de desarrollo de software e impactan en las planificaciones de los proyectos, generando normalmente consecuencias negativas. Es importante también tener en cuenta cómo abordar estos cambios con el equipo de trabajo.

Por ello, antes de diseñar este proceso de transformación laboral es preciso conocer cuáles son los aspectos que, en mayor medida, afectan a la evolución de un proyecto de software. A continuación, se describen las características relacionadas con la tipología de

proyectos, paradigma de desarrollo y fase de comienzo en el proyecto. Estas características definen la metodología de trabajo de los desarrollos de la empresa Viewnext S.A., primera compañía en la se ha llevado a cabo la implantación del modelo presentado en esta Tesis Doctoral.

5.1.1 Tipología de proyectos de software

Es importante conocer los tipos de proyectos existentes, ya que la planificación de las actividades de seguridad del modelo Viewnext-UEx puede sufrir ajustes o modificaciones dependiendo de la tipología de proyecto a la que se vaya a aplicar.

La tipología de un proyecto de software define la forma de trabajo para crear o mantener un producto o servicio. Se definen, de este modo, como proyectos de *construcción* o *mantenimiento* de software.

A continuación, se describen estos dos tipos de proyectos de software en función del objetivo contratado por el cliente:

- **Proyectos de construcción, desarrollo de aplicaciones o nuevos desarrollos**: consiste en la construcción de software a medida en función de los requisitos deseados por el cliente.

- **Mantenimiento de aplicaciones/software**: es el tipo de proyecto conocido como AMS (*Application Management Services*). Son los tipos de proyectos que realizan modificaciones sobre aplicaciones operativas en producción, bien debido al funcionamiento anómalo del sistema (mantenimiento correctivo), o bien por la aparición de nuevos requisitos (mantenimiento evolutivo). Cuando el impacto de los nuevos requisitos es muy alto se convierte en un proyecto del tipo de desarrollo de aplicaciones o nuevos desarrollos.

Como se expone posteriormente, la diferencia fundamental entre ambos tipos de proyectos radica en el enfoque preventivo o reactivo que se da a una de las fases del proceso de implantación diseñado.

5.1.2 Paradigmas de desarrollo

Como se estudiaba en el Capítulo 2 (apartado 2.3 Ciclo de vida del software), los modelos de desarrollo pueden diferenciarse según el momento en el que tienen lugar las actividades a lo largo del proceso de vida del software. De modo que la planificación, organización y coordinación del proceso de construcción del producto final puede ser distinto.

Entre los paradigmas actuales más comunes se encuentran los siguientes:

- **Tradicional**: proceso de construcción de software llevado a cabo de manera secuencial como un conjunto de etapas que se ejecutan una tras otra. En este tipo de metodología el resultado de la fase anterior es el comienzo de la siguiente, siguiendo un flujo en cascada.

- **Ágil**: proceso de desarrollo cuyo enfoque se centra en proporcionar respuestas y desarrollos rápidos que se someten a continuos cambios o modificaciones de mejora. En este tipo de desarrollo el ciclo de vida del software se convierte en cíclico (iteraciones o *sprint*) pasando de manera incremental y temporal (semanal, mensual, etc.) por todas las fases del desarrollo. En la empresa Viewnext S.A., la más extendida es la metodología ágil denominada SCRUM.

La diferencia en los tiempos de liberación del software para cada tipo puede provocar ajustes en la ejecución temporal de las actividades de seguridad del modelo Viewnext-UEx.

5.1.3 Fase de intervención al desarrollo

La política de desarrollo establecida por el cliente provoca que la intervención por parte de las empresas desarrolladoras en los proyectos sea diferente. Por ello, se dividen en dos:

- **Análisis funcional**: el equipo de trabajo realiza por sí mismo la toma de requisitos o historias de usuario, por lo que el modelo Viewnext-UEx se aplica desde la fase inicial del proyecto.
- **Diseño técnico:** el equipo de trabajo recibe por parte del cliente los requisitos o historias de usuario del desarrollo que se debe acometer. Por lo tanto, el modelo Viewnext-UEx se aplica desde el modelado de amenazas.

Como vemos, la intervención del cliente implica la adaptación de las actividades de seguridad y, con ello, el necesario ajuste del modelo propuesto.

5.1.4 Características técnicas

Existen características importantes de un proyecto de software, principalmente de carácter técnico, que se deben conocer para realizar una implantación efectiva del modelo Viewnext-UEx. Estas cuestiones afectan a cómo abordar la seguridad de un proyecto de software y están relacionadas con el tipo de arquitectura de la aplicación, el tipo de acceso a los datos, el manejo de datos sensibles o el uso de herramientas específicas.

La Tabla 5.1 recoge, a modo de formulario, la información relevante de un proyecto de software. Algunas preguntas tienen una respuesta personalizada, dependiendo del proyecto de software, y otras, deben seleccionarse de entre unas opciones pre-establecidas.

Tabla 5.1. Formulario de recogida de información de un proyecto de software. *Fuente: propia*

Pregunta	Respuesta
Usuario tipo	Objetivo general de la aplicación
Nivel colaboración del cliente	Alto Medio Bajo
Tipo arquitectura	Standalone Web Mobile SOA Microservicios Mixta (web/mobile)
Tipo acceso	Internet Intranet Extranet
Manejo de datos sensibles (médicos, personales, económicos, negocio, etc)	Sí / No
Manejo de datos bancarios (tarjetas, cuentas…)	Sí / No
Normativa legal aplicable	Sí / No
Normativa sectorial aplicable	Sí / No
Frecuencia de las entregas (*sprints*, *releases*)	Semanal 2 semanas Mensual Otro
Entorno de desarrollo	Local VDI
Integración continua	Sí / No
Herramienta para control de incurridos	Produce Rational Team Concert (RTC) Otro
Herramienta para gestión de la demanda	Redmine JIRA

	Rational Team Concert (RTC) Otro

La información extraída del formulario anterior permite obtener una visión global de un proyecto de software, lo cual hace más eficaz la implantación del modelo Viewnext-UEx.

5.1.5 Resumen de características de proyectos de software

La Tabla 5.2 resume a grandes rasgos los aspectos, anteriormente vistos, que de manera común describen la mayor parte de los proyectos de software de una empresa desarrolladora. La combinación de estas características y sus particularidades conforma el proyecto de software.

Tabla 5.2. Resumen de características de un proyecto de software. *Fuente: propia*

Tipología	Metodología	Fase de intervención
Nuevos desarrollos	Tradicional	Análisis funcional
Mantenimiento de aplicaciones	Ágil	Diseño técnico

5.2 Fases de implantación

La Figura 5.1 refleja el proceso de implantación del modelo Viewnext-UEx [15]. Como se observa, sigue una metodología basada en 4 fases, que tienen en cuenta la formación en materia de seguridad, la preparación de un ecosistema seguro, una evaluación previa de la seguridad del proyecto y el proceso de integración de las actividades de seguridad del modelo.

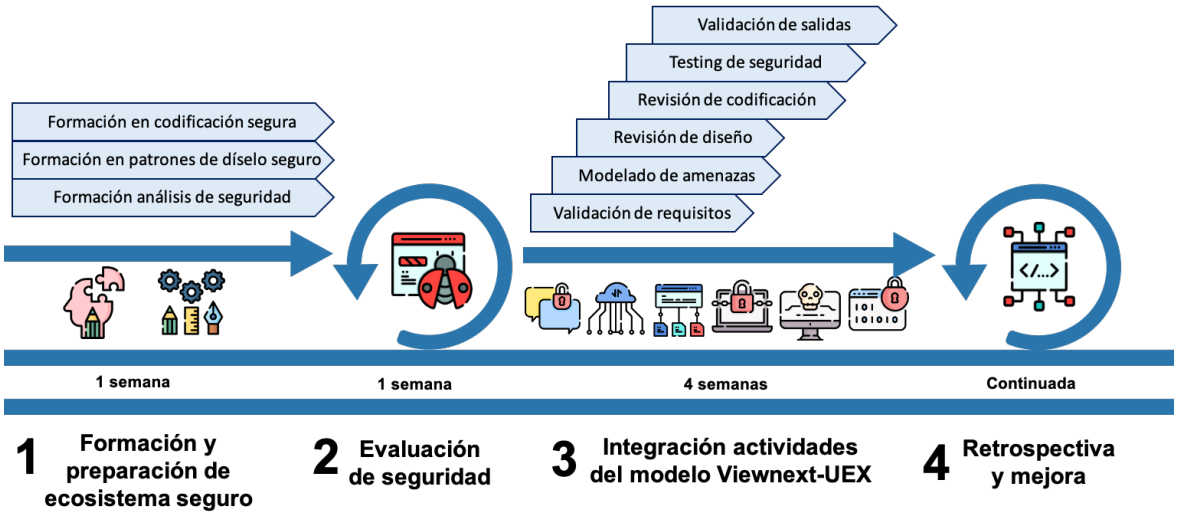


Figura 5.1 Fases de implantación del modelo Viewnext-UEx. *Fuente: propia.*

5.2.1 Fase de formación y preparación del ecosistema seguro

Conocidas las características técnicas de un proyecto de software se aplica la primera fase. Esta fase tiene dos objetivos fundamentales, que para su optimización pueden realizarse de forma paralela. El primero es formar en materia de seguridad a todo el personal involucrado y el segundo construir un ecosistema de herramientas de seguridad que sirvan de apoyo durante la aplicación del modelo Viewnext-UEx.

Se estima una duración de una semana desde que se inicia el proceso de implantación de la metodología.

5.2.1.1 Formación en materia de seguridad

La formación en materia de seguridad mantiene el enfoque preventivo de anticipar la detección de vulnerabilidades desde las fases más tempranas. Se encuentra avalada por la actividad de formación que puede visualizarse en mayor detalle en el apartado 4.2.2.2 Formación. La Tabla 5.3 resume las formaciones consideradas esenciales, el número de horas y la especialización del personal involucrado.

Tabla 5.3. Formaciones en materia de seguridad, número de horas y personal. *Fuente: propia*

Formación	Horas	Personal
Formación sobre análisis de seguridad	20	Analistas Diseñadores
Formación en patrones de diseño seguro		
Formación en buenas prácticas de codificación segura [10]	20	Desarrolladores Testeadores

5.2.1.2 Ecosistema de seguridad

Para apoyar las actividades del modelo Viewnext-UEx se considera fundamental seleccionar un conjunto de herramientas que den soporte y ayuden a mejorar la seguridad de los proyectos de software. Por ello se realiza la evaluación y selección de un ecosistema de herramientas para un enfoque preventivo y continuo [16] cuyo objetivo es abarcar el mayor número de tecnologías y cubrir la mayoría de las fases del ciclo de vida. En la Tabla 5.4 se presenta una selección de las herramientas en función del objetivo dentro del modelo Viewnext-UEx, la fase de ejecución y la actividad de seguridad en la que se aplica:

Tabla 5.4. Herramientas por objetivo, fase de ejecución y actividad. *Fuente: propia*

Objetivo	Fase de ejecución	Actividad de seguridad del Modelo Viewnext-UEx	Herramientas
Integración continua	Todo SDLC		Jenkins IBM Cloud Gitlab
Modelado de amenazas	Requisitos Diseño	Validación de requisitos Modelado de amenazas Revisión de diseño	Microsoft IriusRisk
Análisis estático del código fuente	Codificación	Revisión de desarrollo	SonarQube PMD Kiuwan Checkmarx FindBugs
Pruebas unitarias de seguridad			Junit xUnit.net SoapUI
Pruebas de integración de seguridad			Arquillian SoapUI
Test de penetración	Pruebas	Testing de seguridad	AppScan BurpSuite OWASP ZAP Hdiv modSecurity

5.2.2 Fase de evaluación de seguridad

La segunda fase de implantación del modelo Viewnext-UEx es una evaluación de seguridad del proyecto de software. Esta fase tiene mayor sentido en proyectos con un largo recorrido y que no han realizado auditorías de seguridad. Se busca detectar y resolver los fallos de seguridad existentes midiendo el coste de la resolución de los fallos en la fase de post-producción.

La Figura 5.2 describe el enfoque reactivo e invertido de la evaluación de seguridad. En esta fase se aplican medidas reactivas como el análisis de código y las auditorías de seguridad, bien completas, o bien de partes concretas del proyecto denominadas catas de seguridad.

Se estima una duración de una semana desde que se inicia el proceso de evaluación de la seguridad.

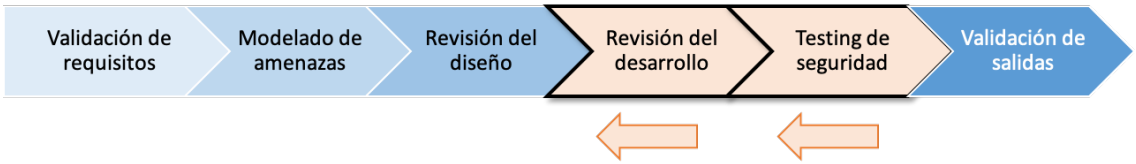


Figura 5.2. Enfoque reactivo e invertido del proceso de evaluación. *Fuente: propia.*

5.2.3 Fase de integración del modelo

La tercera fase de la metodología de implantación del modelo Viewnext-UEx es la encargada de integrar las actividades de seguridad en los procesos de construcción del software. Se comienza de manera incremental desde las fases más tempranas del desarrollo hasta las fases finales. Las primeras acciones son la revisión de los riesgos de negocio, una verificación de los requisitos de seguridad y un análisis de la superficie de ataque de la aplicación.

Se estima una duración de un mes desde que se inicia el proceso de integración.

5.2.4 Retrospectiva y modelo de mejora continua

La cuarta fase de la metodología de implantación del modelo Viewnext-UEx se encarga de tomar como referencia las lecciones aprendidas, los puntos de mejora y las nuevas adaptaciones realizadas por el equipo de trabajo de los proyectos de software.

Esta fase tiene una alta importancia ya que analiza las métricas obtenidas sobre los indicadores de seguridad y costes temporales de la aplicación de las actividades de seguridad en el proceso de desarrollo. De esta forma, se puede realizar una comparativa de seguridad y coste temporal entre un desarrollo realizado con una metodología tradicional y el nuevo modelo propuesto, consiguiendo uno de los objetivos marcados en esta Tesis Doctoral.

5.3 Indicadores de rendimiento

Para evaluar la eficacia del modelo Viewnext-UEx se proponen distintas métricas que miden, por un lado, la seguridad del software desarrollado y, por otro, el rendimiento temporal del proceso seguido en función de las fases y la aplicación de las actividades de seguridad.

5.3.1 Indicadores de la seguridad del desarrollo

El nivel de seguridad del software se evalúa mediante el número de vulnerabilidades detectadas, el tipo y la criticidad de las mismas. El número de vulnerabilidades corresponde a un valor cuantitativo. Para el tipo de vulnerabilidades se pueden utilizar distintas clasificaciones de estándares de referencia como [64]–[66], [72]. Para medir la criticidad se utiliza el estándar *Common Vulnerability Score System* [72] que presenta cinco niveles ordenados de menor a mayor criticidad: informativa, baja, media, alta y crítica. Además, para facilitar su posterior resolución, el tipo de vulnerabilidades se clasifica en dos categorías: arquitectura y desarrollo. Las de arquitectura se encuentran relacionadas con el entorno de la aplicación (*web server, application server, database, frameworks, custom code,* etc.) y las de desarrollo con aspectos más puros de la codificación del software (*injections, XSS, broken authentication, sensitive data exposure,* etc.).

La Tabla 5.5 recoge de forma resumida las métricas para clasificar las vulnerabilidades detectadas en las distintas actividades de seguridad del software.

Tabla 5.5. Resumen de las métricas relativas a la seguridad del software. *Fuente: propia*

Tipo de vulnerabilidad	Categoría interna	Criticidad	
Validación de entradas	Arquitectura	Informativa	[0]
Gestión de sesiones	Desarrollo	Baja	[0.1-3.9]
Política de control de acceso		Media	[4.0-6.9]
Autorización		Alta	[7.0-8.9]
Sistemas y arquitectura		Crítica	[9-10]
Gestión de errores			
Configuración			
Protección de datos			
Etc.			

5.3.2 Indicadores de productividad del desarrollo

Este apartado define y describe una serie de métricas temporales encargadas de evaluar la efectividad del proceso de desarrollo utilizando el modelo Viewnext-UEx. Las tareas del proceso de desarrollo se agrupan en las fases comunes del ciclo de vida del software. Estas fases son las siguientes:

- **Gestión**: tiempo dedicado a gestionar la dirección de los proyectos de software, marcando las directrices globales del proceso de desarrollo. Contempla el análisis y control de las tareas, las reuniones de coordinación entre los miembros del equipo y la entrega del software al cliente. Las actividades del modelo Viewnext-UEx que se incluyen en este apartado son **estrategia y orientación** (directrices globales de la seguridad del proyecto), **validación de salidas** (comprobación de la seguridad del software) y **evaluación y métricas** (análisis y revisión de las métricas del modelo Viewnext-UEx).

- **Análisis y diseño**: tiempo dedicado a las tareas de análisis de requisitos del software y diseño de la arquitectura, de la estructura de datos y representación de la interfaz y algoritmos. Las actividades del modelo Viewnext-UEx que se incluyen en esta fase son **definición de riesgos** (identificación de los riesgos de negocio y su impacto), **validación de requisitos** (verificación de la seguridad de los requisitos obtenidos), **modelado de amenazas** (identificación de ataques desde la visión del atacante) y **revisión del diseño** (comprobación del cumplimiento de los patrones de seguridad del diseño).

- **Codificación**: tiempo que se dedica al proceso de transformar el diseño en código legible por un sistema informático. El modelo Viewnext-UEx incluye en esta fase la actividad de **revisión del desarrollo**, encargada de realizar un análisis de la seguridad del código programado.

- **Pruebas**: tiempo dedicado a comprobar que el código desarrollado funciona y se comporta según los requisitos establecidos en el análisis por el cliente. El modelo Viewnext-UEx incluye la actividad de **testing de seguridad**, que realiza los test de penetración simulando ataques de seguridad reales que comprueben la seguridad del software que ha sido desarrollado.

- **Evaluación**: tiempo dedicado a la realización de una auditoría de seguridad completa o parcial que evalúe el estado de la seguridad de un proyecto de software. Esta fase es coincidente con lo explicado en el proceso de implantación y que se detalla en el apartado 5.2.2 Fase de evaluación de seguridad.

- **Resolución de fallos**: tiempo dedicado a solucionar las vulnerabilidades o los fallos de seguridad detectados en la fase de evaluación anterior.

- **Desarrollo global del módulo:** corresponde a la suma global de las fases explicadas con anterioridad. Recoge la suma de los tiempos realizados para el proceso de desarrollo de un proyecto de software. En el caso de utilizar el nuevo modelo propuesto, contempla también el tiempo dedicado a las actividades de seguridad.

La Tabla 5.6 recoge las fases tomadas para medir el tiempo de realización de un proyecto de software, incluyendo asimismo las actividades de seguridad del modelo Viewnext-UEx. La unidad de medida utilizada para contabilizar la ejecución de cada fase es la hora.

Tabla 5.6. Métricas para medir el tiempo de ejecución de un proyecto de software. *Fuente: propia*

Unidad	Fase	Actividades modelo Viewnext-UEx
Horas	Gestión	Estrategia y orientación Validación de salidas
	Análisis y diseño	Definición de riesgos Validación de requisitos Modelado de amenazas Revisión del diseño
	Codificación	Revisión del desarrollo
	Pruebas	Testing de seguridad
	Evaluación	
	Resolución de fallos de seguridad	
	Desarrollo total del módulo	

5.4 Síntesis del capítulo

Este capítulo ha presentado la metodología de implantación empresarial del modelo Viewnext-UEx. Dicha metodología ha tenido en cuenta las características de los proyectos que más afectan a la planificación de los proyectos. Estas son la tipología de los proyectos de software, el paradigma metodológico, la fase de intervención al desarrollo por parte de la empresa y las características técnicas particulares de cada proyecto.

De igual modo se han recogido las características técnicas de los proyectos, que ayudan a entender el funcionamiento actual del equipo de trabajo.

Posteriormente, se ha descrito la metodología de implantación empresarial del modelo Viewnext-UEx y las cuatro fases que la componen. También, se detallan los indicadores de rendimiento de seguridad y de costes temporales que se van a usar para conocer la eficacia

que aporta el modelo presentado. En las métricas temporales se distinguen las fases del ciclo de vida y qué actividades de seguridad que componen el modelo Viewnext-UEx alberga cada fase.

Bloque III

Resultados y discusión

"La ciencia es la progresiva aproximación del hombre al mundo real".
Max Planck

Capítulo 6

6. Aplicación experimental del modelo Viewnext-UEx

Este capítulo presenta la implementación del modelo Viewnext-UEx en un proyecto de software real y sus respectivos resultados. El objetivo del experimento es evaluar el desarrollo de un módulo de software utilizando la metodología reactiva tradicional, en comparación con otro módulo de software similar utilizando el modelo Viewnext-UEx. Para ello, se describe el diseño del caso de estudio o experimento. En primer lugar, se conocen las características específicas del contexto del experimento (materiales), coincidentes en ambos escenarios (métodos). Seguidamente, se dibujan las fases de los dos escenarios, uno reactivo (método tradicional) y otro preventivo (método propuesto). Tras la ejecución del caso de estudio se presentan los resultados de las actividades de seguridad y el coste temporal para ambos escenarios. Por último, se discuten y comparan los resultados entre los distintos escenarios. Al final, se presenta una breve síntesis con lo expuesto en el capítulo.

6.1 Diseño experimental

Esta tesis plantea un caso de estudio (un experimento comercial) para medir la efectividad del modelo Viewnext-UEx que sigue el esquema de la Figura 6.1. El caso de estudio se lleva a cabo en un proyecto de software real de la empresa de desarrollo de software Viewnext S.A. El experimento se diseña con dos escenarios diferentes, uno reactivo y otro preventivo, siguiendo este último el modelo presentado en esta Tesis Doctoral. El objetivo del experimento es comparar los resultados de seguridad y los costes temporales durante el proceso de desarrollo en ambos escenarios, como se recoge en [17].

Cada escenario desarrolla un módulo de software dentro del mismo proyecto. Los escenarios comparten el mismo equipo de trabajo, mismo *framework* arquitectónico y desarrollan módulos con similar complejidad funcional. Primero se desarrolla el escenario reactivo que sigue el método tradicional y, tras su finalización, el mismo equipo de trabajo codifica el escenario preventivo que aplica el modelo Viewnext-UEx. No se cruza información entre los escenarios para evitar que los resultados se vean influenciados.

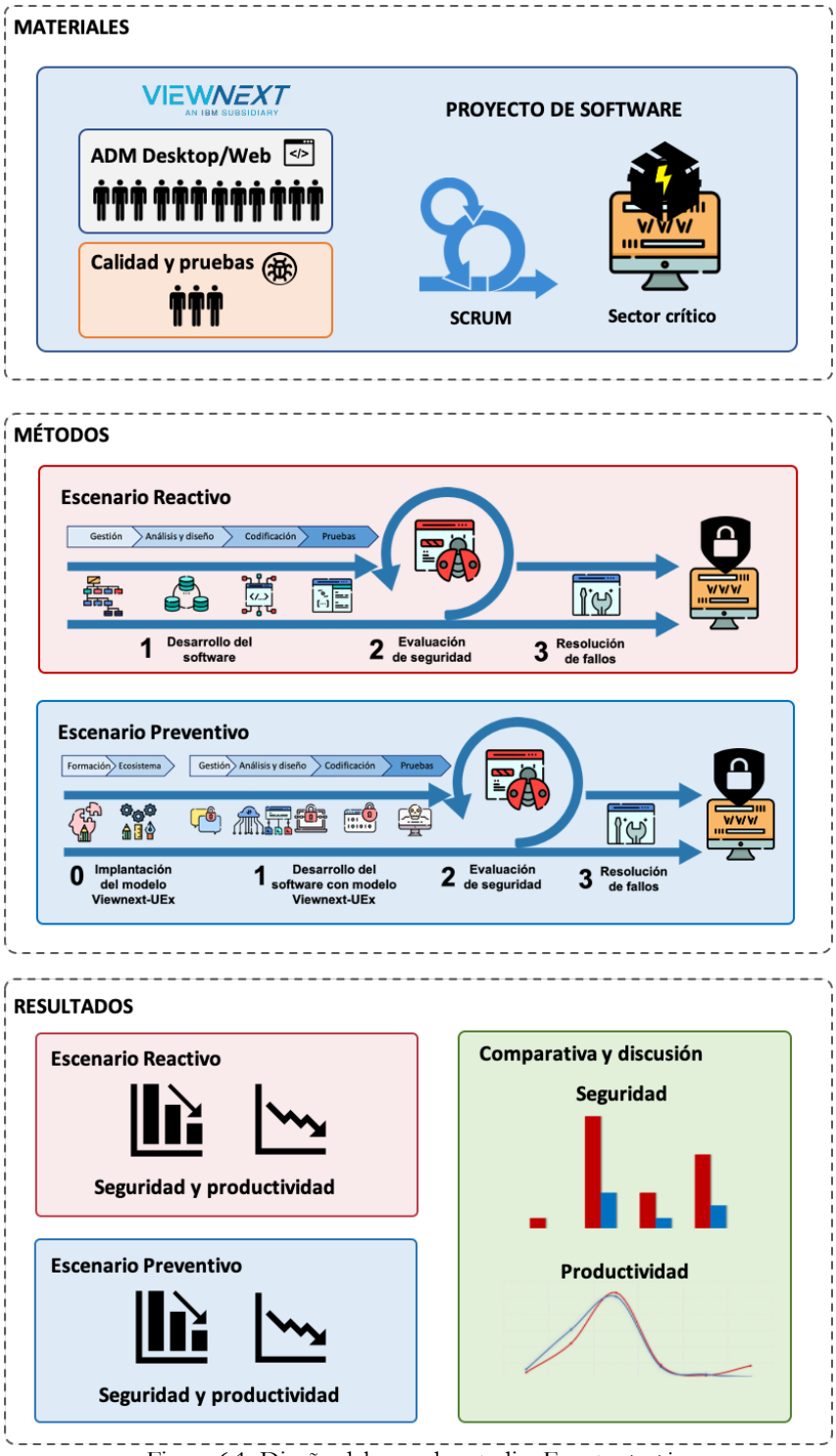


Figura 6.1. Diseño del caso de estudio. Fuente: *propia.*

6.2 Materiales

Los materiales involucrados en el caso de estudio se detallan en los siguientes apartados. Por un lado, se explica el contexto de la empresa de software que colabora en el caso de estudio. Por otro lado, se especifican las características del proyecto de software seleccionado para el experimento y de la aplicación a desarrollar. El software pertenece a una empresa del sector eléctrico, un sector crítico debido a que tanto el riesgo de ataques con éxito como las consecuencias que pueden provocar los fallos de seguridad son consideradas de alta gravedad.

Las vulnerabilidades encontradas en ambos escenarios y que se exponen en el caso de estudio han sido resueltas antes de poner el software a disposición de los usuarios sin generar ningún tipo de riesgo para la empresa. El interés y la colaboración de la empresa ha sido imprescindible para realizar con éxito el diseño del experimento estipulado.

6.2.1 Empresa Viewnext

El caso de estudio se lleva a cabo en Viewnext S.A., empresa de Servicios de Tecnología de la Información del grupo IBM España. Esta factoría de software tiene en el año 2020 un equipo de más de 4.500 profesionales. En la Figura 6.2 se visualiza la cobertura de oficinas y centros de innovación tecnológica que tiene en España y Portugal.

Figura 6.2. Centros de Innovación y oficinas de Viewnext S.A.. *Fuente: www.viewnext.com*

Viewnext S.A. se organiza en prácticas especializadas que prestan servicios desde cualquiera de sus centros de innovación. Estás prácticas son secciones departamentales divididas en función de una tecnología o del propósito de su ejecución. Para contextualizar el experimento o caso de estudio es importante destacar que participan recursos humanos o personales de dos prácticas distintas:

- **Práctica ADM Desktop/Web** encargada del desarrollo y mantenimiento de las aplicaciones. Formada por analistas de software y de negocio, diseñadores y desarrolladores. Ejecutan el conjunto de las actividades de seguridad englobadas en el área de desarrollo de Metodología SDL, salvo el testing de seguridad.

- **Práctica de Calidad y Pruebas** es una práctica transversal a todos los proyectos de software. Esta práctica se encarga de realizar las pruebas o el testing, tanto funcional como de seguridad. También gestiona la calidad integral de las aplicaciones que mantienen o desarrollan otras prácticas.

- Los módulos de software de los escenarios planteados son diseñados y codificados por la práctica ADM Desktop/Web. Las auditorías y evaluaciones de seguridad son realizadas por la práctica de Calidad y Pruebas. Ambas prácticas utilizan el mismo equipo de trabajo para realizar el experimento.

6.2.2 Características del proyecto de software y de la aplicación

El caso de estudio diseñado se realiza en el marco de un proyecto de software real. La utilización de un entorno real garantiza la transferencia de conocimiento, uno de los puntos fuertes de esta Tesis Doctoral.

El proyecto de software se enmarca dentro de la tipología de nuevos desarrollos, correspondiendo al desarrollo proactivo de una aplicación de gestión interna para controlar electricidad. Utiliza la metodología ágil SCRUM con ciclos de liberación mensuales y cuya intervención en los desarrollos arranca desde la fase de toma de requisitos y análisis funcional. El equipo de desarrollo está formado por 15 miembros que pertenecen a la práctica de ADM Desktop/Web. Es un equipo multidisciplinar que hace uso de herramientas para imputar los tiempos de desarrollo, para gestionar la demanda y para el análisis de la seguridad y calidad del código. Sin embargo, el testing es realizado por un miembro externo al equipo del

proyecto. Este miembro pertenece a la práctica de Calidad y Pruebas. La Tabla 6.1 recoge la descripción del proyecto de software.

Tabla 6.1. Descripción del proyecto de software del caso de estudio. *Fuente: propia*

Descripción del proyecto de software	
Sector de negocio	Eléctrico
Nivel de criticidad	Crítica
Tipo de proyecto	Nuevos desarrollos
Ciclo de vida del software	Agile (SCRUM)
Frecuencia de entregas	Mensual
Fase de intervención	Análisis funcional
Tamaño del equipo	15 personas

La Tabla 6.2 presenta las características técnicas de la aplicación a desarrollar en el caso de estudio. Estas características se obtienen del formulario detallado en el apartado Como vemos, la intervención del cliente implica la adaptación de las actividades de seguridad y, con ello, el necesario ajuste del modelo propuesto.

5.1.4 Características técnicas.

Tabla 6.2. Características del proyecto de software del caso de estudio. *Fuente: propia*

Características técnicas de la aplicación	
Usuario tipo	Gestión interna
Tipo desarrollo	Nuevo desarrollo
Nivel colaboración del cliente	Alto
Tipo arquitectura	Web
Tipo acceso	Intranet
Manejo de datos sensibles (médicos, personales, económicos, etc.)	Sí
Manejo de datos bancarios	Sí
Normativa legal aplicable	Sí
Normativa sectorial aplicable	Sí
Ciclo de vida utilizado	Agile-SCRUM
Frecuencia de las entregas (*sprints*, *releases*)	Mensual

Entorno de desarrollo	Local
Especificación de requisitos	Sí
Herramienta integración continúa	Jenkins
Herramienta de tiempos	Produce (IBM)
Herramienta demanda	Redmine
Herramienta análisis de código	SonarQube
Herramienta test de penetración	AppScan (IBM)

La Figura 6.3 describe a alto nivel la arquitectura de la aplicación utilizada en el caso de estudio. La arquitectura pertenece a una aplicación web modelada con la aplicación Microsoft Threat Modeling Tool [40], donde se diferencian los activos de la aplicación, los almacenes de datos, las dependencias externas, los puntos de entrada, los tipos de conexión entre los procesos de la aplicación y las zonas de confianza.

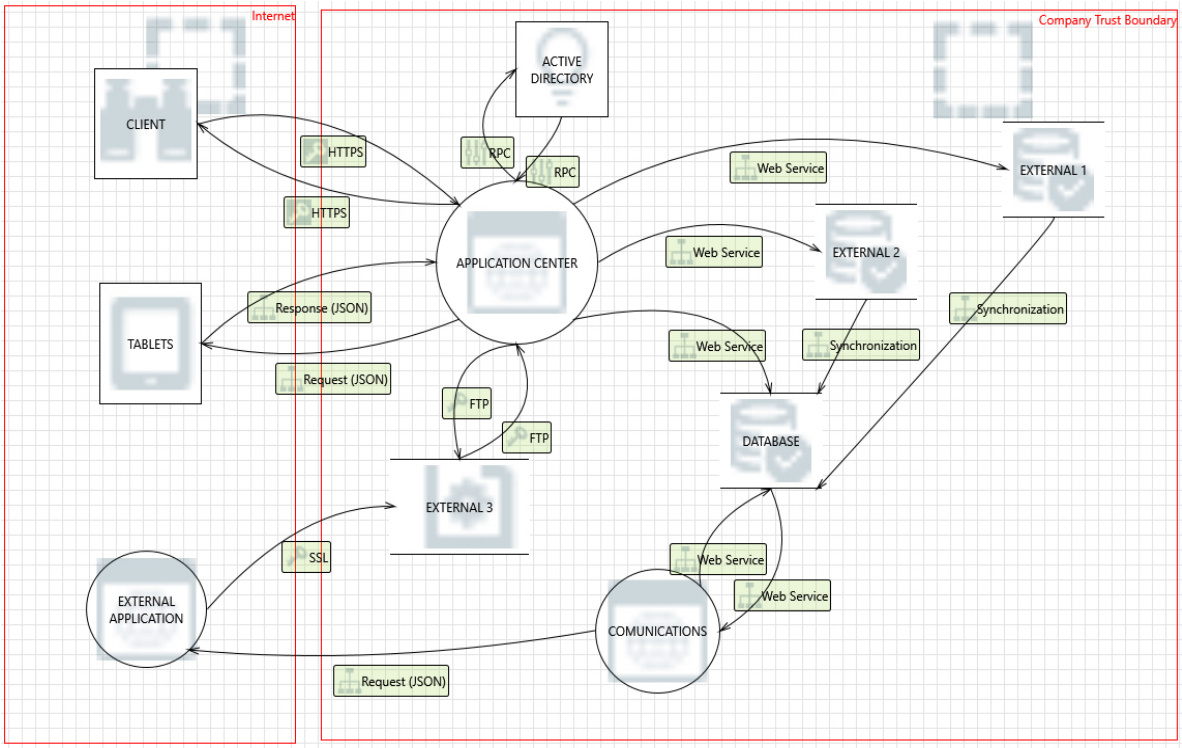


Figura 6.3. Arquitectura de la aplicación del caso de estudio. *Fuente: propia.*

6.3 Métodos

Para comparar los resultados de las actividades de seguridad y los costes temporales de los dos escenarios considerados, en este apartado se presentan con detalle las dos metodologías, con sus métricas de seguridad y tiempo a evaluar en cada escenario. Como se ha comentado, los escenarios se desarrollan dentro del mismo proyecto de software y son ejecutados secuencialmente por el mismo equipo de trabajo.

6.3.1 Escenario reactivo

Este escenario aplica una metodología de desarrollo del software reactiva. El equipo de trabajo construye un módulo (conjunto de requisitos o funcionalidades con una finalidad interpuesta por el cliente) aplicando la metodología habitual dentro de la empresa. Terminada la implementación se realiza una auditoría de seguridad con objeto de identificar posibles vulnerabilidades en el módulo. Finalmente, el equipo de trabajo corrige y resuelve los fallos de seguridad encontrados en la auditoría. En este escenario, el software es desarrollado teniendo en cuenta la seguridad en las fases finales del ciclo de vida. Se enumera la ejecución del escenario reactivo en tres fases:

- **Fase 1**: Desarrollo del módulo de software siguiendo la metodología tradicional.
- **Fase 2**: Auditoría de seguridad del módulo desarrollado.
- **Fase 3**: Corrección y resolución de fallos de seguridad.

La Figura 6.4 describe gráficamente las fases en las que se ha dividido la ejecución del escenario reactivo. Para la primera fase se desglosan las sub-fases del ciclo de vida del software a tener en cuenta para medir los tiempos de ejecución.

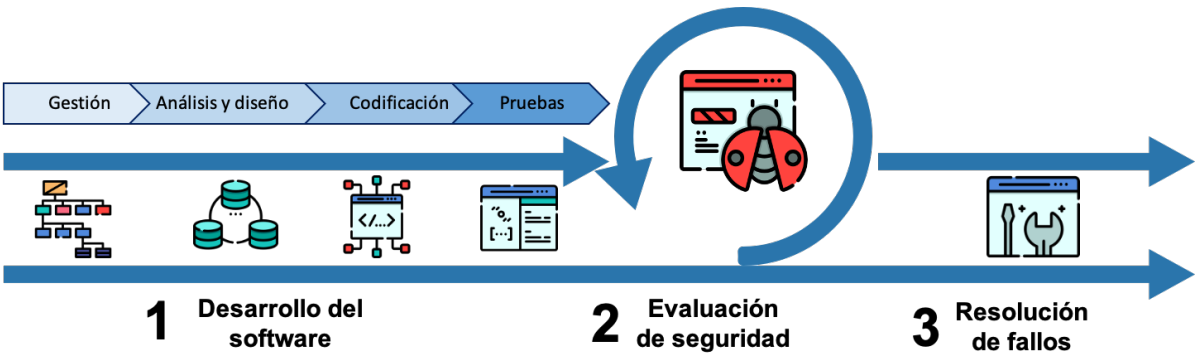


Figura 6.4. Fases del escenario reactivo. *Fuente: propia.*

6.3.2 Escenario preventivo

Este escenario aplica una metodología de desarrollo del software preventiva. En primer lugar, se implanta el modelo Viewnext-UEx, junto con la formación y preparación del ecosistema seguro. Con todo el equipo formado, se construye el módulo ejecutando las actividades de seguridad del modelo Viewnext-UEx. Al igual que en el escenario anterior, cuando se termina el módulo se realiza una auditoría de seguridad que detecte posibles vulnerabilidades en el software. De nuevo, el equipo de trabajo corrige y resuelve los fallos de seguridad encontrados en la auditoría. En este caso, las vulnerabilidades se habrán identificado y resuelto durante todo el ciclo de vida del software. Se enumera la ejecución del escenario preventivo en las siguientes fases:

- **Fase 0:** Implantación del modelo Viewnext-UEx.
  - Formación en seguridad y preparación del ecosistema de seguridad.
  - Integración de las actividades de seguridad.
- **Fase 1**: Desarrollo del módulo de software siguiendo el modelo Viewnext-UEx.
- **Fase 2**: Auditoría de seguridad del módulo desarrollado.
- **Fase 3**: Corrección y resolución de fallos de seguridad.

El escenario preventivo incorpora la implantación del modelo Viewnext-UEx como reflejan las fases de la Figura 6.5. La fase 0 no se tiene en cuenta para medir los tiempos de ejecución del escenario. El resto de fases coinciden con el escenario reactivo (Figura 6.4), siendo la diferencia la incorporación las actividades de seguridad del modelo Viewnext-UEx en el ciclo de vida del software.

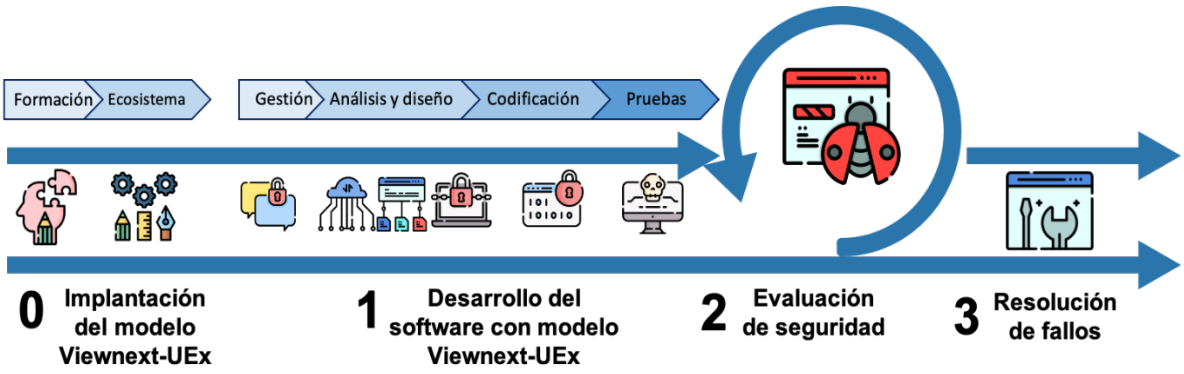


Figura 6.5. Fases del escenario preventivo. *Fuente: propia.*

6.3.3 Evaluación del caso de estudio

La evaluación de los escenarios planteados se fundamenta en dos aspectos principales. La verificación de la seguridad de los módulos desarrollados a través de un conjunto de métricas de seguridad y la productividad del desarrollo utilizando los tiempos de ejecución en función de la fase y la actividad de seguridad en cada escenario. Los siguientes apartados detallan las métricas para realizar la evaluación.

6.3.3.1 Métricas de seguridad

Las métricas de seguridad utilizadas en la evaluación de cada escenario se fundamentan en la identificación y clasificación de vulnerabilidades mediante sus características técnicas. Por lo tanto, se determinan las siguientes características para evaluar ambos escenarios:

- **Número de vulnerabilidades**: número de fallos de seguridad que permiten explotar el software sin tener en cuenta la recurrencia existente.
- **Tipo de vulnerabilidad**: elemento que agrupa y/o clasifica una vulnerabilidad que comparte características similares con otras en base a estándares de referencia como [64]–[66], [72].
    - **Validación de entradas**: vulnerabilidad que permite introducir en los campos de entradas de una aplicación o web, sentencias de uso malintencionado para extraer información confidencial de un sistema.
    - **Gestión de sesiones**: vulnerabilidad relativa a interceptar o suplantar la sesión de un usuario en una aplicación o web con el objetivo de realizar las operaciones accesibles al usuario en el nombre de este.
    - **Política de control de acceso**: vulnerabilidad que busca suplantar la identidad de un usuario en una aplicación o web, para realizar las operaciones accesibles al usuario sin su permiso o consentimiento.
    - **Autorización**: vulnerabilidad destinada a conseguir mayores privilegios de los que posee un usuario en una aplicación y acceder a información confidencial.
    - **Sistemas y arquitectura**: vulnerabilidades relacionadas con una configuración errónea del sistema o entorno de la aplicación.
    - **Otros tipos de vulnerabilidades**: pueden clasificarse en tipos como la gestión de errores, configuración incorrecta por uso de componentes o librerías de código abierto, protección y tratamiento de datos, etc.

- **Categoría**: valor interno otorgado para clasificar las vulnerabilidades en función de dos grandes categorías:
  - **Arquitectura:** vulnerabilidades relacionadas con la explotación de fallos o errores en el entorno o configuración de la aplicación.
  - **Desarrollo:** vulnerabilidades relacionadas con la explotación de aspectos puros del código fuente.
- **Criticidad**: valor otorgado al impacto de explotar una vulnerabilidad. Se utiliza el estándar *Common Vulnerability Score System* (CVSS) [72] con sus cinco niveles de puntuación de la criticidad como recoge la Tabla 6.3:

Tabla 6.3. Clasificación de la criticidad de las vulnerabilidades según CVSS. *Fuente: CVSS*

Calificación	Puntuación
Informativa	[0]
Baja	[0.1]- [3.9]
Media	[4.0]- [6.9]
Alta	[7.0]- [8.9]
Crítica	[9.0]- [10]

6.3.3.2 Métricas de productividad

Comparar los tiempos dedicados a la consecución del experimento comercial permite conocer cuál de los escenarios planteados resulta de mayor productividad para una empresa de desarrollo de software. En el caso de estudio se intenta determinar si es más rentable utilizar un modelo clásico, cuyas medidas reactivas identifican y resuelven las vulnerabilidades al final del proceso de desarrollo, o avanzar hacia el modelo Viewnext-UEx, cuyo enfoque preventivo anticipa la detección de vulnerabilidades antes de entregar el software.

Para ello, se mide el tiempo global de la ejecución del desarrollo del proyecto, los tiempos de la fase relativa a la resolución de los fallos de seguridad y el tiempo dedicado a la integración de las actividades de seguridad en cada una de las fases definidas en el apartado 5.3.2 Indicadores de productividad del desarrollo.

La Tabla 6.4 recoge las métricas temporales en función de cada escenario. Para el escenario preventivo que aplica el modelo Viewnext-UEx se contemplan los tiempos por fase según la actividad de seguridad ejecutada. Sin embargo, el escenario reactivo no

contempla dichos tiempos debido a que no existen actividades de seguridad ejecutadas durante el desarrollo del proyecto.

Tabla 6.4. Resumen de métricas temporales en función del escenario. *Fuente: propia*

Indicador temporal		Escenario reactivo	Escenario preventivo
Total en horas	Desarrollo	✓	✓
	Resolución fallos de seguridad	✓	✓
Porcentaje temporal en función de la fase	Gestión	✓	✓
	Análisis y diseño	✓	✓
	Codificación	✓	✓
	Pruebas	✓	✓
	Evaluación	✓	✓
Porcentaje temporal Actividades de seguridad	Gestión	✗	✓
	Análisis y diseño	✗	✓
	Codificación	✗	✓
	Pruebas	✗	✓
	Evaluación	✗	✓

La Figura 6.6 simula cómo se representarán los tiempos de ejecución de ambos escenarios en función de las fases del caso de estudio. A diferencia del escenario reactivo (Tabla 6.4), el preventivo recoge la particularidad de medir el tiempo dedicado a las actividades específicas de seguridad que corresponde a la aplicación del modelo Viewnext-UEx. Este tiempo dedicado a la seguridad será representado en azul claro, siendo el azul oscuro el tiempo dedicado al resto del desarrollo.

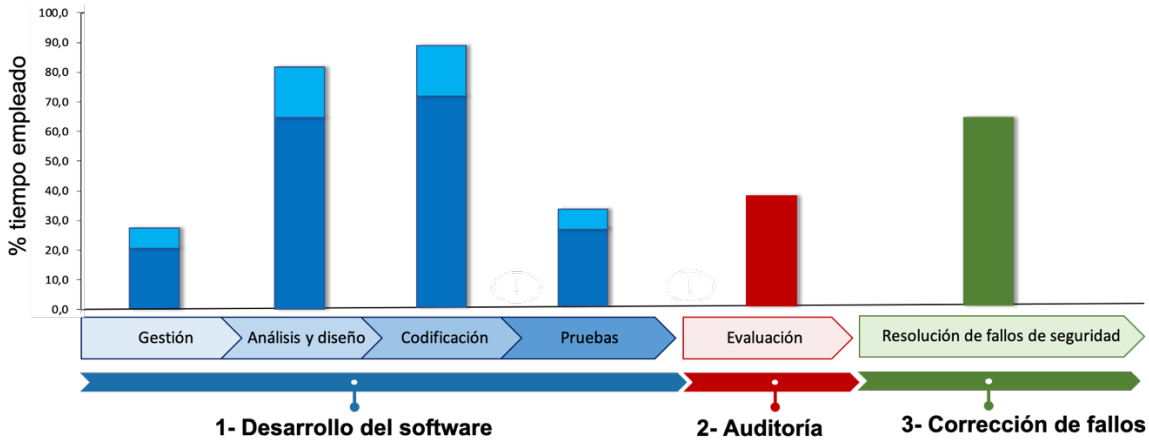


Figura 6.6. Representación gráfica de los tiempos y fases a medir. *Fuente: propia.*

6.4 Resultados escenario reactivo

Se presentan los resultados obtenidos tras la ejecución de las fases del escenario reactivo (Figura 6.4). Este escenario recoge las métricas sobre la seguridad y los costes temporales del software desarrollado. Las métricas de seguridad se extraen de la fase de evaluación y se centran en definir las vulnerabilidades identificadas (apartado 6.3.3.1 Métricas de seguridad). Los costes temporales agrupan los resultados de los tiempos (por fases) dedicados a construir el software de manera reactiva (apartado **¡Error! No se encuentra el origen de la r eferencia.**).

Durante la ejecución del escenario reactivo se encuentran un total de 19 vulnerabilidades distintas. La criticidad es alta en un porcentaje predominante (47,37%), dado que 9 de las vulnerabilidades encontradas alcanzan este nivel de criticidad. El resto de vulnerabilidades se reparten entre criticidad baja (5) y media (4), sumando el mismo porcentaje que las altas sobre el total (47,4%). Por otro lado, solo se encuentra una vulnerabilidad considerada de criticidad crítica (5,26%).

Con respecto a la categoría, destaca que el 63,2% (12) corresponden a fallos de seguridad relacionados con la arquitectura de la aplicación o la configuración errónea del entorno. Las otras 7 vulnerabilidades (36,8%) pertenecen a cuestiones relacionadas con la explotación del código fuente.

Por otro lado, la clasificación de las vulnerabilidades en función del tipo nos indica que 12 de ellas (63,16%) se agrupan en dos tipos. Están relacionadas con la validación de entradas del usuario en la aplicación y la configuración del entorno de la aplicación. El resto de tipos como autorización (2), gestión de la sesiones (2) y manipulación de errores y *logs* (3) conforma el 36,84% restante.

La Figura 6.7 resume gráficamente los resultados comentados en el escenario reactivo:

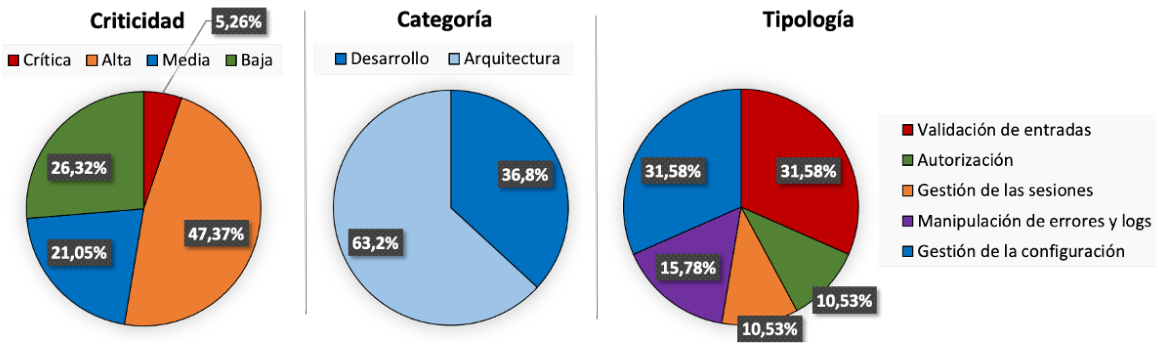


Figura 6.7. Análisis de las vulnerabilidades del escenario reactivo. *Fuente: propia.*

La Tabla 6.5 recoge la información detallada de las vulnerabilidades identificadas en el escenario reactivo. Se clasifican e identifican en función de la tipología, el nombre general que describe la vulnerabilidad, su criticidad y la categoría donde se enmarca.

Tabla 6.5. Información de las vulnerabilidades detectadas en el escenario reactivo. *Fuente: propia*

Tipo	Vulnerabilidad	Criticidad	Categoría
Validación de entradas	*Stored Cross Site Scripting*	Crítica [9-10]	Desarrollo
	Reflected Cross Site Scripting	Alta [7-8.9]	Desarrollo
	Base de datos accesible	Baja [0.1-3.9]	Desarrollo
	Atributo autocomplete no inhabilitado	Baja [0.1-3.9]	Desarrollo
	Cambio de peticiones POST por GET	Alta [7-8.9]	Arquitectura
	Directiva POST con parámetros no validados	Alta [7-8.9]	Desarrollo
Autorización	Escalada de privilegios funcional	Alta [7-8.9]	Desarrollo
	Escalada de privilegios por URL	Alta [7-8.9]	Desarrollo
Gestión de sesiones	Identificador de sesión desprotegido	Alta [7-8.9]	Arquitectura
	Cierre de sesión no implementado correctamente	Media [4.0-6.9]	Arquitectura
Manipulación de errores y *logs*	Información sensible del aplicativo y uso de componentes vulnerables	Media [4.0-6.9]	Arquitectura
	Información sensible en los metadatos	Baja [0.1-3.9]	Arquitectura
	Información sensible en el código fuente	Media [4.0-6.9]	Arquitectura
Gestión de la configuración	Página del servidor por defecto	Baja [0.1-3.9]	Arquitectura
	Aplicativo en HTTP en lugar de HTTPS	Alta [7-8.9]	Arquitectura
	Phising a través marcos	Alta [7-8.9]	Arquitectura
	Inyección de enlaces	Alta [7-8.9]	Arquitectura
	Respuesta de TCP *timestamp*	Baja [0.1-3.9]	Arquitectura
	Conexión concurrente desde distintas *IPs*	Media [4.0-6.9]	Arquitectura

6.4.2 Resultados de productividad

La ejecución de las fases planteadas en el escenario reactivo suma un total de 2228 horas empleadas por el equipo de trabajo. La contabilización se ha realizado según los criterios establecidos para la evaluación del caso de estudio, como se indica en el apartado **¡Error! No se encuentra el origen de la referencia.**. La Tabla 6.6 recoge los tiempos en horas y en porcentajes segmentados en función de la fase ejecutada.

Tabla 6.6. Tiempo por fase del desarrollo del escenario reactivo. *Fuente: propia.*

	1- Fase de desarrollo				2-Evaluación de seguridad	3-Resolución de fallos
	Gestión	A/D	Codificación	Pruebas		
Horas	99	495	1187	198	61	188
%	4,4	22,2	53,3	8,9	2,7	8,5

El porcentaje de tiempo en realizar el proceso de desarrollo (Fase 1) suma el 88,8% del tiempo total, siendo la que mayor dedicación requiere. El 11,2% restante es el porcentaje de tiempo dedicado a la evaluación de seguridad del software (2,7% - Fase 2) y de corregir o resolver las vulnerabilidades encontradas (8,5% - Fase 3). Dentro del proceso de desarrollo destacan el tiempo dedicado al análisis y diseño (22,2%) y la codificación del software al que se dedica más de la mitad de la ejecución del experimento (53,3%). La Figura 6.8 representa gráficamente la división de los tiempos dedicados a la ejecución del escenario reactivo:

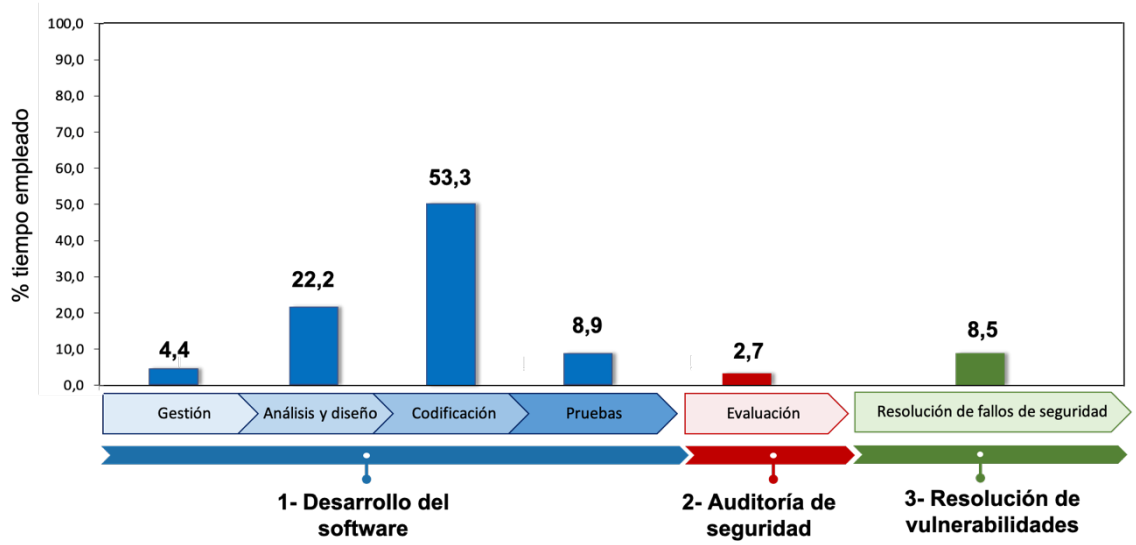


Figura 6.8. Resultados gráficos de los tiempos del escenario reactivo. *Fuente: propia.*

6.5 Resultados escenario preventivo

Los resultados del escenario preventivo (Figura 6.5) son recogidos utilizando la misma metodología que para el escenario reactivo. En el aspecto relativo a la seguridad del software se clasifican e identifican las vulnerabilidades de igual forma. Con respecto a los costes temporales se mantiene la agrupación de los resultados de los tiempos (por fases) con la nueva inclusión del tiempo dedicado a las actividades de seguridad del modelo aplicado.

6.5.1 Resultados de seguridad

En el escenario preventivo se encuentran un total de 6 vulnerabilidades. Destaca la falta de fallos de seguridad con criticidad considerada crítica. El 50% de las vulnerabilidades tienen criticidad alta (3), el otro 50% combinan criticidad media (1) y baja (2).

En cuanto a la categoría, el 83,3% de vulnerabilidades (5) corresponden a fallos de seguridad relacionados con la arquitectura de la aplicación o la configuración errónea del entorno. El 16,7% restante, que corresponde únicamente a una vulnerabilidad pertenece a cuestiones relacionadas con la explotación del código fuente.

Por otro lado, la clasificación de las vulnerabilidades en función del tipo presenta que 5 de ellas (83,3%) pertenece a fallos producidos en la configuración del entorno de la aplicación. La otra vulnerabilidad (16,7%) corresponde a un fallo que se produce en la validación de entradas del usuario en la aplicación. No se recogen vulnerabilidades para el resto de tipos estipulados como son la autorización, gestión de la sesiones o manipulación de errores.

La Figura 6.9 presenta a nivel gráfico un resumen de los resultados analizados del escenario preventivo.

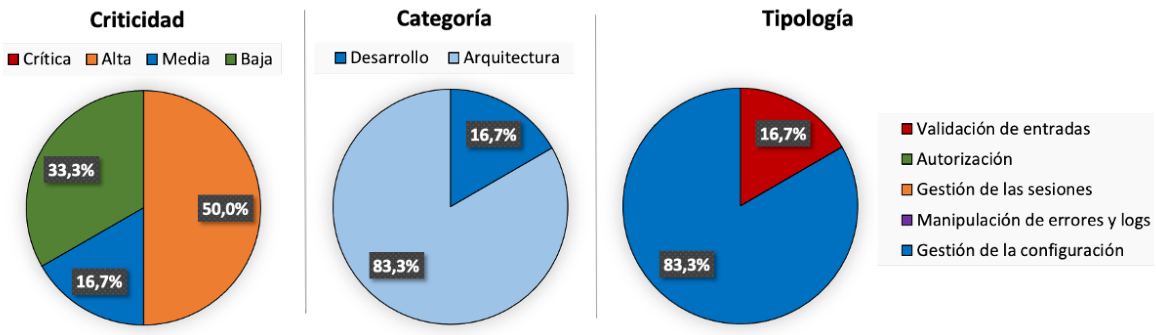


Figura 6.9. Análisis de las vulnerabilidades del escenario preventivo. *Fuente: propia.*

La Tabla 6.7 almacena la información de las vulnerabilidades identificadas en el escenario preventivo, en base a la tipología, la descripción de la vulnerabilidad, su criticidad y la categoría. Esta información se considera fundamental para el progreso y continuación del desarrollo del proyecto, por lo que es almacenada en el repositorio de vulnerabilidades (4.2.2.12 Repositorio de vulnerabilidades).

Tabla 6.7. Información de las vulnerabilidades detectadas en el escenario preventivo. *Fuente: propia*

Categoría	Vulnerabilidad	Criticidad	Tipo
Validación de entradas	Base de datos accesible	Baja [0.1-3.9]	Desarrollo
Gestión de la configuración	Certificados SSL débiles	Alta [7-8.9]	Arquitectura
	Servicios y puertos habilitados indebidamente	Alta [7-8.9]	Arquitectura
	Respuesta de TCP timestamp	Baja [0.1-3.9]	Arquitectura
	Conexión Concurrente desde distintas IPs	Media [4.0-6.9]	Arquitectura
	Firma SMB (Server Message Block) No requerida	Alta [7-8.9]	Arquitectura

6.5.2 Resultados de productividad

La ejecución de las fases planteadas en el escenario preventivo suma un total de 794 horas. Del mismo modo que en el escenario anterior, se utilizan los criterios establecidos para la evaluación del caso de estudio del apartado **¡Error! No se encuentra el origen de l a referencia.**. La Tabla 6.8 recoge los tiempos en horas y en porcentajes segmentados en función de la fase ejecutada. A diferencia de la Tabla 6.8, esta recoge los porcentajes de cada fase con la diferenciación en los tiempos dedicados a ejecutar las actividades de seguridad del modelo Viewnext-UEx.

Tabla 6.8. Tiempo por fase del desarrollo del escenario preventivo. *Fuente: propia.*

	1- Fase de desarrollo				2- Evaluación de seguridad	3- Resolución de fallos
	Gestión	A/D	Codificación	Pruebas		
Horas	49 (6,17%)	215 (27,07%)	383 (48,25%)	54 (6,79%)	25 (3,1%)	10 (1,3%)
Seguridad	1 (0,13%)	28 (3,53%)	21 (2,65%)	8 (1,01%)		
%	6,3	30,6	50,9	7,8	3,1	1,3

El proceso de desarrollo (Fase 1) suma en el escenario preventivo el 95,6% del tiempo total, siendo también la que mayor dedicación requiere. En este escenario el tiempo dedicado al resto de fases suma un 4,4%, repartidos entre la evaluación de seguridad (3,1% - Fase 2) y la corrección o resolución de vulnerabilidades encontradas (1,3% - Fase 3). Durante el proceso de desarrollo se recalca el tiempo dedicado al análisis y diseño (30,6%), a la codificación del software (50,9%) y a las pruebas (7,8%).

Con respecto a las actividades exclusivamente de seguridad del modelo Viewnext-UEx, de la fase de desarrollo (Fase 1), la dedicación temporal ascienda a un total de 58 horas, destacando que la mayor cantidad de tiempo se dedica a la fase de análisis y diseño (28 horas) y a la de codificación (21 horas), como puede apreciarse en la Tabla 6.8. En porcentaje, se corresponde con el 7,32% del total del caso de estudio (sumando los porcentajes de seguridad en la Tabla 6.8). De forma preventiva se introduce la seguridad en el software en las fases más tempranas, consiguiendo anticipar la identificación de vulnerabilidades.

La Figura 6.10 representa gráficamente la división de los tiempos dedicados a la ejecución del escenario preventivo. Destaca la zona marcada en azul claro de la fase de desarrollo (Fase 1) que corresponde a las actividades específicas de seguridad del modelo Viewnext-UEx.

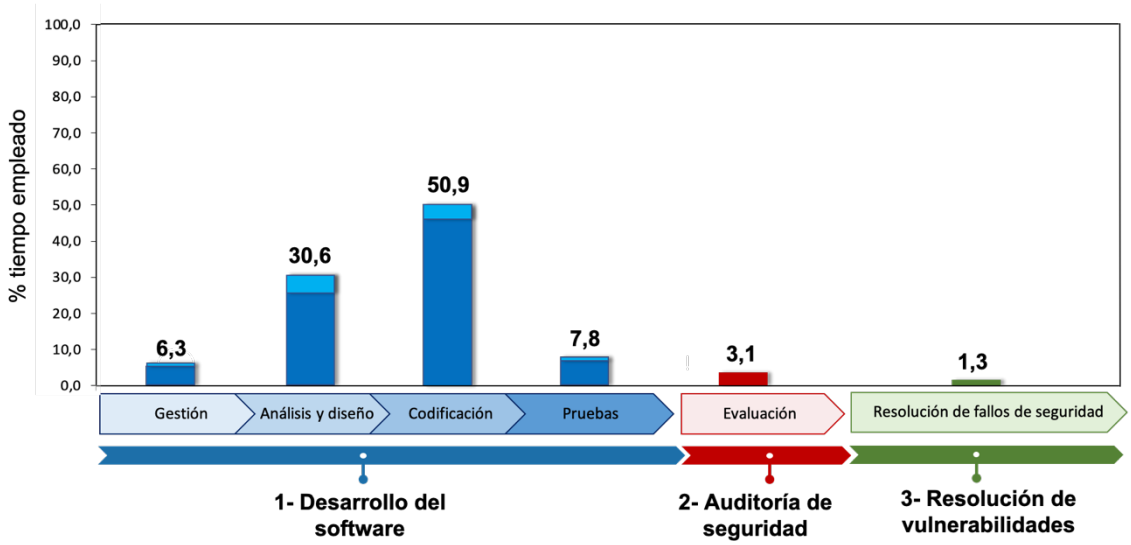


Figura 6.10. Resultados gráficos de los tiempos del escenario preventivo. *Fuente: propia.*

6.6 Comparativa de resultados y discusión

Esta sección presenta una comparativa de resultados y discusión entre el escenario reactivo y el preventivo, siguiendo los mismos criterios de seguridad y de productividad de los apartados anteriores. Ambos escenarios, como ya se ha señalado, aunque con

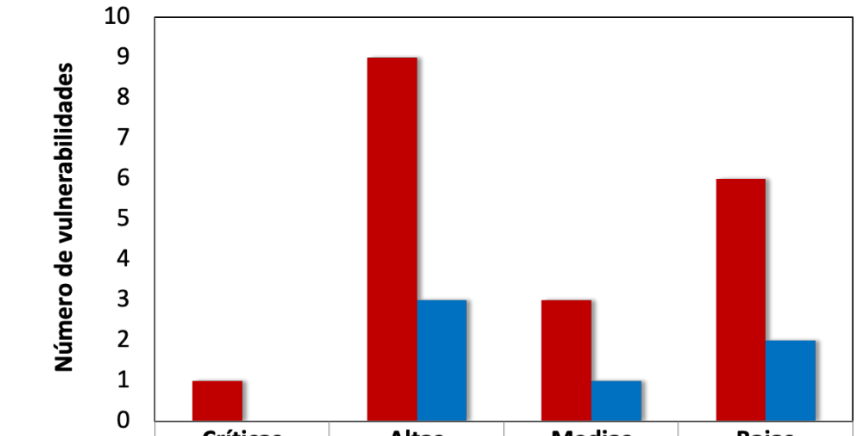
metodologías distintas (reactiva y preventiva) y diferente dedicación temporal, han sido codificados por el mismo equipo de trabajo (mismas personas), en el mismo proyecto de software y sin que se produzca cruce de información entre los desarrollos de ambos escenarios.

6.6.1 Comparativa y discusión de la seguridad

Realizando un estudio comparativo entre los escenarios anteriormente expuestos, lo primero que se observa es que el escenario preventivo presenta menos vulnerabilidades que el escenario reactivo. En el escenario reactivo se identifican 19 fallos de seguridad frente a 6 del escenario reactivo. En porcentajes implica que el escenario preventivo tiene una reducción del 68,42% en la cantidad de vulnerabilidades en el desarrollo de software con respecto al escenario reactivo. Como puede apreciarse, se trata de un decremento considerable en cuanto al número de vulnerabilidades.

En segundo lugar, es importante determinar no solo la cantidad de vulnerabilidades, sino también su gravedad. En este sentido, en el estudio del impacto o criticidad de las vulnerabilidades también se aprecia una reducción de la severidad en el escenario preventivo. El escenario reactivo presenta que el 73,68% de las vulnerabilidades se clasifican como críticas, altas o medias, frente al 50% de vulnerabilidades detectadas en el escenario preventivo cuya criticidad fue alta o media, sin encontrarse ninguna crítica.

La Figura 6.11 ofrece la comparativa del número de vulnerabilidades de cada escenario en función de su criticidad.



	Críticas	Altas	Medias	Bajas
Reactivo	1	9	3	6
Preventivo	0	3	1	2

Figura 6.11. Comparativa de vulnerabilidades en número y criticidad. *Fuente: propia.*

En un tercer nivel de detalle, en cuanto al tipo de vulnerabilidades producidas, ambos escenarios detectan un mayor número de vulnerabilidades relacionadas con aspectos de la arquitectura que con cuestiones de la codificación (desarrollo). La Tabla 6.9 compara las vulnerabilidades de ambos escenarios por categoría, tipo (arquitectura o desarrollo) y criticidad (según el estándar *Common Vulnerability Scoring System*). Puede observarse que en el escenario preventivo, las vulnerabilidades se reducen a un conjunto menor de categorías de vulnerabilidades (en este caso, a la validación de entradas y gestión de configuraciones). Ello puede ayudar también a centrar los esfuerzos de depuración en un rango menor de situaciones indeseables. Si bien es cierto que este número menor de tipos diferentes de vulnerabilidades es consecuencia directa también de la reducción en el número de vulnerabilidades producidas en el escenario preventivo.

Por todo lo expuesto, es destacable que el modelo Viewnext-UEx no solo reduce el número de vulnerabilidades producidas y detectadas, así como el tipo de vulnerabilidades del conjunto considerado (aspectos cuantitativos), sino que, además, las vulnerabilidades identificadas son de menor gravedad en cuanto a sus consecuencias (aspecto cualitativo). Es decir, que el modelo propuesto en esta Tesis Doctoral, a modo de avance, apunta a una mejora cuantitativa y cualitativa en la producción de software. Por el momento, no se han encontrado estudios con datos reales que prueben la validez metodológica a nivel de seguridad de un modelo y que permitan la comparativa de los resultados obtenidos.

Tabla 6.9. Información del conjunto de vulnerabilidades encontradas cada escenario. *Fuente: propia*

Categoría	Vulnerabilidad	Criticidad	Tipo	Reactivo	Preventivo
Validación de entradas	*Stored Cross Site Scripting*	Crítica [9-10]	Desarrollo	X	
	Reflected Cross Site Scripting	Alta [7-8.9]	Desarrollo	X	
	Base de datos accesible	Baja [0.1-3.9]	Desarrollo	X	X
	Atributo autocomplete no inhabilitado	Baja [0.1-3.9]	Desarrollo	X	
	Cambio de peticiones POST por GET	Alta [7-8.9]	Arquitectura	X	
	Directiva POST con parámetros no validados	Alta [7-8.9]	Desarrollo	X	
Autorización	Escalada de privilegios funcional	Alta [7-8.9]	Desarrollo	X	
	Escalada de privilegios por URL	Alta [7-8.9]	Desarrollo	X	
Gestión de sesiones	Identificador de sesión desprotegido	Alta [7-8.9]	Arquitectura	X	
	Cierre de sesión no implementado correctamente	Media [4.0-6.9]	Arquitectura	X	
Manipulación de errores y *logs*	Información sensible del aplicativo y uso de componentes vulnerables	Media [4.0-6.9]	Arquitectura	X	
	Información sensible en los metadatos	Baja [0.1-3.9]	Arquitectura	X	
	Información sensible en el código fuente	Media [4.0-6.9]	Arquitectura	X	
Gestión de la configuración	Página del servidor por defecto	Baja [0.1-3.9]	Arquitectura	X	
	Aplicativo en HTTP en lugar de HTTPS	Alta [7-8.9]	Arquitectura	X	
	Phising a través marcos	Alta [7-8.9]	Arquitectura	X	
	Inyección de enlaces	Alta [7-8.9]	Arquitectura	X	
	Certificados SSL débiles	Alta [7-8.9]	Arquitectura		X
	Servicios y puertos habilitados indebidamente	Alta [7-8.9]	Arquitectura		X
	Respuesta de TCP *timestamp*	Baja [0.1-3.9]	Arquitectura	X	X
	Conexión concurrente desde distintas *IPs*	Media [4.0-6.9]	Arquitectura	X	X
	Firma SMB (*Server Message Block*) No requerida	Alta [7-8.9]	Arquitectura		X

6.6.2 Comparativa y discusión de la productividad

La Figura 6.12 presenta en porcentajes el tiempo global dedicado a las distintas fases (desarrollo de software, evaluación de seguridad y corrección de vulnerabilidades) para los dos escenarios del experimento llevado a cabo en esta Tesis Doctoral. La fase de desarrollo se divide, a su vez, en las sub-fases de gestión, análisis y diseño, codificación y pruebas, que corresponden con la tarea que cumplen dentro del ciclo de vida del desarrollo del software. Los resultados de cada escenario son representados en porcentajes de tiempo para facilitar la comparativa, ya que la cantidad total de tiempo empleado es diferente en cada escenario.

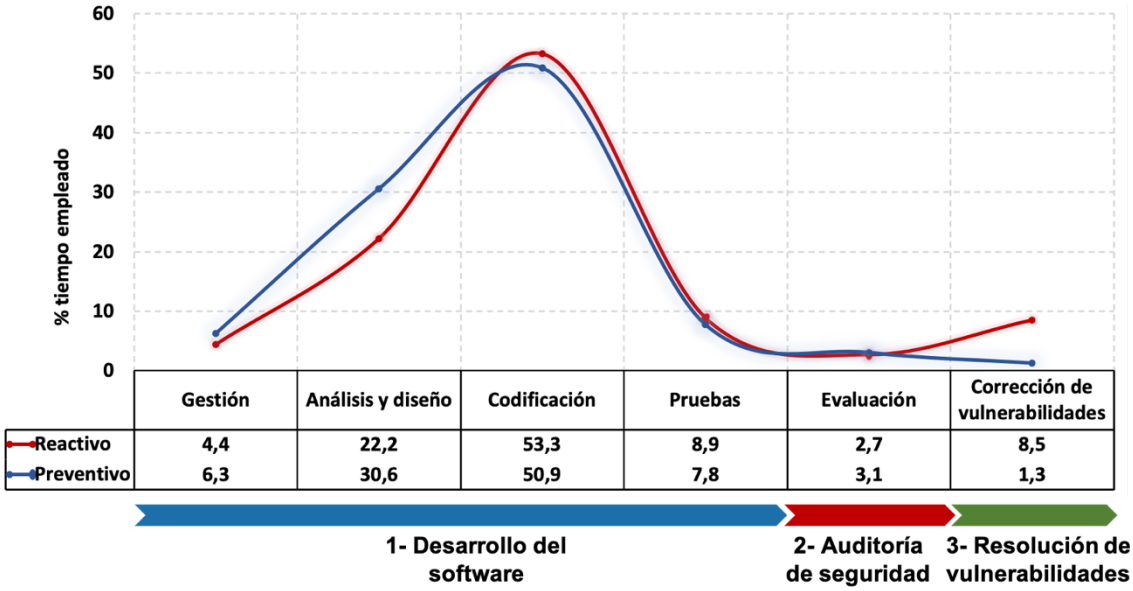


Figura 6.12. Comparativa de los tiempos empleados en cada escenario por fase del proceso. *Fuente: propia.*

En los datos anteriores se aprecia que, en comparativa con el escenario reactivo, el escenario preventivo destina más tiempo a las fases iniciales de gestión y de análisis y diseño, y menos a la de codificación. Esto es debido a que el escenario preventivo distribuye el coste temporal relacionado con las actividades de seguridad del modelo Viewnext-UEx entre todas las fases del desarrollo de software (Tabla 5.6).

Sin embargo, es de destacar la fase de corrección o resolución de vulnerabilidades, donde existe una diferencia de 7,2% del escenario preventivo frente al reactivo. El escenario reactivo se ve afectado en esta fase (Fase 3) por tener que corregir un gran número de vulnerabilidades detectadas en la fase de evaluación (Fase 2). La causa de ello es que el enfoque reactivo de la metodología no afronta la seguridad hasta la fase de evaluación, encontrando al final del desarrollo mayor número de vulnerabilidades.

Resulta interesante poner de relieve esta considerable reducción de esfuerzos, precisamente en la fase final de desarrollo de un proyecto, donde es usual que se acumulen retrasos de fases anteriores, que puedan incluso haber repercutido en costes por dedicaciones extra del personal implicado, por ejemplo.

Esta reducción de tiempos en la fase de corrección de vulnerabilidades en el escenario preventivo viene a compensar el tiempo dedicado a las actividades de seguridad distribuidas durante las fases del desarrollo de software (Fase 1). De esta forma, la fase de resolución de vulnerabilidades en el escenario preventivo produce un coste temporal mínimo.

Por ello, se puede afirmar que considerar un escenario reactivo producirá una mayor dedicación en la etapa final del proyecto donde, con el software totalmente desarrollado, deberán producirse numerosas correcciones. Esta circunstancia, como se ha comentado, producirá en el equipo del proyecto una carga de trabajo extra inesperada en la planificación. En este sentido, es deseable reducir al máximo la cantidad de vulnerabilidades que deben solucionarse y evitar posibles situaciones no planificadas, tal y como tendría lugar bajo un modelo preventivo.

Por el contrario, la distribución de actividades de seguridad del modelo Viewnext-UEx en el escenario preventivo consigue distribuir su coste temporal de manera uniforme durante la fase del desarrollo (Fase 1) consiguiendo minimizar el tiempo de la resolución de vulnerabilidades en la fase final. Además, como se ha demostrado, el software final desarrollado en este escenario no solo es funcionalmente correcto, sino también más seguro.

6.7 Discusiones adicionales y experiencias

Se recogen algunas consideraciones como discusión final del experimento. Es importante insistir en que el experimento se divide en dos escenarios que aplican metodologías diferentes (reactivo y preventivo), pero que comparten materiales (mismo proyecto de software, herramientas, etc.) y que son codificados de manera secuencial por el mismo equipo de trabajo sin, además, interferir la información de uno a otro. Estas consideraciones adicionales se agrupan, por un lado, en los aspectos científicos extraídos del experimento y, por otro, en la experiencia cualitativa aportada por el responsables del proyecto de software con la función de *Scrum Manager* y miembros del equipo de trabajo con el objetivo de conocer la experiencia por parte de la empresa Viewnext S.A.

Con respecto a las discusiones finales del experimento se obtienen las siguientes afirmaciones:

- Ambos escenarios producen software igual de seguro, aunque el proceso de desarrollo del software seguido por el escenario preventivo evita la mayoría de las vulnerabilidades del escenario reactivo, con menos esfuerzo en la corrección de vulnerabilidades.
- La implementación del modelo Viewnext-UEx en el escenario preventivo produce menor impacto temporal en las fases finales del experimento y mayor seguridad al software desde el momento de su desarrollo inicial.

Por otro lado, se exponen las valoraciones cualitativas del responsable del proyecto de software y el equipo de trabajo en el que se ha realizado el experimento.

- Los resultados del experimento han generado en el equipo de trabajo una concienciación positiva ante la necesidad lógica de incluir la seguridad en todo el proceso de desarrollo del software.
- La identificación de vulnerabilidades detectadas en fases avanzadas (escenario reactivo) ha impactado negativamente en la planificación temporal establecida inicialmente en el proyecto. Además, ha producido un mayor nivel de carga de trabajo que ha recaído en el equipo de desarrolladores del proyecto.
- La aparición de vulnerabilidades desconocidas produce incertidumbre y desasosiego en el equipo de trabajo con respecto al resto de módulos desarrollados en el proyecto con anterioridad, a los que no se les ha aplicado el modelo Viewnext-UEx.
- La mayor dedicación en el escenario preventivo, o mayor atención, a la fase de análisis debido a las actividades de seguridad, ha generado una reducción importante en el tiempo de codificación. No ha ocurrido así con el tiempo destinado al diseño y pruebas del proyecto que se ha mantenido relativamente similar.
- La atención extra provocada por las actividades del modelo Viewnext-UEx en la toma de requisitos, ha permitido concienciar al cliente para poder ampliar el tiempo dedicado en esta fase del proceso de desarrollo que, erróneamente, se da por cubierta en la mayoría de los casos.
- El equipo de trabajo reporta que los fallos de seguridad encontrados mediante el análisis de código estático son fáciles de corregir, pero, por el contrario, los fallos detectados en la evaluación de seguridad (testing de seguridad manual) son complejos e incluso pueden requerir cambios arquitectónicos, pudiendo generar un aumento en los recursos para su resolución.

6.8 Síntesis del capítulo

Este capítulo expone los resultados de un caso de estudio (experimento comercial) que construye software en dos escenarios diferentes, uno reactivo y otro preventivo, que únicamente se diferencian en la metodología del proceso de desarrollo del software.

Primeramente, se ha descrito el diseño del caso de estudio detallando los materiales (proyecto de software y características específicas del contexto) y el método de cada escenario. Seguidamente, se han establecido las fases de ejecución de cada escenario, en el que el reactivo ha seguido el método tradicional y el preventivo ha aplicado la metodología del modelo Viewnext-UEx presentado en esta tesis.

Posteriormente, se ha establecido un proceso de evaluación para los dos escenarios. Dicha evaluación se ha realizado utilizando distintas métricas de seguridad y de temporalidad.

Tras la realización de cada escenario se analizan, por separado, los resultados obtenidos. Por último, estos resultados han sido discutidos mediante el análisis en profundidad y una comparativa que ha permitido extraer unas discusiones finales, que presentadas junto con la experiencia del equipo de trabajo y el responsable del proyecto de software dan por finalizado el capítulo.

"Para el investigador no existe alegría comparable a
la de un descubrimiento, por pequeño que sea."
Alexander Fleming

Capítulo 7

7. Conclusiones y trabajos futuros

La principal conclusión que se ha obtenido en esta Tesis Doctoral es:

La incorporación preventiva y sistemática de las prácticas de seguridad del modelo Viewnext-UEx reduce tanto el número de vulnerabilidades como su gravedad, lo que conduce a una mejora en los aspectos de seguridad y optimización en los tiempos de desarrollo del ciclo de vida del software.

Esta conclusión global se ha obtenido a partir de las conclusiones parciales que se exponen a continuación:

El estudio de la literatura ha evidenciado la falta de trabajos de investigación que analicen pormenorizadamente los modelos de desarrollo de software seguro más populares, siendo necesario presentar en esta Tesis Doctoral un estudio del arte riguroso que recoge un amplio conjunto de técnicas que integran la seguridad en el software mientras es analizado, diseñado, codificado, probado e implantado.

De este análisis, se concluye la necesidad de realizar desarrollos guiados por normas y estándares que fuercen el uso de prácticas seguras para construir software seguro y de calidad, extrayéndose de la comparativa de los modelos estudiados un total de 11 actividades de seguridad consideradas imprescindibles a incluir en todo modelo de desarrollo seguro.

Se evidencia, del mismo modo, ciertas limitaciones en los modelos comparados que hace necesaria la incorporación de nuevas actividades que permitan construir un modelo de desarrollo de software seguro que garantice la calidad en los desarrollos.

En esta línea, se crean 3 nuevas actividades de seguridad que, junto con las 11 detectadas de la revisión de modelos anteriores, configuran el modelo Viewnext-UEx propuesto en esta Tesis Doctoral.

El nuevo modelo propuesto se conforma, por tanto, de 14 actividades que se clasifican en 4 áreas de desarrollo (Políticas, Metodología SDL, Supervisión y Observatorio) y se ejecutan de forma preventiva, logrando así crear un nuevo marco de trabajo (modelo Viewnext-UEx) que incorpora de manera sistemática prácticas de seguridad a lo largo de todo el ciclo de vida de desarrollo del software.

El proceso experimental llevado a cabo para probar la eficacia de este nuevo modelo diseñado lleva a plantear un caso de estudio con dos escenarios diferentes, uno con enfoque reactivo y otro preventivo, que permita comparar la eficacia del nuevo modelo propuesto frente al modelo estándar de desarrollo de software, a través de la comparativa de métricas específicas de seguridad y costes. Con ello se concluye que ambos escenarios desarrollan software con el mismo nivel de seguridad, pero que el escenario preventivo que aplica el modelo Viewnext-UEx y, por tanto, que integra la seguridad del software desde las fases iniciales del desarrollo, reduce drásticamente la aparición de vulnerabilidades, su criticidad y el tiempo de resolución de los fallos de seguridad, frente a los resultados obtenidos del modelo reactivo que aplica la seguridad únicamente en las fases finales del desarrollo.

Asimismo, el desarrollo del proceso experimental en el marco de un contexto de trabajo real como ha sido la empresa Viewnext S.A. ha permitido poner de relieve la optimización del nuevo modelo y su adaptación a proyectos reales en factorías de desarrollo. La disponibilidad de datos reales, no simulados, obtenidos a través de esta investigación garantizan el ajuste del nuevo modelo al contexto empresarial y las posibilidades de transferencia directa del conocimiento obtenido mediante este estudio, facilitando que cualquier empresa tecnológica integre este modelo como su metodología de desarrollo de software seguro.

Los resultados obtenidos de la creación y ejecución del modelo Viewnext-UEx, mostrados en esta Tesis Doctoral, han llevado a la empresa Viewnext S.A., no solo a adoptar

esta nueva metodología de desarrollo de software seguro en su sistema de producción, sino a comercializarla, lo que prueba la validez de la propuesta y su ajuste al mercado.

Siguiendo la estela de esta investigación, las siguientes líneas de trabajo a abordar se corresponderían con:

- Estudiar la validación del modelo Viewnext-UEx mediante la implantación en un número considerable de proyectos de software que permitan contrastar los resultados obtenidos con el caso de estudio expuesto en esta Tesis Doctoral.

- Adaptar el modelo Viewnext-UEx para que cumpla la filosofía DevOpSec en la que el desarrollo se relaciona muy directamente con las operaciones y la seguridad.

Referencias

[1] Centro Criptológico Nacional, "Ciberamenazas y Tendencias Edición 2017," 2017.

[2] Centro Criptologico Nacional, "Informe de Ciberamenazas y Tendencias 2018," 2018. [Online]. Available: https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/2835-ccn-cert-ia-09-18-ciberamenzas-y-tendencias-edicion-2018-1/file.html.

[3] Centro Criptológico Nacional, "Informe de Ciberamenazas y Tendencias 2019," 2019. [Online]. Available: https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/3776-ccn-cert-ia-13-19-ciberamenazas-y-tendencias-edicion-2019-1/file.html.

[4] I. Cisco Systems, "Informe anual sobre ciberseguridad de 2017," 2017. http://www.cisco.com/c/dam/m/digital/1226019/Cisco_2017_ACR_es-xl.pdf (accessed Jun. 20, 2017).

[5] A. S. S. Sodiya, S. A. A. Onashoga, and O. B. B. Ajayi, "Towards building secure software systems," *Inf. Universe J. Issues Informing Sci. & Inf. Technol.*, vol. 3, pp. 635–646, 2006.

[6] National Institute of Standards and Technology, "The Economic Impacts of Inadequate Infrastructure for Software Testing," *Quality*, 2002. https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf (accessed Jun. 18, 2017).

[7] J. C. Sancho Núñez, A. Caro Lindo, and P. García Rodríguez, "Análisis de

metodologías de Desarrollo de Software Seguro," in *Jornadas Nacionales de investigación en Ciberseguridad (JNIC)*, 2016, pp. 42–47.

[8] J. C. Sancho Núñez, "Un Modelo particularizado para el Desarrollo de Software Seguro," in *Catálogo de Investigación Joven en Extremadura. Volumen III*, In press, 2020.

[9] J. C. Sancho Núñez, A. Caro Lindo, P. García Rodríguez, and Á. Quesada, "Categorización de Actividades de Seguridad en el Desarrollo de Software," in *Jornadas de Ingeniería del Software y Bases de Datos*, 2016, pp. 565–568, [Online]. Available: http://www.kybele.etsii.urjc.es/jisbd2013/.

[10] J. C. Sancho Núñez, M. L. Castaño, A. C. Lindo, J. A. Félix, G. Rodríguez, and A. B. Gómez, "Herramienta de entrenamiento para el desarrollo de software seguro," in *Actas de las XXIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2019).*, 2019, pp. 1–4, [Online]. Available: https://biblioteca.sistedes.es/articulo/herramienta-de-entrenamiento-para-el-desarrollo-de-software-seguro/.

[11] J. C. Sancho Núñez, A. Caro Lindo, J. Méndez Chaves, and P. García Rodríguez, "Taxonomy of threats, vulnerabilities and good practices for the secure software development," in *Experiencias Innovadoras de Investigación para un Futuro Sostenible*, Octaedro, 2020.

[12] J. C. Sancho Núñez, A. Caro Lindo, L. Fondón, and J. A. Félix de Sande, "Herramienta para la identificación de requisitos de seguridad en un Modelo de Desarrollo Seguro," in *Reunión Española sobre Criptología y Seguridad de la Información (RECSI)*, 2018, pp. 92–95.

[13] A. Bravo Gomez, J. C. Sancho Núñez, and A. Caro Lindo, "Extracción de conocimiento a partir de fuentes de datos reales procedentes de la monitorización de eventos de seguridad," in *V Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2019)*, 2019, pp. 86–93.

[14] J. C. Sancho, A. Caro, M. Ávila, and A. Bravo, "New approach for threat classification and security risk estimations based on security event management," *Futur. Gener. Comput. Syst.*, vol. 113, pp. 488–505, 2020, doi: 10.1016/j.future.2020.07.015.

[15] J. C. Sancho Núñez, A. Caro Lindo, P. García Rodríguez, and J. A. Félix de Sande, "Metodología de Implantación Empresarial de un Modelo de Desarrollo de Software

Seguro," in *Jornadas Nacionales de investigación en Ciberseguridad (JNIC)*, 2017, pp. 128–133, [Online]. Available: http://jornadasciberseguridad.riasc.unileon.es/.

[16] J. A. Félix de Sande, J. C. Sancho Núñez, and A. Caro Lindo, "Evaluación y selección de un ecosistema de herramientas para un enfoque preventivo y continuo en modelos de desarrollo seguro de software," in *IV Jornadas Nacionales de investigación en Ciberseguridad (JNIC)*, 2018, pp. 87–94.

[17] J. C. S. Nunez, A. C. Lindo, and P. G. Rodriguez, "A Preventive Secure Software Development Model for a software factory: a case study," *IEEE Access*, vol. 8, pp. 1–1, 2020, doi: 10.1109/access.2020.2989113.

[18] J. C. Sancho Núñez, A. Caro Lindo, P. García Rodríguez, and J. A. Félix de Sande, "Modelo Emergente Preventivo para producir software seguro," in *V Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2019)*, 2019, pp. 151–158.

[19] Organización Internacional de Normalización, "ISO 27001 Seguridad de la Información," 2005. [Online]. Available: https://www.iso.org/isoiec-27001-information-security.html.

[20] OWASP Foundation, "OWASP Top Ten 2017," *2017*. https://owasp.org/www-project-top-ten/2017/ (accessed Mar. 23, 2020).

[21] NIST, "Software Assurance Tools: Web Application Security Scanner Functional Specification Version 1.0." https://www.nist.gov/publications/software-assurance-tools-web-application-security-scanner-functional-specification (accessed Mar. 23, 2020).

[22] WASC, "The WASC Threat Classification v2.0." http://projects.webappsec.org/w/page/13246978/Threat Classification (accessed Mar. 23, 2020).

[23] SANS and MITRE, "CWE/SANS TOP 25 Most Dangerous Software Errors." https://www.sans.org/top25-software-errors (accessed Mar. 23, 2020).

[24] MITRE, "Common Weakness Enumeration (CWE™)." https://cwe.mitre.org/data/index.html (accessed Mar. 23, 2020).

[25] NVD, MITRE, and NIST, "2020 CWE Top 25 Most Dangerous Software Weaknesses." https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html (accessed Oct. 10, 2020).

[26] A. Apvrille and M. Pourzandi, "Secure software development by example," *IEEE Security and Privacy*, vol. 3, no. 4. pp. 10–17, 2005, doi: 10.1109/MSP.2005.103.

[27] M. Solinas, L. Antonelli, and E. Fernandez, "Software secure building aspects in computer engineering," *IEEE Lat. Am. Trans.*, vol. 11, no. 1, pp. 353–358, 2013, doi: 10.1109/TLA.2013.6502829.

[28] D. Mellado, E. Fernandez-Medina, and M. Piattini, "A Security Requirements Engineering Process in Practice," *Lat. Am. Trans. IEEE (Revista IEEE Am. Lat.*, 2007.

[29] P. Pietikäinen, J. Röning, T. Siiskonen, and V. Ylimannela, *Handbook of The Secure Agile Software Development Life Cycle*. University of Oulu, 2014.

[30] OWASP Project, "OWASP Cornucopia." https://owasp.org/www-project-cornucopia/assets/files/Owasp-cornucopia-ecommerce_website-EN.pdf (accessed Mar. 01, 2018).

[31] V. Asthana, I. Tarandach, N. O'Donoghue, B. Sullivan, and M. Saario, "Practical Security Stories and Security Tasks for Agile Development Environments," *SAFECode*, p. 34, 2012, [Online]. Available: http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf.

[32] "Common Weakness Enumeration: CWE." http://cwe.mitre.org/index.html (accessed May 05, 2019).

[33] Continuum Security, "IriusRisk." https://iriusrisk.com/threat-modeling-tool/ (accessed Mar. 23, 2020).

[34] Adam Shostack, *【AdamShostack】Threat Modeling：Designing for Security*. 2014.

[35] ATT&CK, "MITRE." https://attack.mitre.org/.

[36] T. Xin and B. Xiaofang, "Online banking security analysis based on STRIDE threat model," *Int. J. Secur. its Appl.*, vol. 8, no. 2, pp. 271–282, 2014, doi: 10.14257/ijsia.2014.8.2.28.

[37] M. Venkatasen and P. Mani, "A risk-centric defensive architecture for threat modelling in e-government application," *Electron. Gov.*, vol. 14, no. 1, pp. 16–31, 2018, doi: 10.1504/EG.2018.089537.

[38] M. Hirano, N. Tsuzuki, S. Ikeda, and R. Kobayashi, "LogDrive: a proactive data collection and analysis framework for time-traveling forensic investigation in IaaS cloud environments," *J. Cloud Comput.*, vol. 7, no. 1, pp. 1–25, 2018, doi: 10.1186/s13677-018-0119-2.

[39] D. Seifert and H. Rez, "A security analysis of cyber-physical systems architecture for healthcare," *Computers*, vol. 5, no. 4, 2016, doi: 10.3390/computers5040027.

[40] Microsoft, "Microsoft Threat Modeling Tool," 2016. https://www.microsoft.com/en-us/download/details.aspx?id=49168 (accessed Sep. 20, 2018).

[41] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," *Proc. - Annu. Comput. Secur. Appl. Conf. ACSAC*, vol. Part F1334, pp. 55–64, 1999, doi: 10.1109/CSAC.1999.816013.

[42] C. W. (Johnny) Sia, "Misuse Cases and Abuse Cases in Eliciting Security Requirements," *Dep. Comput. Sci. Univ. Auckl.*, pp. 1–14, 2005.

[43] C. W. Ten, C. C. Liu, and M. Govindarasu, "Vulnerability assessment of cybersecurity for SCADA systems using attack trees," *2007 IEEE Power Eng. Soc. Gen. Meet. PES*, vol. 2, pp. 1–8, 2007, doi: 10.1109/PES.2007.385876.

[44] B. Schneier, "Attack Trees," in *Dr. Dobb's Journal*, 1999, [Online]. Available: https://www.schneier.com/academic/archives/1999/12/attack_trees.html.

[45] G. Díaz and J. R. Bermejo, "Static analysis of source code security: Assessment of tools against SAMATE tests," *Inf. Softw. Technol.*, 2013, doi: 10.1016/j.infsof.2013.02.005.

[46] D. Baca, B. Carlsson, K. Petersen, and L. Lundberg, "Improving software security with static automated code analysis in an industry setting," *Softw. - Pract. Exp.*, vol. 43, no. 3, pp. 259–279, Mar. 2013, doi: 10.1002/spe.2109.

[47] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Security Testing: A Survey," *Adv. Comput.*, vol. 101, pp. 1–51, 2016, doi: 10.1016/bs.adcom.2015.11.003.

[48] M. Qasaimeh, A. Shamlawi, and T. Khairallah, "Black box evaluation of web application scanners: Standards mapping approach," *J. Theor. Appl. Inf. Technol.*, vol. 96, no. 14, pp. 4584–4596, Jul. 2018.

[49] S. Lipner and M. Howard, "The Trustworthy Computing Security Development Lifecycle," *Annual Computer Security Applications Conference, copatrocinada por IEEE*, 2004. .

[50] Microsoft Corporation, "Agile Development Using Microsoft Security Development Lifecycle," 2010. https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx (accessed Jun. 19, 2017).

[51] C. O. C. Redwood Shores, "Oracle Software Security Assurance," 2011. https://www.oracle.com/support/assurance/index.html.

[52] OWASP Project, "Comprehensive, Lightweight Application Security Process." https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/introduction-to-the-clasp-process (accessed Mar. 15, 2020).

[53] N. Davis, P. L. Miller, W. R. Nichols, and R. C. Seacord, "TSP-Secure," in *Proceedings of the Fourth Annual TSP Symposium*, 2009, pp. 3–8, [Online]. Available: http://resources.sei.cmu.edu/asset_files/ConferencePaper/2009_021_001_298907.pdf.

[54] OWASP Project, "Software Assurance Maturity Model," 2009. http://www.opensamm.org/downloads/SAMM-1.0.pdf.

[55] G. McGraw, B. Chess, and S. Miques, "Building security In maturity model," *2012 Faulkner Information Services*, no. May. pp. 1–61, 2011, doi: 10.1126/science.1130-a.

[56] J. de V. Mohino, J. B. Higuera, J. R. B. Higuera, and J. A. S. Montalvo, "The application of a new secure software development life cycle (S-SDLC) with agile methodologies," *Electron.*, vol. 8, no. 11, 2019, doi: 10.3390/electronics8111218.

[57] L. Williams, G. McGraw, and S. Migues, "Engineering Security Vulnerability Prevention, Detection, and Response," *IEEE Softw.*, vol. 35, no. 5, pp. 76–80, 2018, doi: 10.1109/MS.2018.290110854.

[58] B. De Win, R. Scandariato, K. Buyens, J. Grégoire, and W. Joosen, "On the secure software development process: CLASP, SDL and Touchpoints compared," *Inf. Softw. Technol.*, vol. 51, no. 7, pp. 1152–1171, 2009, doi: 10.1016/j.infsof.2008.01.010.

[59] J. Grégoire, K. Buyens, B. De Win, R. Scandariato, and W. Joosen, "On the secure software development process: CLASP and SDL compared," in *Proceedings - ICSE 2007 Workshops: Third International Workshop on Software Engineering for Secure Systems, SESS'07*, 2007, doi: 10.1109/SESS.2007.7.

[60] G. McGraw, "Software Security: Building Security in," in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2006, doi: 10.1109/ISSRE.2006.43.

[61] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," *Comput. Stand. Interfaces*, vol. 50, no. October 2016, pp. 107–115, 2017, doi: 10.1016/j.csi.2016.10.001.

[62] OWASP, "OWASP Top 10 - The Ten Most Critical Web Application Security Risks," 2017.

[63] OWASP, "Application Security Verification Standard 3.0.1," 2016.

[64] CWE, "CWE - CWE-1026: Weaknesses in OWASP Top Ten (2017) (4.0)," 2020. https://cwe.mitre.org/data/definitions/1026.html (accessed Jul. 17, 2020).

[65] SANS, "Top 25 Software Errors | SANS Institute," *Sans*, 2011. https://www.sans.org/top25-software-errors/ (accessed Jul. 17, 2020).

[66] CVE, "CVE - Common Vulnerabilities and Exposures (CVE)," 2020. https://cve.mitre.org/ (accessed Jul. 17, 2020).

[67] Security Knowledge Framework, "Security Knowledge Framework." https://www.securityknowledgeframework.org/ (accessed Jul. 17, 2020).

[68] OWASP, "OWASP WebGoat Project." https://owasp.org/www-project-webgoat/

(accessed Mar. 07, 2020).

[69] "Damn Vulnerable Web Application (DVWA)." http://www.dvwa.co.uk/ (accessed Mar. 10, 2020).

[70] J. Druin, "InfoSec Reading Room Introduction to the OWASP Mutillidae II Web," *SANS Inst. InfoSec Read. Room*, 2013, [Online]. Available: https://www.sans.org/reading-room/whitepapers/infosec/introduction-owasp-mutillidae-ii-web-pen-test-training-environment-34380.

[71] S. Davis, "Threat Modeling with STRIDE," *Webtrends*, 2015, [Online]. Available: https://www.webtrends.com/blog/2015/04/threat-modeling-with-stride/.

[72] FIRST, "Common Vulnerability Scoring System v3.0: Specification Document," *Forum Incid. Response Secur. Teams*, pp. 1–21, 2015, [Online]. Available: https://www.first.org/cvss/specification-document.

Bloque IV

Lista de Publicaciones

Apéndice A

A. Publicaciones relacionadas con la tesis

Los resultados de esta Tesis Doctoral se han publicado en artículos de revistas internacionales, congresos nacionales e internacionales revisados por pares y capítulos de libro. Se presentan, siguiendo el orden cronológico, las contribuciones científicas desglosadas en los apartados anteriores.

A.1 Artículos en revistas internacionales

1. José Carlos Sancho Núñez, Andrés Caro Lindo and Pablo García Rodríguez. *A Preventive Secure Software Development Model for a software factory: a case study*. IEEE Access. 8 (1):77653–77665. April 2020. DOI: 10.1109/access.2020.2989113.

   (Impact Factor = 3.745 -Q1)

2. José Carlos Sancho Núñez, Andrés Caro Lindo, Mar Ávila Vegas and Alberto Bravo Gómez. *New approach for threat classification and security risk estimations based on security event management.* Future Generation Computer Systems. 113:488–505. December 2020. DOI: 10.1016/j.future.2020.07.015.

   (Impact Factor = 6.125 -Q1)

A.2 Congresos nacionales e internacionales revisados por pares

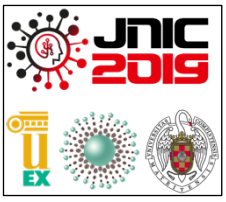
3. José Carlos Sancho Núñez, Andrés Caro Lindo, Pablo García Rodríguez. *Análisis de metodologías de Desarrollo de Software Seguro.* Actas de las II Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2016). 2016. Páginas: 42–47. Granada, España, 15-17 de junio de 2016.

4. José Carlos Sancho Núñez, Andrés Caro Lindo, Pablo García Rodríguez, Ángel Quesada Canabal. *Categorización de Actividades de Seguridad en el Desarrollo de Software.* Actas de las Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2016) [Online]. Available: https://biblioteca.sistedes.es/wp-content/uploads/2016/08/JISBD_2016_paper_70.pdf . Salamanca, España, 14-16 de septiembre de 2016.

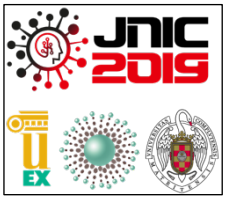
5. José Carlos Sancho Núñez, Andrés Caro Lindo, Pablo García Rodríguez, José Andrés Félix de Sande. *Metodología de Implantación Empresarial de un Modelo de Desarrollo de Software Seguro.* Actas de las III Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2017). 2017. Páginas: 128–133. Madrid, España, 31 mayo - 2 de junio de 2017.

6. José Andrés Félix de Sande, José Carlos Sancho Núñez, Andrés Caro Lindo. *Evaluación y selección de un ecosistema de herramientas para un enfoque preventivo y continuo en modelos de desarrollo seguro de software.* Actas de las IV Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2018). 2018. Páginas: 87–94. San Sebastián, España, 14 – 16 de junio de 2018.

7. José Carlos Sancho Núñez, Andrés Caro Lindo, Lucio Fondón, José Andrés Félix de Sande. *Herramienta para la identificación de requisitos de seguridad en un Modelo de Desarrollo Seguro.* Actas de la XV Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2018). 2018. Páginas: 92–95. Granada, España, 3-5 de octubre de 2018.



8. José Carlos Sancho Núñez, Andrés Caro Lindo, Pablo García Rodríguez, José Andrés Félix de Sande. *Modelo Emergente Preventivo para producir software seguro.* Actas de las V Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2019). 2019. Páginas: 151–158. Cáceres, España, 5-7 de junio de 2019.



9. Alberto Bravo Gómez, José Carlos Sancho Núñez, Andrés Caro Lindo. *Extracción de conocimiento a partir de fuentes de datos reales procedentes de la monitorización de eventos de seguridad.* Actas de las V Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2019). 2019. Páginas: 86–93. Cáceres, España, 5-7 de junio de 2019.

10. José Carlos Sancho Núñez, María Luz Castaño, Andrés Caro Lindo, José Andrés Félix de Sande, Pablo García Rodríguez, Alberto Bravo Gómez. *Herramienta de entrenamiento para el desarrollo de software seguro.* Actas de las XXIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2019). 2019. Páginas: 1–4, [Online]. Available: https://biblioteca.sistedes.es/articulo/herramienta-de-entrenamiento-para-el-desarrollo-de-software-seguro. Cáceres, España, 2-4 de septiembre de 2019.

A.3 Capítulos de libro

11. José Carlos Sancho Núñez, "*Un Modelo particularizado para el Desarrollo de Software Seguro*" in Catálogo de Investigación Joven en Extremadura. Volumen III, In press, 2020.

12. José Carlos Sancho Núñez, Andrés Caro Lindo, José Méndez Chaves, Pablo García Rodríguez, "*Taxonomy of threats, vulnerabilities and good practices for the secure software development*" in Experiencias Innovadoras de Investigación para un Futuro Sostenible, Octaedro, 2020.

B. Actividades de transferencia y otras publicaciones

Al margen de los resultados obtenidos en la Tesis Doctoral, aunque derivados de su conocimiento, se han realizado las actividades de difusión y transferencia que se exponen a continuación. Además, algunas de estas han concluido en otras publicaciones docentes.

B.1 Actividades de transferencia y difusión.

13. Participación en el MOOC *"Investigación en Informática Forense y Ciberderecho"* impartido a través de la plataforma *MiríadaX*. El MOOC dirigido a profesionales del mundo del derecho e investigadores en informática forense, ofrece una panorámica general de las técnicas que se llevan a cabo en los procesos de investigación tecnológica, teniendo siempre cuenta los aspectos legales. Con dos ediciones tuvo una participación superior a 6500 usuarios inscritos. Vídeo de presentación del MOOC.

14. Coorganizador de las dos ediciones de *ForoCiber*, único evento de la Comunidad Autónoma de Extremadura que trata las temáticas del Derecho Tecnológico y la Ciberseguridad con expertos de gran prestigio.

15. Participación en cuatro ediciones del Curso de Experto Profesional en "*Derecho Tecnológico e Informática Forense*" impartido como título propio de la Universidad de Extremadura.

B.2 Publicaciones docentes relacionadas.

16. José Carlos Sancho Núñez, Andrés Caro Lindo, Miguel Sánchez Cabrera, "*Las claves de éxito de un MOOC con una tasa de finalización superior al 25%"* in I Ágora Internacional De Educación, Investigación y Empleo, 2019.

17. José Carlos Sancho Núñez, Andrés Caro Lindo, José Andrés Félix De Sande, Laura Martín Sánchez, "*Cybersecurity Challenge: Detección de talento en ciberseguridad mediante una competición virtual de Capture The Flag*" in IV Jornadas Nacionales De Investigación En Ciberseguridad, 2019, pp. 51–52. San Sebastián, España, 14 – 16 de junio de 2018.

18. Andrés Caro Lindo, José Carlos Sancho Núñez, Mar Ávila Vegas, Miguel Sánchez Cabrera, *"MOOC 'Investigación En Informática Forense Y Ciberderecho', experiencia y resultados.* Actas de las V Jornadas Nacionales de investigación en Ciberseguridad (JNIC 2019). 2019. Páginas: 133–134. Cáceres, España, 5-7 de junio de 2019.

19. Laura Martín Sánchez, José Carlos Sancho Núñez, Arturo Durán Domínguez, "*Capture The Flag: Prácticas de ciberseguridad mediante técnicas e-learning*," in I Congreso Internacional de Campus Digitales en Educación Superior, 2018, pp. 113–115, [Online]. Available: http://dehesa.unex.es/bitstream/handle/10662/8659/978-84-09-07196-8.pdf.