UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

MAESTRÍA EN GERENCIA DE SISTEMAS DE INFORMACIÓN

Tema: "ANÁLISIS DE LA METODOLOGÍA SDL PARA SU APLICACIÓN EN EL DESARROLLO DE SOFTWARE SEGURO EN INSTITUCIONES FINANCIERAS"

Trabajo de Investigación, previo a la obtención del Grado Académico de Magister en Gerencia de Sistemas de Información

Autor: Ing. Ruth Elizabeth Aleaga Guaigua

Director: Ing. Guevara Aulestia David Omar, Mg.

Ambato – Ecuador

2019

A la Unidad Académica de Titulación de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial

El Tribunal receptor del Trabajo de Investigación presidido por la Ingeniera Elsa Pilar Urrutia Magister e integrado por los señores Ingeniero Franklin Oswaldo Mayorga Mayorga Magister, Ingeniero Hernán Fabricio Naranjo Avalos Magister, Ingeniero Carlos Israel Núñez Miranda Magister, designados por la Unidad Académica de Titulación de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato, para receptar el Trabajo de Investigación con el tema: "**ANÁLISIS DE LA METODOLOGÍA SDL PARA SU APLICACIÓN EN EL DESARROLLO DE SOFTWARE SEGURO EN INSTITUCIONES FINANCIERAS**", elaborado y presentado por la Ingeniera Ruth Elizabeth Aleaga Guaigua, para optar por el Grado Académico de Magister en Gerencia de Sistemas de Información; una vez escuchada la defensa oral del Trabajo de Investigación el Tribunal aprueba y remite el trabajo para uso y custodia en las bibliotecas de la UTA.

Ing. Elsa Pilar Urrutia Mg.

Presidente del Tribunal

Ing. Franklin Oswaldo Mayorga Mayorga Mg.

Miembro del Tribunal

Ing. Hernán Fabricio Naranjo Avalos Mg.

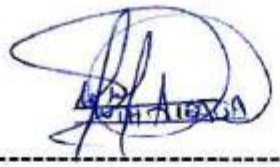
Miembro del Tribunal

Ing. Carlos Israel Núñez Miranda Mg.

Miembro del Tribunal

AUTORÍA DEL TRABAJO DE INVESTIGACIÓN

La responsabilidad de las opiniones, comentarios y críticas emitidas en el Trabajo de Investigación presentado con el tema: "**ANÁLISIS DE LA METODOLOGÍA SDL PARA SU APLICACIÓN EN EL DESARROLLO DE SOFTWARE SEGURO EN INSTITUCIONES FINANCIERAS**", le corresponde exclusivamente a: Ingeniera Ruth Elizabeth Aleaga Guaigua, Autora bajo la Dirección del Ingeniero Guevara Aulestia David Omar Mg., Director del Trabajo de Investigación; y el patrimonio intelectual a la Universidad Técnica de Ambato.



Ing. Ruth Elizabeth Aleaga Guaigua

c.c. 1804010864

AUTORA

Ing. Guevara Aulestia David Omar, Mg.

c.c. 1802605749

DIRECTOR

DERECHOS DE AUTORA

Autorizo a la Universidad Técnica de Ambato, para que el Trabajo de Investigación, sirva como un documento disponible para su lectura, consulta y procesos de investigación, según las normas de la Institución.

Cedo los Derechos de mi trabajo, con fines de difusión pública, además apruebo la reproducción de este, dentro de las regulaciones de la Universidad.

Ing. Ruth Elizabeth Aleaga Guaigua

c.c. 1804010864

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

AGRADECIMIENTO

Agradezco a Dios por permitirme cumplir mis sueños, brindándome salud, vida e inteligencia.

A mis padres, a mis primos que son para mí como mis hermanos; por estar siempre ahí apoyándome y alentándome con sus consejos y no dejando que desmaye en las cosas que me he propuesto. A esa persona especial que amo y que me ha impulsado día a día a cumplir con mi objetivo.

A mi director de tesis Ing. David Guevara que ha sabido guiarme desde un inicio para concluir exitosamente con mi trabajo de investigación.

DEDICATORIA

Dedico el presente trabajo a Dios por las oportunidades que me ha dado en la vida y que a pesar de las circunstancias las he sabido aprovechar.

A mi familia que siempre ha estado a mi lado y que quiero mucho. Deseo que Dios les bendiga siempre.

Ruth Aleaga

UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL
DIRECCIÓN DE POSGRADO
MAESTRÍA EN GERENCIA DE SISTEMAS DE INFORMACIÓN

TEMA:

"ANÁLISIS DE LA METODOLOGÍA SDL PARA SU APLICACIÓN EN EL DESARROLLO DE SOFTWARE SEGURO EN INSTITUCIONES FINANCIERAS"

AUTOR: Ing. Ruth Elizabeth Aleaga Guaigua
DIRECTOR: Ing. Guevara Aulestia David Omar, Mg.
FECHA: 08 de noviembre del 2018

RESUMEN EJECUTIVO

La Cooperativa OSCUS se ha caracterizado por ser una institución sólida y reconocida dentro de la ciudad de Ambato, los sistemas que maneja son desarrollados por el personal de la misma institución por lo cual es significativo que se aplique seguridad en el código fuente, para ellos se ha propuesto la implementación de la metodología SDL, que de acuerdo a sus características se ajusta a las necesidades de la institución y con el equipo de desarrollo que posee la Cooperativa se puede emprender la implementación.

La seguridad en los proyectos de desarrollo juega un papel importante, al mismo tiempo que la funcionalidad de los sistemas. Seguir una metodología de desarrollo de software seguro que sea ágil e incremental, permite a los programadores incrementar su productividad, reduciendo tiempo empleado en los proyectos y costes de los mismos.

De acuerdo a las fases que tiene la metodología SDL se puede ir analizando cada una de ellas e ir identificando amenazas, riesgos, vulnerabilidades, las cuales pueden ser corregidas a tiempo, se comprueba la aplicación de buenas prácticas de desarrollo y todo el proceso del ciclo de vida que un desarrollo de software requiere. Además, con la utilización de herramientas de análisis de código se puede llegar a identificar amenazas de forma rápida de tal manera que al corregirlas se consigue asegurar el software.

La propuesta de implementación va dirigida a mejorar la seguridad del software que se desarrolla dentro de la Cooperativa, junto con el cumplimiento del ciclo de vida de software. Si bien es cierto no se puede obtener seguridad en la totalidad de los sistemas, puesto que cada vez aparecen nuevas amenazas de acuerdo a lo que avanza la tecnología, pero se pretende reducir el nivel de riesgo de ataques y de esta manera proteger la integridad de los datos.

Descriptores: equipo de desarrollo, productividad, proyectos de desarrollo, amenazas, vulnerabilidades, costos de proyectos, buenas prácticas de desarrollo, ciclo de vida, análisis de código, seguridad del software, nivel de riesgo.

UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL
DIRECCIÓN DE POSGRADO
MAESTRÍA EN GERENCIA DE SISTEMAS DE INFORMACIÓN

THEME:

"ANALYSIS OF THE SDL METHODOLOGY FOR ITS APPLICATION IN THE DEVELOPMENT OF SAFE SOFTWARE IN FINANCIAL INSTITUTIONS"

AUTHOR: Ing. Ruth Elizabeth Aleaga Guaigua
DIRECTED BY: Ing. Guevara Aulestia David Omar, Mg.
DATE: 08 de noviembre de 2018

EXECUTIVE SUMMARY

The OSCUS Cooperative has been characterized as a solid and recognized institution within the city of Ambato, the systems it manages are developed by the staff of the same institution, which is why it is significant that security is applied to the source code, for them it is has proposed the implementation of the SDL methodology, which according to its characteristics is adapted to the needs of the institution and with the development team that the Cooperative has, implementation can be undertaken.

Security in development projects plays an important role at the same time as the functionality of the systems. Following a secure software development methodology that is agile and incremental allows programmers to increase their productivity, reducing time spent on projects and costs thereof.

According to the phases of the SDL methodology, each one of them can be analyzed and threats, risks and vulnerabilities identified, which can be corrected in time, the application of good development practices and the whole cycle process is checked of life that a software development requires. In addition, with the use of code analysis tools it is possible to identify threats quickly in such a way that correcting them ensures that the software is secured.

The implementation proposal is aimed at improving the security of the software that is developed within the Cooperative, along with compliance with the software life cycle. Although it is true that security can not be obtained in all the systems, as new threats appear every time according to the progress of the technology, but it is intended to reduce the level of attack risk and thus protect the integrity of the systems data.

Descriptors: development team, productivity, development projects, threats, vulnerabilities, project costs, good development practices, life cycle, code analysis, software security, risk level.

INTRODUCCIÓN

La seguridad del software se ha vuelto un tema muy sensible y de gran interés en nuestro medio. En especial en Instituciones Financieras que son establecimientos críticos en donde su principal actividad es el manejo de dinero.

Según la era digital va avanzando se hace más complejo el aplicar seguridades a los sistemas informáticos, electrónicos y digitales. Se debe tratar de identificar la mayoría de problemas de seguridad que existen y solucionarlos en su momento.

El hecho de encontrarse los sistemas expuestos en la red pública como es el Internet, implica ya un riesgo potencial. Principalmente los servicios de Banca en Línea que cada Institución Financiera brinda a sus Socios/Clientes.

La plataforma y el desarrollo de software con los que están creados los sistemas son primordiales. Por tanto, se deben aplicar metodologías de desarrollo de software seguro para minimizar el riesgo de ataques. SDL (Secure Development Lifecycle) es una metodología que puede ayudar a construir código seguro.

El CAPÍTULO I, EL PROBLEMA contiene: el tema de investigación, el planteamiento del problema, su contexto, análisis crítico, prognosis, formulación del problema, interrogantes, delimitación, justificación y objetivos.

El CAPÍTULO II MARCO TEÓRICO contiene: antecedentes de la investigación, fundamentación filosófica, fundamentación legal, categorías fundamentales, hipótesis y señalamiento de variables.

El CAPÍTULO III METODOLOGÍA contiene: el enfoque de investigación, modalidad básica de la investigación, nivel o tipo de investigación, población y muestra, operacionalización de variables, plan de recolección de información y plan de procesamiento de la información.

El CAPÍTULO IV MARCO ADMINISTRATIVO contiene: Los recursos requeridos, cronograma, bibliografía y los respectivos Anexos.

CAPÍTULO I

1. EL PROBLEMA DE INVESTIGACIÓN

1.1 Tema de Investigación

Análisis de la metodología SDL para su aplicación en el desarrollo de software seguro en Instituciones Financieras.

1.2 Planteamiento del Problema

1.2.1 Contextualización

La seguridad en el desarrollo de software se ha visto como un tema de investigación de suma importancia. Cada vez, lo que se busca en los sistemas es que tengan incorporados aspectos de seguridad y más si se encuentran expuestos en la red Internet.

A nivel general en el mundo entero se observa como la seguridad en el desarrollo de software va siendo un aspecto fundamental y se busca las características como son:

- La confidencialidad del software que permite que cierto contenido se accesible para unos usuarios e inaccesibles para otros.
- La integridad que el código no pueda ser modificado de manera incorrecta.
- La disponibilidad que el software esté accesible y operativo todo el tiempo que el usuario lo requiera.

En el Ecuador, unas instituciones financieras desarrollan sus propios sistemas financieros, los cuales son desarrollados por el mismo personal de la institución por tanto son dueños de su código, otras instituciones adquieren los servicios de proveedores externos o a su vez el consumo de servicios web para su funcionamiento. Estos sistemas constituyen el crecimiento de las instituciones, por lo que la seguridad con la que están desarrollados eleva la integridad de la

información de Socios/Clientes, así como de las transacciones que se realizan diariamente.

En Tungurahua también se observa que existen instituciones financieras que, aunque sean pequeñas realizan la misma actividad, por tanto, la seguridad con la que el software es desarrollado debe ser tomado en cuenta al momento de programar, de tal manera que se pueda aplicar medidas preventivas o correctivas dependiendo la situación actual. El desarrollo en algunos casos suele ser de forma tradicional, sin la aplicación de buenas prácticas de seguridad con lo que se tiene como resultado un software vulnerable e inseguro. Esto se puede detectar con la ayuda de técnicas de Hacking Ético el cual nos muestra cuales son los puntos débiles de los sistemas sobre todo en los que se encuentras expuestos en el internet.

Las instituciones financieras manejan servicios en línea los cuales requieren sean seguros, como son las transferencias interbancarias en las que existe un mayor riesgo de ataques debido a que éstas se efectúan a través del Internet. A pesar de que los sitios web poseen certificados de seguridad, esto no garantiza que la información viaje segura. Se puede lograr que la información sea segura aplicando buenas prácticas de seguridad, así como la aplicación de criptografía al momento de desarrollar aplicativos.

1.2.2 Análisis Crítico

En la actualidad se está viviendo ciberataques a todo tipo de empresas incluidas las entidades financieras tanto públicas como privadas, estos ataques van encaminados a los sistemas informáticos, por tanto no se puede pasar por alto los temas de seguridad al momento de desarrollar software, por mínimo que parezca. Estos deben ser desarrollados con alguna metodología que ayude a mejorar la calidad del software.

La falta de seguridad en el desarrollo de software puede debilitar los sistemas enormemente a las entidades que se encuentran tecnológicamente equipadas. Según va creciendo el Internet también va creciendo el riesgo de ataques maliciosos por parte de hackers.

Considerando que los hackers son personas que pueden violentar las seguridades de los sistemas una vez que hayan encontrado alguna debilidad. Se debe aplicar buenas prácticas de desarrollo y codificación segura, para reducir los ataques que pueden ser perjudiciales para las entidades.

Con herramientas de monitoreo y detección de vulnerabilidades se pueden detectar brechas en los sistemas, especialmente los que viajan a través del Internet. Son de gran ayuda para tomar medidas correctivas en caso de ser necesario.

Actualmente los desarrolladores deben enfocarse en temas de seguridad y en los atacantes, ya no solo en dar atención a los requerimientos de usuarios. Para esto al momento de desarrollar software se debe contar con una metodología que permita incluir buenas prácticas de seguridad tanto al código como a los recursos críticos de software.

1.2.3 Prognosis

El desarrollar software sin aplicar algún proceso de control en el código nos conlleva a ser vulnerables a los ataques por parte de hackers o alguna aplicación maliciosa. Se puede sufrir pérdidas de información sensible que posee las instituciones financieras.

Al tratarse de instituciones financieras, los riesgos son potenciales, puesto que la actividad principal del sistema financiero son las transferencias interbancarias. Si los sistemas desarrollados para este fin no contaren con las debidas seguridades, se pierde credibilidad por parte de los Socios/Clientes que se pueden ver afectados

en el caso de que sus transacciones no lleguen a ser efectivas y se sientan perjudicados.

Si no se aplica una adecuada metodología para controlar la seguridad orientada al desarrollo de software, se corre el riesgo de que las instituciones sean vulnerables a los ataques informáticos lo cual conllevaría a ser menos competitivos en la sociedad y ante los clientes ya que no se brindaría un servicio de calidad.

1.2.4 Formulación del Problema

¿Incide el análisis de la metodología SDL en el desarrollo de software seguro en las Instituciones Financieras?

1.2.5 Interrogantes (Subproblemas)

- ¿Se hace uso de principios y buenas prácticas para lograr una codificación segura en el desarrollo de software?
- ¿Cuáles son los procesos del ciclo de vida de desarrollo seguro SDL (Secure Development Lifecycle)?
- ¿Se puede aplicar la metodología SDL en software existente para corregir las vulnerabilidades principalmente en los servicios online?

1.2.6 Delimitación del objeto de investigación

Campo: Instituciones Financieras
Área: Departamento de Tecnología de Información y comunicación (TIC)
Aspecto: Tecnología, análisis de la metodología SDL

1.2.6.1 Delimitación Espacial:

Cooperativa de Ahorro y Crédito Oscus Ltda.

1.2.6.2 Delimitación Temporal:

El desarrollo del proyecto se realizara desde enero a junio 2018

1.2.6.3 Unidades de Observación:

Departamento de TIC

1.3 Justificación

En las Instituciones Financieras existen sistemas que manejan información sensible y que se requiere que ésta sea segura y confiable. Muchos de estos sistemas pueden ser vulnerables a ataques, como es el caso de las aplicaciones Web que pueden ser accedidas desde cualquier navegador en cualquier parte de mundo.

A pesar de contar con protocolos de seguridad como por ejemplo SSL que en cierta forma protegen de los atacantes, se requiere que el código fuente también sea seguro. Esto se lo puede conseguir con la aplicación de la metodología SDL (Secure Development Lifecycle) que es el Ciclo de Desarrollo Seguro.

El desarrollo de software seguro involucra aplicar validaciones de entradas, lo que significa es que se debe controlar la información que el usuario ingresa antes de ser procesada. Controles de autenticación a los sistemas y/o aplicaciones Web en el caso de que la información sea privada. Uso de prácticas de Criptografía para brindar confidencialidad, proteger la información del usuario y que ésta sea íntegra. Manejo personalizado de excepciones, esto significa no revelar información en las respuestas de error, estos mensajes deben se genéricos.

En las Instituciones Financieras que administran su propio código fuente tienen una gran ventaja como es el de aplicar este tipo de metodología. En el manejo de código se puede emplear defensas estratificadas siempre que sea posible. SDL ayudará a reducir el número y la gravedad de las vulnerabilidades en el software, que los sistemas sean más restringidos y que no sean de fácil acceso por parte de atacantes.

Según lo redactado y de acuerdo a las situaciones que se exponen las instituciones financieras, este proyecto de investigación es factible debido a que se cuenta con

herramientas, recursos humanos, económicos y accesibilidad a la información para cumplir con el objetivo planteado.

1.4 Objetivos

1.4.1 Objetivo General

Analizar la metodología SDL para su aplicación en el desarrollo de software de forma que mejore la calidad del software de las Instituciones Financieras.

1.4.2 Objetivos Específicos:

- Investigar sobre conceptos referente a la metodología SDL (Secure Development Lifecycle), ventajas, beneficios y aplicación.
- Analizar las fases de la metodología SDL enfocada al desarrollo de software seguro.
- Identificar las vulnerabilidades en las aplicaciones Web que utilizan las Instituciones Financieras mediante el uso de herramientas de monitoreo.
- Elaborar una propuesta de implementación de la metodología SDL en aplicaciones Web para determinar su factibilidad y mitigación de ataques.

CAPÍTULO II

2 MARCO TEÓRICO

2.1 Antecedentes de Investigativos

Revisadas investigaciones relacionadas al desarrollo de software seguro y metodología SDL, se mencionan las siguientes:

Según Martha Ascencio Mendoza y Pedro Julián Moreno Patiño (Ascencio Mendoza & Moreno Patiño, 2011) en la investigación "Desarrollo de una Propuesta Metodológica para Determinar la Seguridad en una Aplicación Web.". Proyecto realizado en la Universidad Tecnológica de Pereira en el año 2011 en la que trata sobre la seguridad en el entorno Web. En este proyecto la autora realiza un estudio sobre vulnerabilidades en las aplicaciones Web. Las herramientas que se pueden utilizar para detectar estas vulnerabilidades y además analiza las metodologías de seguridad para aplicaciones Web existentes, en dicha investigación recomienda que:

- Se debe incluir dentro de la Política de Seguridad las políticas de Programación Segura, para mitigar las vulnerabilidades.
- También incluir dentro de los equipos existentes en el proceso de desarrollo al equipo de seguridad, quienes estarán presentes en cada una de las fases del proceso de desarrollo contribuyendo con los controles de seguridad.

Según Ericka Jamin Robles Gomez (Robles Gomez, 2011) en el estudio "Ciclo de Vida de Desarrollo de Software Seguro en Metodologías Ágiles", en el año 2011. Propone la inclusión de desarrollo de software seguro (SDL) en Metodologías Ágiles (SCRUM). Considera que es muy importante que los programadores se enfoquen más en los atributos de calidad, con la finalidad de que durante el desarrollo del software se vaya mitigando los riesgos o vulnerabilidades de el/los sistemas y no cuando ya se encuentra

a punto de lanzar el producto. Concluye que para tener éxito sobre la aplicación de un método de desarrollo seguro, se debe concientizar y divulgar al personal de la empresa donde se vaya a implementar.

En un artículo publicado por Marta Castellaro, Susana Romaniz, Juan Ramos y Pablo Pessolani (Castellaro, Romaniz, Ramos, & Pessolani, 2009) con el tema "Hacia la Ingeniería de Software Seguro", en la Facultad Regional Santa Fe – Universidad Tecnológica Nacional. La autora reconoce que actualmente ya no solo es necesario cumplir con los requerimientos del usuario al momento de desarrollar software si no que se debe garantizar que el mismo sea seguro. Manifiestan que no se debe tratar de manera independiente la Ingeniería de Software y la Ingeniería de Seguridad, analizan metodologías que abarque la seguridad y el proceso de ciclo de vida de desarrollo de software, por tanto, unen estos conceptos en una sola disciplina denominada Ingeniería de Software Seguro.

Según José Rolando Solano Campos (Solano Campos, 2015) en su tesis de Posgrado con el tema "Implementación de la metodología Six Sigma SDLC para la mejora de la calidad del proceso de desarrollo web" de la Universidad Nacional, Heredia - Corta Rica 2015. Realiza la combinación del método estadístico Six Sigma y el desarrollo de software que poseen similitud en los pasos de Six Sigma y las fases de SDL, que resulta muy bueno al utilizar herramientas adecuadas, ayuda a mejorar el desempeño del negocio dirigido a la satisfacción del cliente, reducir costos, así como generar un mecanismo fundamental de desarrollo.

2.2 Fundamentación Filosófica

El presente proyecto de investigación utiliza el paradigma Critico Propositivo, es crítico porque cuestiona el origen del problema, y es propositivo porque plantea una solución al problema propuesto.

2.3 Fundamentación Legal

El presente proyecto de investigación se sustenta en lo dispuesto por la Superintendencia de Economía Popular y Solidaria en la RESOLUCIÓN No. SEPS-IGT-IR-IGJ-2018-0279 y el reglamento de Gestión de Tecnología de la Cooperativa de Ahorro y Crédito OSCUS Ltda.:

RESOLUCIÓN No. SEPS-IGT-IR-IGJ-2018-0279 (SEPS, 2018)

NORMA DE CONTROL PARA ADMINISTRACIÓN DE RIESGO OPERATIVO Y RIESGO LEGAL EN LAS ENTIDADES DEL SECTOR FINANCIERO POPULAR Y SOLIDARIO BAJO EL CONTROL DE LA SUPERINTENDENCIA DE ECONOMÍA POPULAR Y SOLIDARIA

Artículo 9.3.- Políticas, procesos, procedimientos y metodologías para la administración de la tecnología de información:

9.3.2.- Las entidades señaladas deberán garantizar que el proceso de adquisición, desarrollo, implementación y mantenimiento de las aplicaciones satisfagan los objetivos del negocio, considerando al menos lo siguiente:

- **a)** Una metodología que permita la adecuada administración y control del proceso de compra de software y del ciclo de vida de desarrollo y mantenimiento de aplicaciones, con los usuarios involucrados;
- **b)** Requerimientos funcionales aprobados por el área solicitante;
- **c)** Requerimientos técnicos y análisis de la relación y afectación a la capacidad de la infraestructura tecnológica actual, aprobados por el área técnica;
- **d)** Ambientes de prueba, desarrollo y producción, con la debida segregación de accesos. Para el caso de entidades que hayan tercerizado el servicio de desarrollo de sistemas, deberán contar al menos con ambientes de prueba y producción;
- **e)** Mitigación de las vulnerabilidades del código fuente de las aplicaciones;

f) Pruebas técnicas y funcionales que reflejen la aceptación de los usuarios autorizados;

g) Procedimientos de control de cambios que considere su registro, manejo de versiones, segregación de funciones y autorizaciones e incluya los cambios emergentes; y,

h) Procedimientos de migración de la información, que incluyan controles para garantizar las características de integridad, disponibilidad y confidencialidad.

REGLAMENTO DE LA GESTIÓN DE TECNOLOGÍA DE LA INFORMACIÓN Y COMUNICACIÓN INFORMATICA (OSCUS)

TíTULO IV
ARQUITECTURA Y REGLAS DE SINTAXIS DE DATOS DE DESARROLLO

Capítulo II
REDACCIÓN DEL CÓDIGO FUENTE

Art.63 "Para el desarrollo de aplicativos se utilizará la programación orientada a objetos, servicios, modelos (modelo vista vista modelo MVVM, modelo vista controlador MVC)."

Art.64 "Para la codificación de identificadores se utilizará las siguientes reglas:

a. Estilo Pascal.- La primera del identificador y la primera letra de cada palabra subsiguiente irán en mayúsculas. Este estilo se usará para la codificación de: Clases, Tipos de datos enumerados, Clases de Excepción, Interfaz, Métodos, Espacio de nombres y Propiedades.
b. Estilo Camel.- La primera letra de un identificador con minúscula y la primera letra de cada palabra subsiguiente irá en mayúscula. Este estilo se usará para la codificación de parámetros.
c. Estilo Mayúsculas.- Todas las letras de un identificador en mayúscula. Este estilo se usará para la codificación de valores enumerados.

d. No se usarán mayúsculas para diferenciar parámetros, propiedades ni métodos.
e. No se usarán abreviaturas o contracciones como parte del nombre de identificadores y parámetros.
f. No se utilizarán acrónimos a menos que sean ampliamente conocidos para reemplazar nombres largos o frases.
g. Se deberá usar nombres descriptivos que sean una combinación de una o más palabras que ayuden a determinar el propósito o finalidad del identificador dentro de su contexto.
h. No se usará como nombres de identificadores las palabras reservadas del lenguaje de programación que se utilice.
i. Se utilizará espacios de nombres en singular y plural siempre que sea semánticamente apropiado."

Capítulo III
REDACCIÓN DE CLASES

Art.65 "Para nombrar las clases que definen el comportamiento de un objeto se utilizará un sustantivo o combinación de sustantivos en estilo Pascal, sin utilizar prefijos de tipo ni caracteres de subrayado, además se evitará utilizar abreviaciones."

Art.66 "Los campos de una clase se declararán como privados, no se deberán usar campos públicos o protegidos."

Art.67 "Se deberá usar la palabra reservada const para declara campos constantes."

Art.68 "Se utilizará clases base para agrupar subclases que comparten un conjunto común de funcionalidades."

Art.69 "Para los aplicativos con framework inferior a la versión 2.5 se creará un constructor para cada clase a fin de que el compilador no genere un constructor por emisión que posteriormente podría cambiar la concepción lógica de la clase."

Art.70 "Para los aplicativos con framework inferiores a la vesion 2.5, dentro de la misma clase se utilizará siempre la palabra clave **this** para calificar miembros de

la instancia. Y siempre se utilizará el nombre de la clase para calificar a miembros estáticos."

Art.71 "Siempre se deberá proporcionar propiedades de acceso **get** y **set** para los campos en lugar de hacerlos públicos y solo lectura (get) cuando el usuario no pueda cambiar el valor del miembro de datos subyacentes."

Capítulo IV
REDACCIÓN DE METODOS

Art.72 "Los nombres de los parámetros deberán ser lo suficientemente descriptivos para determinar el significado en estilo Camel comenzando con las palabras un o una."

Art.73 "Los métodos se describirán utilizando verbos o frases adverbiales en estilo Pascal."

Art.74 "Los métodos evitaran generación de errores o excepciones en la ejecución de los mismos."

Art.75 "Se utilizarán métodos sobrecargados para:

a. Proporcionar diferentes métodos que hagan semánticamente la misma operación con diferentes listas de parámetros.
b. Para refinar la funcionalidad de la clase.
c. Para permitir múltiples versiones de la misma operación."

Capítulo V
MANEJO Y ESPECIFICACIÓN DE ERRORES

Art.76 "Todos los caminos de código que resulten en una excepción deben proporcionar un método para verificar el éxito sin emitir la excepción evitando el retorno de códigos de error."

Art.77 "Los errores que generen las aplicaciones deberán se guardados para la monitorización y corrección, especificando claramente la excepción generada."

Art.78 "Los mensajes de error que se presenten al usuario deberán se gramaticalmente correctos sin exponer información privilegiada; limitándose a indicar el error."

Art.79 "En el caso de existir excepciones, se deben proveer mecanismos para que las aplicaciones sigan disponibles al usuario. "

Capítulo VI
COMENTARIOS DEL CÓDIGO FUENTE

Art.80 "Se incluirá en el código comentarios que describan con suficiente nivel de detalle el propósito, función o decisión de implementación para facilitar el mantenimiento de las aplicaciones institucionales desarrolladas, utilizando los siguientes tipos de comentarios:

a. Comentario para bloque: Se utilizará cuando sea necesario comentar un conjunto apreciable de código fuente.
b. Comentario de línea: Deberán ser insertados por el Analista Programador Senior y Junior a fin de aclarar los conceptos o ideas de implementación."

Art.81 "Para los comentarios que hagan referencia al mantenimiento de código que cambia o agrega funcionalidad se considerará lo siguiente:

a. Para los aplicativos con framework inferiores a la versión 2.5. cuando el cambio sea producto de un mantenimiento o nueva función solicitada a través de un requerimiento de soporte técnico, esto se hará constar en la cabecera de la clase.
b. Si se trata de un bloque de código se encerrará entre dos líneas de comentarios identificando la fecha de realización y las iniciales del Analista Programador Senior o Junior que realizó el cambio.
c. Si se trata de un cambio mínimo en una sola línea, al final de esta se pondrá la fecha e iniciales del Analista Programador Senior o Junior que realizó el cambio."

2.4 Categorías Fundamentales

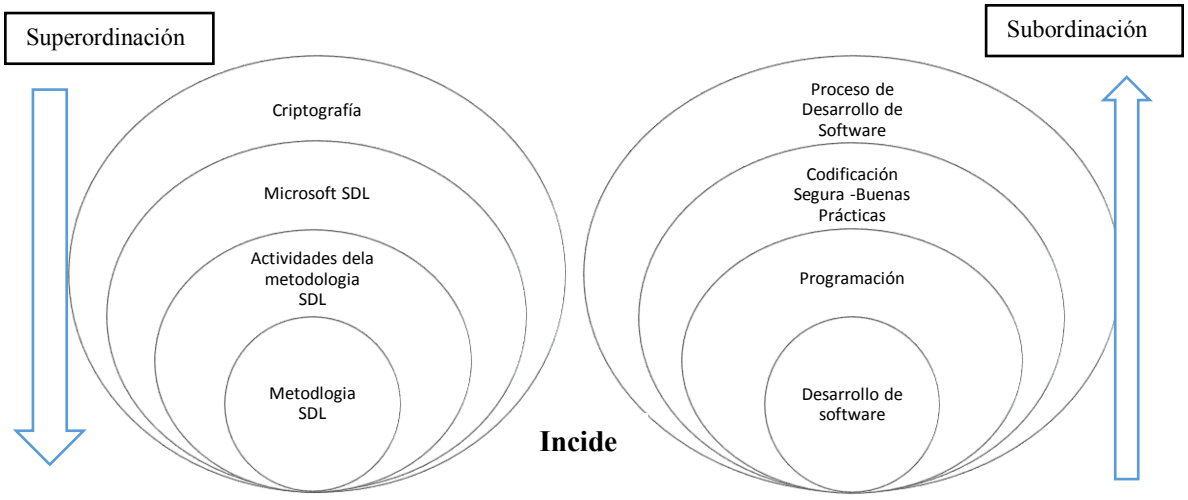


Figura 2.1 Variable Independiente
Elaborado por: Investigador

Figura 2.2 Variable Dependiente
Elaborado por: Investigador

Constelación de Ideas, Mándala Variable Independiente u otros

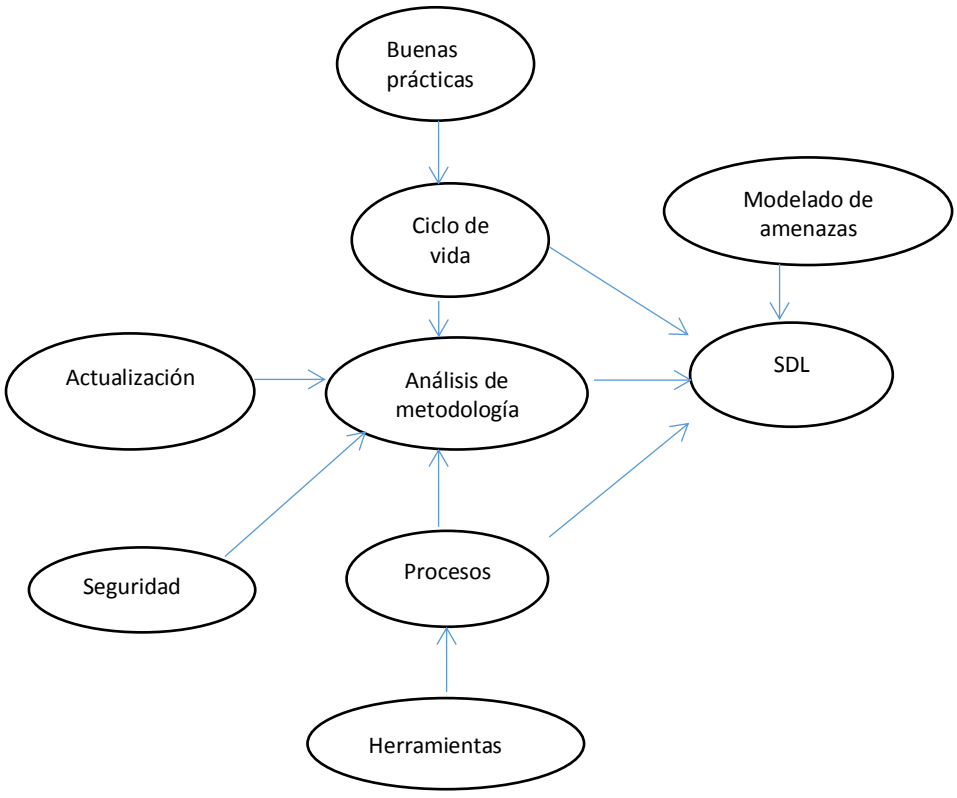


Figura 2.3 Constelación de Ideas de la Variable Independiente

Elaborado por: Investigador

Constelación de Ideas, Mándala Variable Dependiente u otros

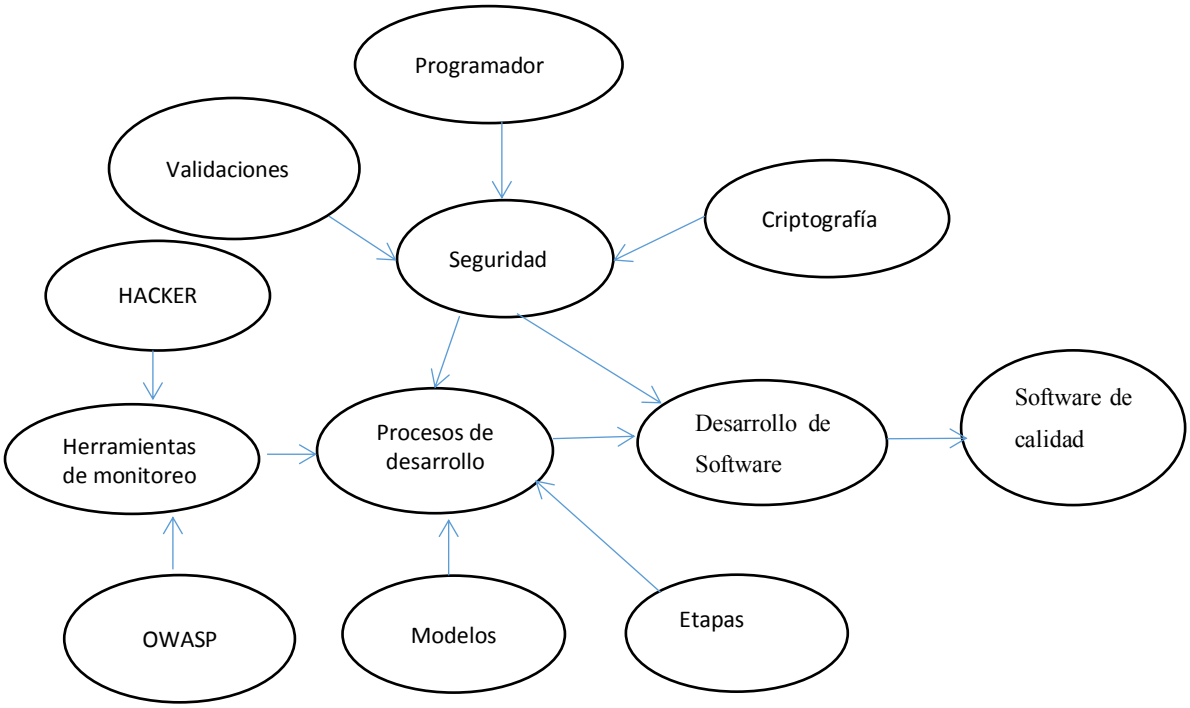


Figura 2.4 Constelación de Ideas de la Variable Dependiente

Elaborado por: Investigador

2.4.1 Categorías de la Variable Independiente

- **Criptografía**

  La criptografía ha evolucionado en la configuración electrónica como un medio para transferir de manera segura información sobre un sistema de comunicación que se presume que es inseguro, como las líneas telefónicas o una red de comunicaciones públicas (por ejemplo, Internet). En este contexto electrónico computarizado, la criptografía proporciona las herramientas necesarias para proteger de forma digital mensajes electrónicos sensibles y valiosos de manera que se asegure la privacidad entre el remitente autenticado y el destinatario autenticado del comunicado, aunque el mensaje está sujeto a interceptación en el sistema de comunicación inseguro. (Terrence R., Jeffrey F., & Daniel R., 1997)

Tipos de Criptografía:

Simétricas:

La criptografía simétrica utiliza la misma clave para cifrar y descifrar el mensaje de datos, es decir se basa en un secreto compartido. Es por esta razón que la seguridad de este proceso depende de la posibilidad de que una persona no autorizada consiga la clave de sesión o clave secreta.

Los algoritmos criptográficos simétricos tienen dos versiones: cifrador en bloque y cifrador en flujo. Una cifra es una palabra para describir un algoritmo de cifrado. Los cifradores en bloque codifican datos en bloques pequeños de longitud fija de 64 bits de longitud. Hay muchos cifradores en bloque que incluyen DES, 3-DES, RC2, RC5, RC6 y Rijndael (conocido como AES) (Mendoza T., 2008).

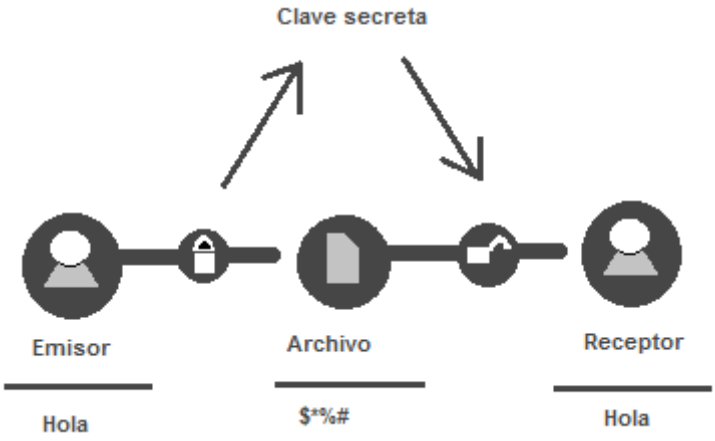


Figura 2.5 Criptografía Simétrica
Elaborado por: Investigador

Asimétrica: Los algoritmos asimétricos son diferentes a los simétricos en un sentido muy importante. Cuando se genera una clave simétrica, simplemente se escoge un número aleatorio de la longitud apropiada. Al generar claves asimétricas el proceso es más complejo.

Los algoritmos asimétricos se llaman asimétricos porque en lugar de usar una sola clave para realizar la codificación y la decodificación, se utilizan dos claves diferentes: una para cifrar y otra para descifrar. Estas dos claves se encuentran

asociadas matemáticamente, cuya característica fundamental es que una clave no puede descifrar lo que cifra.

Cuando se completa la generación de una clave asimétrica se define una clave de cifrado (clave pública) y una clave de descifrado (clave privada); la primera puede ser conocida por todo el mundo, pero, de otro lado se debe tener mucho cuidado en ocultar la clave privada. Las claves asimétricas tienen la sorprendente propiedad de que lo que se está cifrando con una clave sólo se puede descifrar con la otra (Mendoza T., 2008).

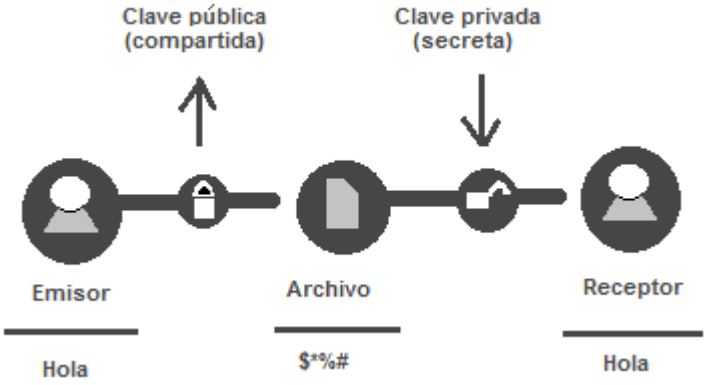


Figura 2.6 Criptografía Asimétrica
Elaborado por: Investigador

- **Microsoft SDL (Secure Development Lifecycle)**

El ciclo de vida de desarrollo de seguridad de Microsoft es un proceso de desarrollo de software utilizado y propuesto por Microsoft para reducir los costos de mantenimiento del software, problemas de seguridad, aumentar la confiabilidad del software y resolver vulnerabilidades de seguridad de manera oportuna. Se basa en el modelo espiral clásico (Implementación simplificada del proceso SDL de Microsoft, 2010).

El proceso SDL de Microsoft se basa en tres conceptos básicos: formación, mejora continua de los procesos y responsabilidad. La formación continua de los roles técnicos dentro de un grupo de desarrollo de software es fundamental. Si se invierte de manera apropiada en la transferencia de los conocimientos, las organizaciones podrán responder adecuadamente a los cambios que sufren las tecnologías y las amenazas. Dado que los riesgos para la seguridad no son estáticos, SDL destaca especialmente la importancia de comprender la causa y el

efecto de las vulnerabilidades de seguridad y requiere una evaluación periódica de los procesos de SDL y la introducción de cambios como respuesta a los avances tecnológicos o las nuevas amenazas. Se recopilan datos para evaluar la eficacia de la formación, se usan métricas de procesos para documentar su conformidad y métricas posteriores al lanzamiento ayudan a definir los futuros cambios. Por último, SDL requiere el archivado de todos los datos necesarios para realizar el mantenimiento de una aplicación en caso de que surjan problemas. Si lo combinan con detallados planes de comunicación y de respuesta en materia de seguridad, las organizaciones podrán orientar de manera concisa y contundente a todas las partes implicadas (Implementación simplificada del proceso SDL de Microsoft, 2010).

Aseguramiento del Software y Aseguramiento de la Calidad

La relación entre SA y Quality Assuarance (QA) es también un tema de discusión entre las organizaciones involucradas con la seguridad del software, el desarrollo de software y la calidad del software; y está siendo considerada desde dos direcciones: (Castellaro, Romaniz, Ramos, & Pessolani, 2009)

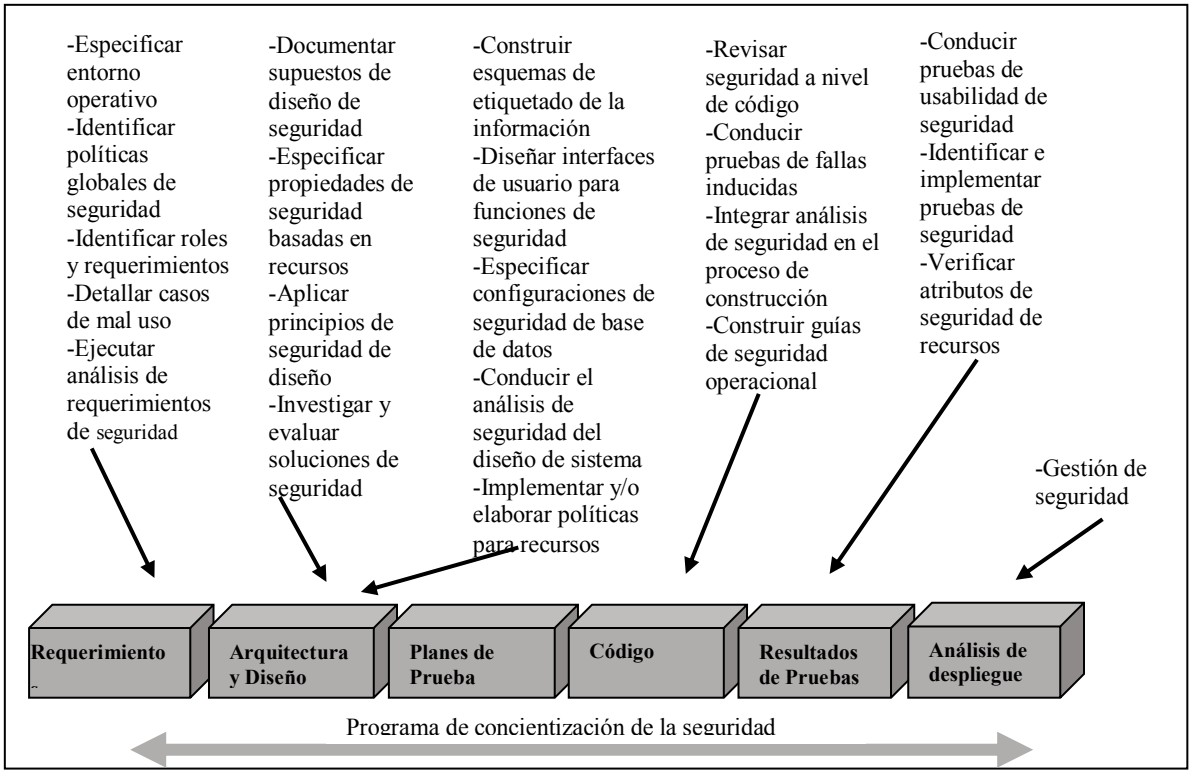


Figura 2.7Actividades y Artefactos en las distintas metodologías
Elaborado por: Investigador

1. Adicionar las prácticas relacionadas con la gestión del riesgo para garantizar la calidad del software seguro: cuáles son las prácticas que se deben adicionar al proceso QA, y cómo se deberán llevar a cabo dichas adiciones.
2. QA aplicado a las prácticas de un SDLC seguro: cuáles son las prácticas de QA que serán más útiles para garantizar la seguridad.

- **Metodología SDL**

Security Development Lifecycle (SDL)

Es un proceso para mejorar la seguridad de software propuesto por la compañía de Microsoft en el año 2004; con dieciséis actividades enfocadas a mejorar la seguridad del desarrollo de un producto de software.

Las prácticas que propone SDL van desde una etapa de entrenamiento sobre temas de seguridad, pasando por análisis estático, análisis dinámico, fuzz testing del código hasta tener plan de respuesta a incidentes. Una de las características principales de SDL es el modelado de amenazas que sirve a los desarrolladores para encontrar partes del código, donde probablemente exista vulnerabilidades o sean objeto de ataques (Brito Abundis, 2013).

Ventajas:

- Cada fase de la metodología tiene consideraciones de seguridad
- La inclusión de seguridad en cada fase, ahorra tiempo y dinero
- Mejora continua de procesos
- La metodología SDL cuenta con procesos bien definidos a través de todas las fases, fáciles a seguir de tal forma que hasta los miembros menos experimentados del equipo de trabajo pueden seguir el proyecto con facilidad.
- Esta metodología promueve la consistencia entre los diferentes proyectos, de esta forma asegura que el mantenimiento entre los diferentes productos de software puede ser administrado de una manera clara, ya que la transferencia de conocimiento entre los miembros del equipo es mínimamente requerida (Solano Campos, 2015).

Beneficios:

Los principales beneficios de SDL para los desarrolladores internos son la exposición reducida a la privacidad y confiabilidad. Los beneficios de la seguridad para las aplicaciones internas son difíciles de cuantificar. La privacidad tiene un componente de riesgo que los gerentes de alto nivel y los gerentes de riesgo entienden, y la confiabilidad tiene un componente de acuerdo de tiempo de servicio y nivel de servicio que los gerentes también entienden.

Aplicabilidad de SDL

Se considera someter a SDL las aplicaciones que posean una o varias de las siguientes características (Robles Gomez, 2011):

- Aplicaciones implementadas en un entorno empresarial
- Aplicaciones que procesan información de identificación personal (PII) u otro tipo de información confidencial
- Aplicaciones que se comunican frecuentemente a través de Internet u otras redes

2.4.2 Categorías de la Variable Dependiente

- **Desarrollo de software**

El desarrollo de software es una difícil tarea. Prueba de esto, es que existen varias propuestas metodológicas que inciden en las distintas fases del proceso de desarrollo. Por otra parte, están aquellas metodologías tradicionales que se basan especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas y las herramientas y notaciones que se usaran (Roche Saldarriaga & Suarez Arias, 2009).

- **Programación**

El primer objetivo de un programador es descomponer la tarea para especificar un plan detallado y realizable (un algoritmo) que solucione el problema. El segundo es implementar este plan en un lenguaje de programación. El tercero

es depurara el programa resultante; este proceso puede ser tan complejo como los anteriores (Casares Charles, 1999).

- **Codificación Segura - Buenas Prácticas:** Según la publicación realizada por Cesar Cuenca se detallan a continuación las siguientes buenas prácticas: (Cuenca Diaz)
    - ✓ Codificación Segura -Buenas Prácticas
    - ✓ Validación de entradas
    - ✓ Codificación de Salidas
    - ✓ Estilo de Programación
    - ✓ Manejo de Log de Cambios
    - ✓ Practicas Criptográficas
    - ✓ Manejo de errores de Logs
    - ✓ Manejo de Archivos
    - ✓ Manejo de Memoria
    - ✓ Estandarización y Reutilización
    - ✓ Funciones de Seguridad

- **Proceso de Desarrollo de software**

Un proceso de desarrollo de software es un conjunto de acciones que permiten transformar de forma eficiente la necesidad de un usuario en una solución de software efectiva (LAGOS SANTELICES, 2012).

La definición de un proceso de desarrollo de software es una descripción de este proceso en la que se identifican roles, tareas y artefactos, y que permiten guiar a los ingenieros durante el desarrollo de software

Un proceso definido permite:

- ➢ Comunicación efectiva acerca del proceso entre los usuarios, desarrolladores, gerentes, clientes e investigadores.
- ➢ Mejora de la comprensión de la gerencia, entregando una base precisa para la automatización del proceso y facilitando la movilidad del personal.

- Facilita la reutilización del proceso, disminuyendo los costos asociados a la definición de procesos.
- Soporta la evolución del proceso proveyendo de medios efectivos para el aprendizaje del proceso y un sólido fundamento para la mejora de procesos.
- Ayuda en la administración del proceso. La administración efectiva requiere planes claros y una manera precisa y cuantificada de medir el estado contra ellos. Los procesos definidos hacen posibles tales herramientas.

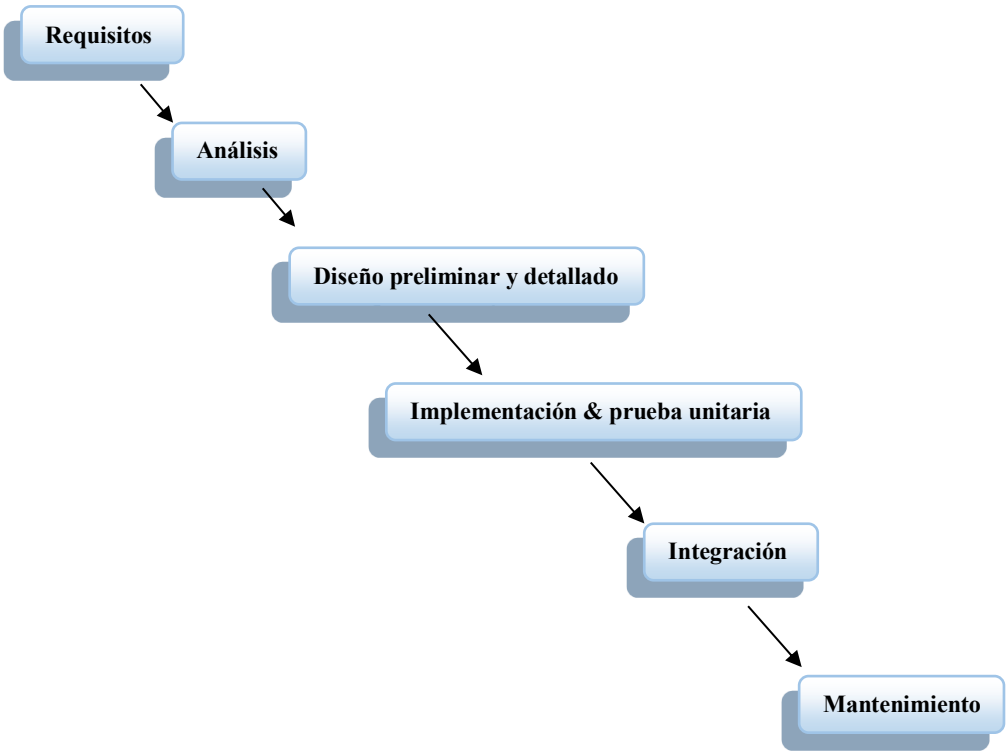


Figura 2.8: Proceso de desarrollo de software
Elaborado por: Investigador

2.5 Hipótesis

La aplicación de la metodología SDL incide en el desarrollo de software seguro en las instituciones financieras

2.6 Señalamiento de Variables

Variable Independiente: Metodología SDL

Variable Dependiente: Desarrollo de software seguro

CAPÍTULO III

3 METODOLOGÍA

3.1 Enfoque

El enfoque que tendrá el presente trabajo de investigación es cuali-cuantitativo, es cuantitativo porque se realizarán encuestas de las cuales se podrán medir los resultados; también es cualitativa porque se utilizará la observación de lo que está sucediendo en las instituciones financieras.

3.2 Modalidad básica de la investigación

Investigación Bibliográfica

La investigación será bibliográfica porque se basará en consultas en libros, artículos, publicaciones, proyectos de investigación relacionados al presente tema, con el que se fundamentará el marco teórico.

Investigación de Campo

La investigación será también de campo porque se acudirá a la institución financiera a percibir el proceso del desarrollo de software donde el equipo de desarrolladores.

3.3 Nivel o tipo de investigación

Investigación Exploratoria

La investigación será de nivel exploratorio porque al acudir al lugar donde se presenta el problema nos ayudará a despejar muchas dudas y tener claro lo que se va a proponer.

Investigación Descriptiva

La investigación será descriptiva por que se detallarán todas las fases que involucra el desarrollo de código seguro aplicando SDL.

Explicativa

La investigación es explicativa porque con la utilización de herramientas de monitoreo se podrá detectar con mayor fluidez las vulnerabilidades de los sistemas.

Investigación Correlacional

La investigación será correlacional por que busca evaluar la relación que existe entre el proceso normal de un desarrollo de software y las fases de la metodología SDL.

3.4 Población y Muestra

Se trabajará con todo el equipo de desarrollo y programadores de otras instituciones financieras lo que significa la totalidad del universo.

Población	Número	Porcentaje
Programadores de OSCUS	4	33.33%
Programadores de otras Instituciones Financieras	8	66.67%
Total	12	100%

Tabla 3.1Población de Estudio
Elaborado por: Investigador

En vista que la población a ser investigada es de apenas 12 personas se trabajará con la totalidad del universo sin que sea necesario sacar muestras representativas.

3.5 Operacionalización de Variables

3.5.1 Variable Independiente:

Cuadro No. 1: Metodología SDL

Conceptualización o Descripción	Dimensiones	Indicadores	Ítems Básicos	Técnicas e Instrumentos
La metodología SDL constituye una serie de fases consecutivas a seguir para mantener la integración de la seguridad y de la privacidad del software, mejora continua y responsabilidad, con el objetivo de reducir el número y la gravedad de vulnerabilidades del software	- integración de seguridad y privacidad del software - Mejora continua y responsabilidad - Reducir el número y gravedad de vulnerabilidades del software	- Unificación de procesos - Técnicas de programación - Herramientas actualizadas - Trabajo en equipo - Monitoreo de red - Menos brechas abiertas	- ¿Qué procesos se pueden unificar? - ¿La aplicación de técnicas de programación contribuyen a la mejora del software? - ¿Se cuenta con herramientas actualizadas para el desarrollo? - ¿Se tiene colaboración en el equipo de desarrollo? - ¿Con el monitoreo se consigue identificar las posibles brechas abiertas?	- Entrevista - Guía de la Entrevista - Entrevista - Guía de la Entrevista - Software de monitoreo de red - Herramienta

Tabla 3.2 Variable Independiente: Metodología SDL

Elaborado por: Investigador

3.5.2 Variable Dependiente:

Cuadro No. 2: desarrollo de software seguro

Conceptualización o Descripción	Dimensiones	Indicadores	Ítems Básicos	Técnicas e Instrumentos
El desarrollo de software seguro significa incluir seguridad en el software el cual es la clave principal para la sostenibilidad y ciclo de vida del mismo, por lo que debe ser diseñado, construido y probado para su seguridad; con esto se busca que continúe ejecutándose correctamente bajo ataques.	- Sostenibilidad y ciclo de vida	- Procesos - Fases	- ¿Cómo saber cuáles son los procesos o fases a seguir en el desarrollo?	- Encuesta - Cuestionario
	- Diseñado, construido y probado	- Requerimientos normativos - Pruebas exitosas durante el desarrollo	- ¿Se han definido correctamente los requerimientos normativos al inicio del desarrollo para obtener resultados exitosos en las pruebas?	- Encuesta - Cuestionario
	- Ejecución correcta	- Comportamiento del software ante ataques	- ¿Se desarrolla con criterio de fallo en mente?	- Entrevista - Guía de la Entrevista

Tabla 3.3 Variable Dependiente: desarrollo de software seguro

Elaborado por: Investigador

3.6 Recolección de Información

La técnica a emplearse será la encuesta dirigida a programadores de algunas instituciones financieras para lo que es necesario utilizar como instrumento el cuestionario a través de preguntas cerradas, lo que ayudará a la obtención más concreta de la información que queremos obtener (Ver Anexo 1).

PREGUNTAS BÁSICAS	EXPLICACIÓN
¿Para qué?	Para alcanzar los objetivos de la investigación
¿De qué personas u objetos?	Programador OSCUS Programado otras Instituciones Financieras
¿Sobre qué aspectos?	Seguridad en el desarrollo de software
¿Quién, Quiénes?	Investigador: Ing. Ruth Aleaga
¿Cuándo?	Segundo semestre del 2018
¿Dónde?	Cooperativa de Ahorro y Crédito Oscus Ltda y otras Instituciones Financieras
¿Cuántas veces?	Una
¿Qué técnicas de recolección?	Encuesta Software de monitoreo de código fuente
¿Con qué?	Cuestionario Herramientas
¿En qué situación?	En horas laborables con ética profesional y esmero.

Tabla 3.4 Recolección de la Información

Elaborado por: Investigador

3.7 Procesamiento y Análisis

- Revisión crítica de la información recogida; es decir limpieza de información defectuosa, contradictoria, incompleta, no pertinente y otras fallas.
- Repetición de la recolección, en ciertos casos individuales para corregir errores de contestación.
- Tabulación o cuadros variables de la hipótesis y objetivos:
- Manejo de información (reajuste de cuadros con casillas vacías o con datos tan reducidos cuantitativamente que no influyen significativamente en los análisis).
- Estudio estadístico de datos para presentación de resultados.

CAPÍTULO IV

4 ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

4.1 Análisis e interpretación de los resultados

Para llevar a cabo el análisis e interpretación de resultados se ha utilizado como técnica: la encuesta; la cual va dirigida a 12 programadores de diferentes instituciones financieras (Cooperativas y Mutualistas) en las que desarrollan software. De las que se obtuvo los siguientes resultados:

Pregunta 1: ¿Conoce sobre metodologías de desarrollo de software seguro?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	9	75%
NO	3	25%
TOTAL	**12**	**100%**

Tabla 4.1. Resultados Pregunta N° 1
Elaborado por: Investigador

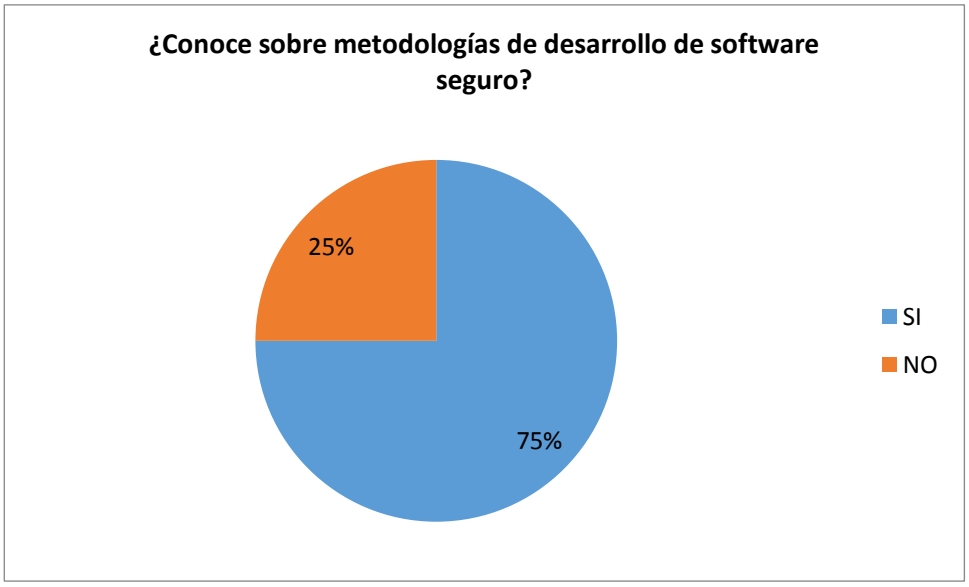


Figura 4.1 Pregunta N° 1
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 9 respondieron que SI conocen sobre metodologías de software seguro y 3 desconocen sobre metodologías de desarrollo de software seguro.

Análisis: Esto significa que en su mayoría, los programadores si conocen sobre metodologías para desarrollar software seguro.

Pregunta 2: ¿Existe alguna metodología de seguridad que aplique en el desarrollo de sistemas dentro de la Cooperativa?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	6	50%
NO	6	50%
TOTAL	**12**	**100%**

Tabla 4.2. Resultados Pregunta N° 2
Elaborado por: Investigador

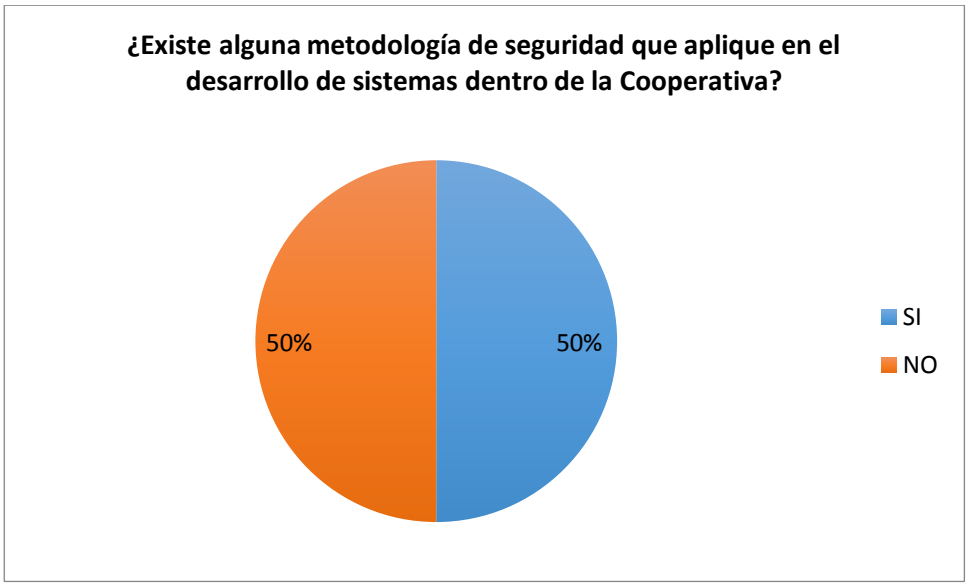


Figura 4.2. Pregunta N° 2
Elaborado por: Investigador

Interpretación: De un total de 12 personas encuestadas, 6 personas respondieron que SI aplican alguna metodología de seguridad en el desarrollo de software, y las otras 6 NO aplican ninguna metodología.

Análisis: En el sistema financiero si se están aplicando alguna metodología de seguridad en el desarrollo de software.

Pregunta 3: ¿Aplica alguna técnica de encriptación a nivel de programación?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	3	25%
NO	9	75%
TOTAL	**12**	**100%**

Tabla 4.3. Resultados Pregunta N° 3
Elaborado por: Investigador

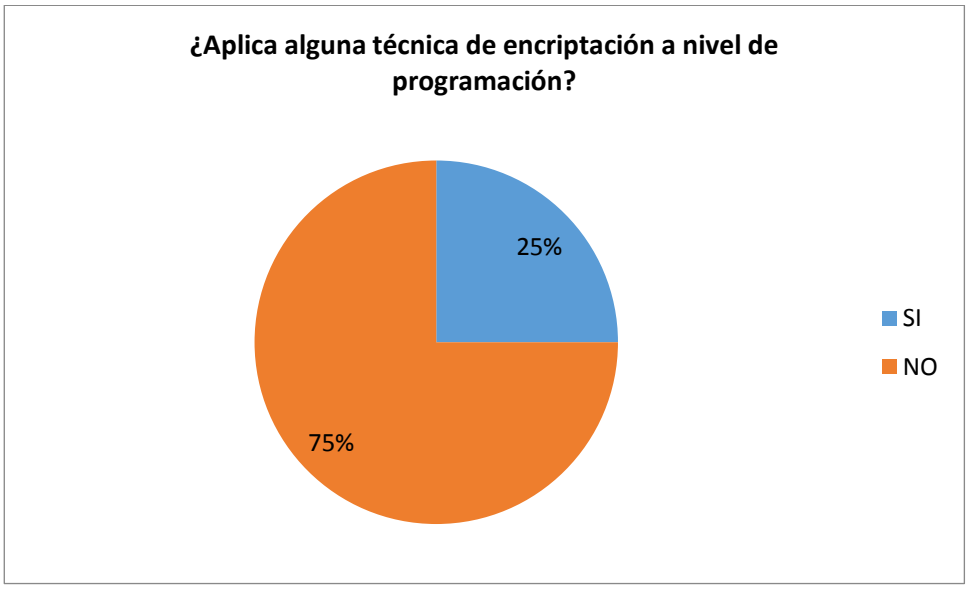


Figura 4.3 Pregunta N° 3
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 3 respondieron que si utilizan alguna técnica de encriptación, y 9 personas respondieron que no utilizan.

Análisis: De acuerdo a las respuestas se puede evidenciar que son pocas los programadores que aplican técnicas de encriptación en el desarrollo de software.

Pregunta 4: ¿Aplica herramientas de análisis de código que valide la seguridad?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	4	33%
NO	8	67%
TOTAL	**12**	**100%**

Tabla 4.4. Resultados Pregunta N° 4
Elaborado por: Investigador

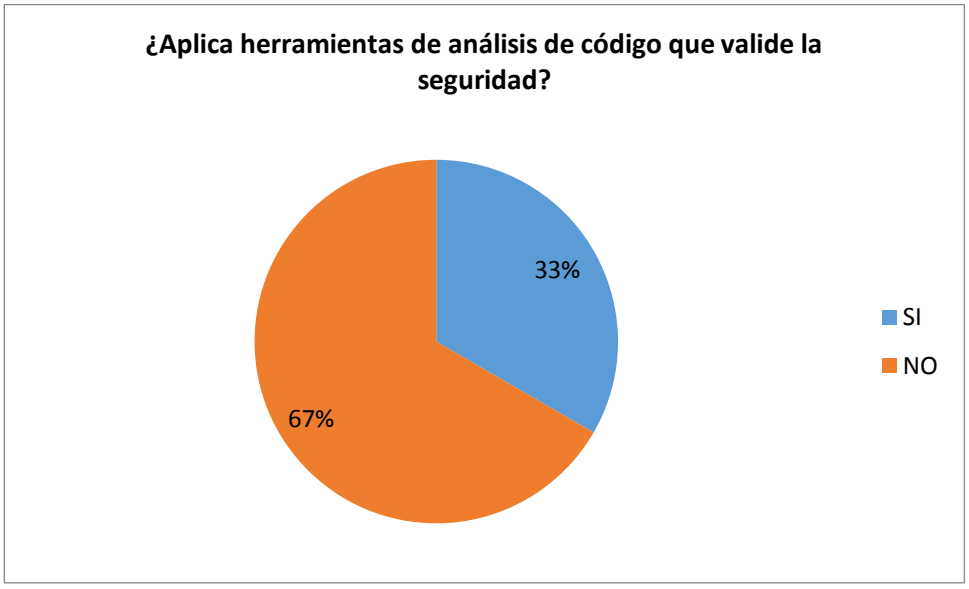


Figura 4.4 Pregunta N° 4
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 4 respondieron que aplican herramientas de análisis de código para validar la seguridad, 8 respondieron que no.

Análisis: Se puede evidenciar que la mayoría de programadores no aplican ninguna herramienta y que esto puede ocasionar que se queden brechas abiertas, lo cual genera una vulnerabilidad a nivel de programación. Por tanto, la seguridad en el código debe primar.

Pregunta 5: ¿Se realizan auditorías al código desarrollado en la Cooperativa?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	7	58%
NO	5	42%
TOTAL	**12**	**100%**

Tabla 4.5. Resultados Pregunta N° 5
Elaborado por: Investigador

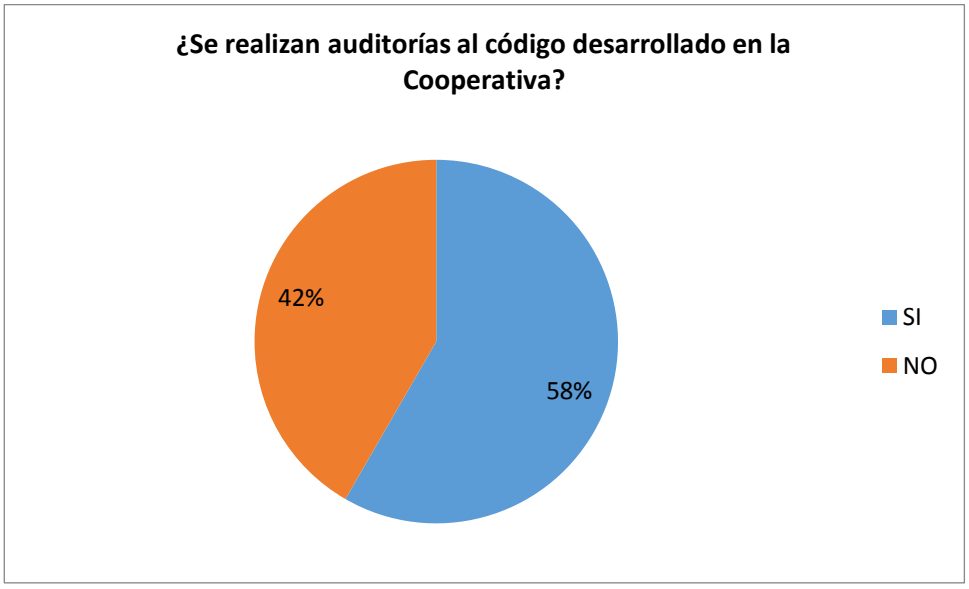


Figura 4.5 Pregunta N° 5
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 7 respondieron que si se realizan auditorías al código desarrollado en la institución a la que pertenecen, 5 respondieron que no se realizan auditorías.

Análisis: Se evidencia que al menos si existen auditorias en el código que desarrollan, por lo que puede servir de ayuda para mejorar la seguridad del código.

Pregunta 6: ¿Se han detectado amenazas hacia el código por parte de intrusos?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	0	0%
NO	12	100%
TOTAL	**12**	**100%**

Tabla 4.6. Resultados Pregunta N° 6
Elaborado por: Investigador

Figura 4.6 Pregunta N° 6
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, todas respondieron que no se han detectado amenazas hacia el código desarrollado, equivalente al 100%.

Análisis: Hasta el momento no se han detectado amenazas en el código, pero se puede estar expuesto a que esto ocurra y mientras la tecnología va avanzando se deben ir también implementando seguridades.

Pregunta 7: ¿Considera que el software desarrollado es vulnerable a algún tipo de amenaza?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	8	67%
NO	4	33%
TOTAL	**12**	**100%**

Tabla 4.7. Resultados Pregunta N° 7
Elaborado por: Investigador

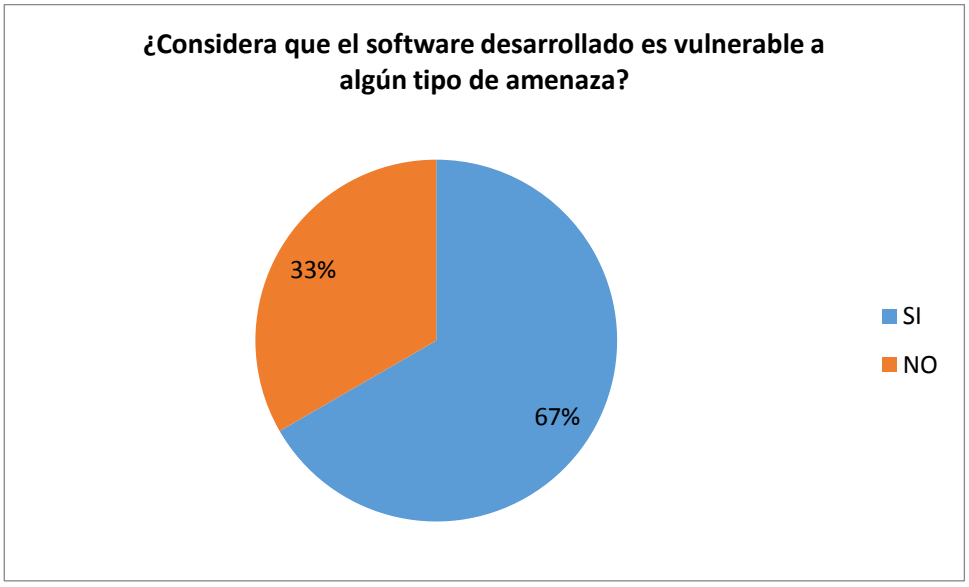


Figura 4.7 Pregunta N° 7
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 8 respondieron que consideran que el software desarrollado es vulnerable a amenazas mientras que 4 consideran que no es vulnerable.

Análisis: La mayor parte de los programadores encuestados están conscientes de que el software que desarrollan son vulnerables a amenazas.

Pregunta 8: ¿Se preocupa de la funcionalidad del sistema antes que de la seguridad?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	10	83%
NO	2	17%
TOTAL	**12**	**100%**

Tabla 4.8. Resultados Pregunta N° 8
Elaborado por: Investigador

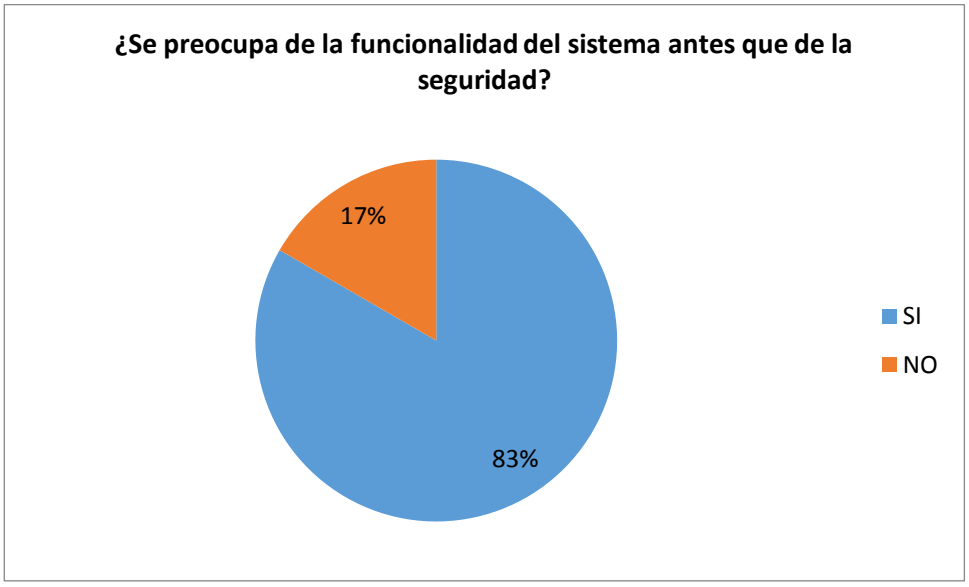


Figura 4.8 Pregunta N° 8
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 10 respondieron que su interés está centrado en la funcionalidad del sistema y 2 respondieron que se preocupan de la seguridad.

Análisis: El mayor porcentaje de programadores encuestados se preocupan más de la funcionalidad del sistema antes que de la seguridad, lo cual implica riesgos porque a más de que el sistema sea funcional, debe proveer seguridad.

Pregunta 9: ¿Considera indispensable la aplicación de alguna metodología de seguridad en el desarrollo de software?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	12	100%
NO	0	0%
TOTAL	**12**	**100%**

Tabla 4.9. Resultados Pregunta N° 9
Elaborado por: Investigador

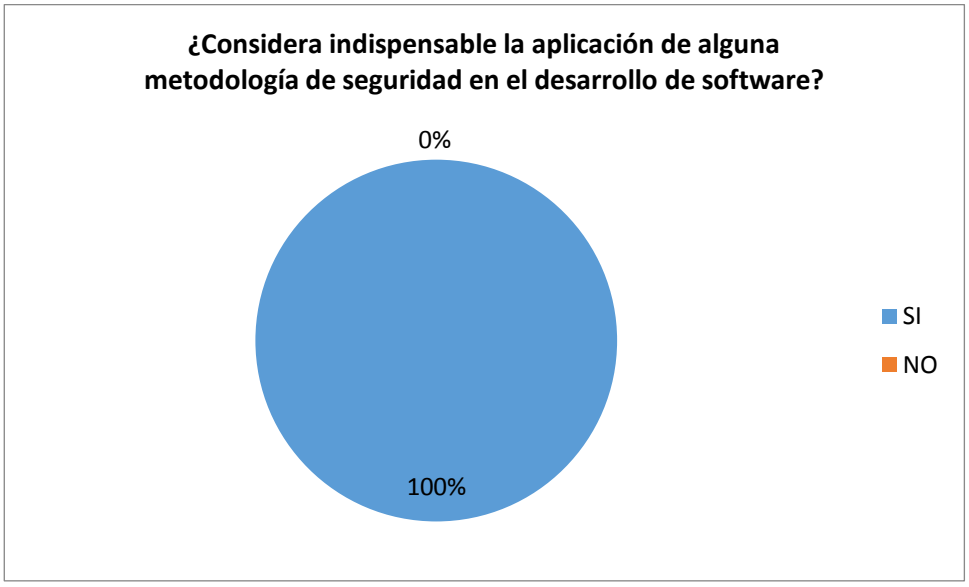


Figura 4.9 Pregunta N° 9
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, todos respondieron que es indispensable la aplicación de alguna metodología de seguridad en el desarrollo de software, lo que equivale al 100%.

Análisis: Todos los programadores encuestados están de acuerdo que se debe aplicar alguna metodología para desarrollar software seguro.

Pregunta 10: ¿Se ve afectado el negocio por la carencia de seguridad en los sistemas financieros que utilizan en cada institución?

OPCIÓN	VALOR	PORCENTAJE (%)
SI	6	50%
NO	6	50%
TOTAL	**12**	**100%**

Tabla 4.10. Resultados Pregunta N° 10
Elaborado por: Investigador

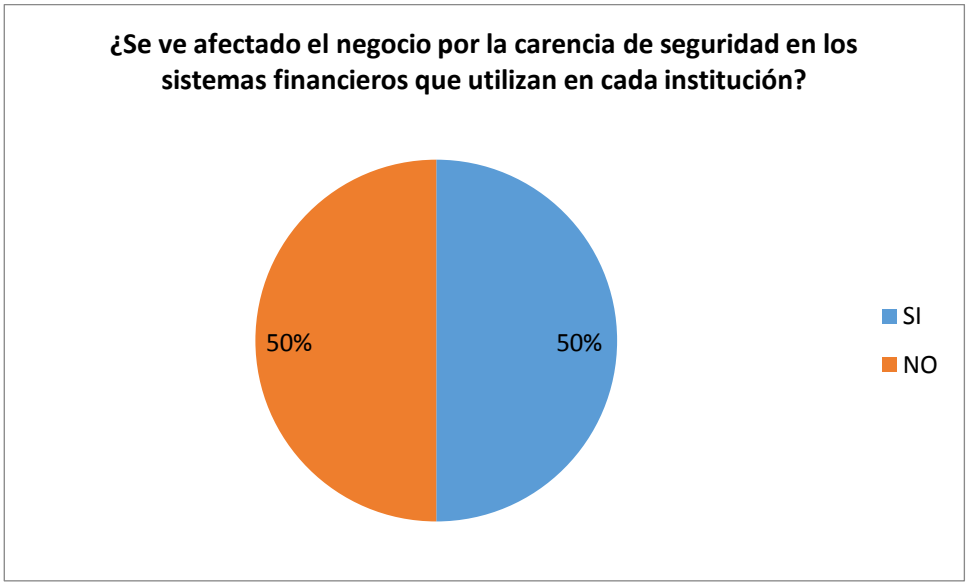


Figura 4.10 Pregunta N° 10
Elaborado por: Investigador

Interpretación: De las 12 personas encuestadas, 6 respondieron que si se ve afectado el negocio por la carencia de seguridad en el código desarrollado, 6 respondieron que no.

Análisis: La mitad de los programadores encuestados han identificado que el negocio de cada institución financiera a la que pertenecen se ve afectada por la carencia de seguridad en el software que desarrollan.

4.2 Verificación de la Hipótesis

Para verificar la hipótesis y determinar si es aplicable o no la metodología SDL para el desarrollo de código seguro, se ha tomado como referencia el método estadístico t - Student, debido a que la población es reducida.

4.2.1 Planteamiento de la Hipótesis

Hipótesis Nula (H_0): La aplicación de la metodología SDL NO incide en el desarrollo de software seguro en las instituciones financieras.

Hipótesis Alterna (H_1): La aplicación de la metodología SDL SI incide en el desarrollo de software seguro en las instituciones financieras.

- **Nivel de significancia**

  Luego de determinar la hipótesis nula e hipótesis alternativa, se determina el nivel de significancia, en este caso se trabajará con un nivel de significancia del 0.05.

- **Prueba Estadística**

  La fórmula para la prueba estadística aplicada es:

$$t = \frac{\hat{X}_D - \mu_0}{\frac{S_D}{\sqrt{n}}}$$

- **Simbología**

  t = estadístico t de Student

  $\hat{X}_D$ = media aritmética diferencial muestral

  S_D = desviación estándar diferencia muestral

  n = tamaño de muestra

  μ_0 = constante

- **Aplicación de la fórmula**

PREGUNTAS	SI	NO	X_D	$X-\hat{X}$	$(X-\hat{X})^2$
1	9	3	6	5	25
2	6	6	0	0	0
3	3	9	-6	-6	36
4	4	8	-4	-4	16
5	7	5	2	2	4
6	0	12	-12	-12	144
7	8	4	4	4	16
8	10	2	8	8	64
9	12	0	12	12	144
10	6	6	0	0	0

		TOTAL	**10**		**449**

Tabla 4.11. Tabla de fórmulas
Elaborado por: Investigador

- **Media Aritmética**

$$\hat{X}_D = \frac{\Sigma\, \chi_1}{n_1}$$

$$\hat{X}_D = \frac{10}{10}$$

$$\hat{X}_D = 1$$

- **Cálculo de la desviación estándar**

$$\boldsymbol{S_D} = \sqrt{\frac{\sum_i^n (\boldsymbol{\chi_1} - \widehat{\boldsymbol{\chi_1}})^2}{\boldsymbol{n_1 - 1}}}$$

$$\boldsymbol{S_D} = \sqrt{\frac{449}{10-1}}$$

$$\boldsymbol{S_D} = \sqrt{\frac{449}{9}}$$

$$\boldsymbol{S_D} = 7{,}063$$

- **Cálculo t de Student**

$$\boldsymbol{t = \frac{\widehat{\chi}_D - \mu_0}{\frac{S_D}{\sqrt{n}}}}$$

$$\boldsymbol{t = \frac{1 - 0}{\frac{7{,}063}{\sqrt{10}}}}$$

$$\boldsymbol{t = \frac{1}{2{,}23}}$$

$$\boldsymbol{t = 0{,}447}$$

- **Cálculo de los grados de libertad**

Grado de Libertad (GL) = n - 1

(GL) = 10 - 1

(GL) = 9

Con 9 grados de libertad y un nivel de significancia 0,05 según tabla de cuantiles de la distribución t de Student: ***t* = 2,262**

	AREA DE DOS COLAS						
g1	**0,20**	**0,10**	**0,05**	**0,02**	**0,01**	**0,001**	**0,0001**
1	3,078	6,314	12,706	31,821	63,657	636,619	6366,198
2	1,886	2,920	4,303	6,695	9,925	31,598	99,992
3	1,638	2,353	3,182	4,541	5,841	12,924	28,000
4	1,533	2,132	2,776	3,747	4,604	8,610	15,544
5	1,476	2,015	2,571	3,365	4,032	6,869	11,178
6	1,440	1,943	2,447	3,143	3,707	5,959	9,082
7	1,415	1,895	2,365	2,998	3,499	5,408	7,885
8	1,397	1,860	2,306	2,896	3,355	5,041	7,120
9	1,383	1,833	2,262	2,821	3,250	4,781	6,594
10	1,372	1,812	2,228	2,764	3,169	4,587	6,211

Tabla 4.12. Tabla de cuantiles de la distribución t de Student
Fuente: Dpt. Estadística Inv. Operativa Universidad de Valencia

- **Decisión Final**

  El valor calculado t = 0,447; con 9 grados de libertad y 95% de confiabilidad al buscar en la tabla t de Student el valor crítico equivale a 2,262 por lo tanto el valor t calculado se encuentra dentro del intervalo de confianza y de acuerdo a la regla de decisión se rechaza la hipótesis nula y se acepta la hipótesis alternativa confirmando que la aplicación de la metodología SDL SI incide en el desarrollo de software seguro en las instituciones financieras.

CAPÍTULO V

5 CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- De acuerdo a la encuesta aplicada a algunas Instituciones Financieras se determina que los sistemas utilizados en ellas no están siendo desarrollados con la seguridad necesaria, en algunas no se aplican métodos de encriptación para garantizar la seguridad en la información. Esta es una vulnerabilidad de alto riesgo para las instituciones.
- La falta de aplicación de metodologías de seguridad en el desarrollo de software involucra riesgos en las Instituciones Financieras, esto puede ser causal de pérdidas económicas e inestabilidad de los sistemas que se desarrollan dentro de las instituciones.
- La encuesta aplicada a varios programadores de diferentes instituciones financieras ayudó a determinar que es necesario implementar metodologías de seguridad para el desarrollo de software, la cual contribuya en la mejora de los sistemas haciéndolos más eficaces y eficientes. De tal manera se pueda contar con bases sólidas en todos los sistemas que se desarrollan dentro de cada institución.

5.2 Recomendaciones

- Se recomienda la aplicación de seguridades en el desarrollo de software. Los programadores deberían hacer uso de técnicas de encriptación de datos en los programas que se desarrollan e implementan. Esto puede contribuir al aseguramiento de la información que se maneja dentro del sistema financiero o algún otro aplicativo que se desarrolle en la institución.
- Se recomienda la ejecución de auditorías de seguridad al código desarrollado ya que con esto se puede detectar vulnerabilidades y se puede aplicar medidas de seguridad preventivas y/o correctivas, podría ser de forma periódica según se vaya avanzando con el desarrollo.

- Se recomienda hacer uso de herramientas que ayuden a detectar falencias en el código. Los programadores deberían considerar la seguridad del código como algo primordial a parte de la funcionalidad del sistema, de no ser así se puede llegar a ser víctimas de ataques malintencionados por parte de agentes externos maliciosos.
- La implementación de alguna metodología de seguridad en el desarrollo de software en las Instituciones Financieras mitigará riesgos de seguridad y generará sistemas de calidad, pues al momento la carencia de metodologías provoca que el código sea vulnerable a amenazas, ya que se está desarrollando sin aplicar ninguna medida de seguridad; solo enfocándose en la parte funcional del sistema y dejando de lado la seguridad que es muy importante, más aún cuando la actividad económica a la que se dedica la institución se relaciona directamente al manejo de dinero

CAPÍTULO VI

6 PROPUESTA

6.1 Tema

Análisis de la metodología SDL para su aplicación en el desarrollo de software seguro en Instituciones Financieras.

6.2 Datos informativos

6.2.1 Institución ejecutora

Cooperativa de Ahorro y Crédito Oscus Ltda.

6.2.2 Beneficiarios

Área de desarrollo - Departamento de sistemas

6.2.3 Ubicación

Provincia: Tungurahua
Cantón: Ambato
Dirección: Lalama 06-39 entre Sucre y Bolívar

6.2.4 Equipo técnico responsable

- Investigadora: Ing. Ruth Elizabeth Aleaga Guaigua
- Responsable del Departamento de Sistemas

6.3 Antecedentes de la propuesta

La Cooperativa de Ahorro y Crédito Oscus Ltda. es una institución financiera que cuenta con 11 agencias a nivel nacional. Provee un único sistema financiero para todas las agencias y una sola base de datos que se encuentra consolidado en la oficina Matriz, por lo tanto en el data center de la Matriz se encuentran ubicados todo lo que son servidores principales y desde ahí se controla la información proveniente de todas las oficinas operativas.

Adicional, la Cooperativa cuenta con un sitio alterno o contingente ubicado en la ciudad de Guayaquil, allí existen servidores con características similares a los de la oficina Matriz, en los cuales se encuentran los backups del sistema y de la base de datos. La Cooperativa brinda algunos servicios para los cuales cuenta con otros sistemas independientes, entre ellos se destacan: Chequera Virtual, Sistema de Gestión de Calidad (SGC), Sistema para la Geo-Referenciación, etc. Todos ellos son desarrollados por el personal (programadores) de la Cooperativa.

Actualmente los programadores realizan su trabajo de forma independiente, sin seguir alguna metodología de seguridad que les ayude a desarrollar código seguro en sus aplicaciones. Esto se ha podido identificar mediante la aplicación de encuestas en el área de desarrollo, con las cuales se obtuvo evidencias de que si existen falencias y carencia de seguridad en el desarrollo. Si bien es cierto se constató que si aplican técnicas de encriptación, lo cual si contribuye a la seguridad de la información más no al código que desarrollan.

6.4 Justificación

Con el resultado del análisis e interpretación de cada una de las preguntas de la encuesta se ha logrado evidenciar la falta de seguridad que existe en el desarrollo de software sobre los sistemas implementados en la Cooperativa.

El presente proyecto se enfocará en la investigación sobre la aplicación de seguridad en el código que se desarrolla en la Cooperativa y gracias a las

factibilidades tecnológicas, operativas y legales que posee la institución se podrá hacer posible que se desarrolle el proyecto.

La propuesta de implementación de la metodología de desarrollo de software seguro SDL contribuirá a que el software desarrollado sea de calidad y garantice la seguridad necesaria tanto para el código como para la información que se opera en los sistemas. Por tanto la Cooperativa podrá proveer un servicio más eficiente a sus socios y clientes.

6.5 Objetivos

6.5.1 General

Analizar la metodología SDL para su aplicación en el desarrollo de software para que se integre seguridad en el código y se garantice calidad en los sistemas de la Cooperativa.

6.5.2 Específicos

- Detectar vulnerabilidades en el aplicativo Chequera Virtual que se encuentra expuesto en el internet
- Determinar las fases de la metodología SDL para tener un marco de referencia y aplicarla en los sistemas de la Cooperativa
- Analizar cada una de las fases que componen la metodología SDL para su aplicación en los sistemas que se desarrollan en la Cooperativa

6.6 Análisis de factibilidad

6.6.1 Factibilidad tecnológica

El proyecto es tecnológicamente factible, ya que la Cooperativa dispone de la tecnología necesaria y de equipos para la ejecución de pruebas de detección de vulnerabilidades y de aplicación de la metodología. Así como también cuenta con

ambientes de desarrollo en donde se puede ejecutar e instalar el software necesario para dar cumplimiento con los objetivos y obtener resultados satisfactorios.

6.6.2 Factibilidad organizacional

Es factible organizacionalmente debido a que la alta gerencia y el responsable de sistemas brindan el apoyo necesario para la ejecución del proyecto, teniendo el pleno conocimiento de la carencia de seguridad que existe en los sistemas que se encuentran operativamente funcionando en la Cooperativa y consideran necesario aplicar las debidas seguridades en el desarrollo de software.

6.6.3 Factibilidad económica-financiera

Económicamente es factible, puesto que no se requiere de adquisición de equipos para la implementación de la metodología. La Cooperativa cuenta con los recursos necesarios para poder ejecutar los procesos que se lleven a cabo en cada fase de la metodología, en cuanto al software que se utilizara para el análisis de vulnerabilidades son herramientas gratuitas y software libre.

6.6.4 Factibilidad legal

El proyecto de investigación titulado "ANÁLISIS DE LA METODOLOGÍA SDL PARA SU APLICACIÓN EN EL DESARROLLO DE SOFTWARE SEGURO EN INSTITUCIONES FINANCIERAS", está legalmente aprobado para su aplicación por parte de la Gerencia de la Cooperativa y cuenta con el respaldo del Responsable de Sistemas para poder hacer uso de los recursos de hardware y software que sean necesarios para su desarrollo.

6.6.5 Factibilidad operativa

El presente proyecto de investigación aportará operativamente al mejoramiento de desarrollo de software seguro, con la aplicación de las fases propuestas en la metodología SDL.

El equipo de desarrollo de la Cooperativa se encuentra integrado por cuatro programadores. Cada uno de ellos se encarga de realizar reuniones de trabajo con los dueños de los procesos que son quienes solicitan los requerimientos de

desarrollo. Los programadores junto con el responsable de TIC se dedican a analizar la factibilidad del requerimiento solicitado.

6.7 Fundamentación

Que es un software seguro?

- Diseñado, construido y probado para ser seguro
- Continúa ejecutándose correctamente bajo un ataque
- Diseñado con el fallo en mente

Propiedades fundamentales de un software seguro (Castellaro, Romaniz, Ramos, & Pessolani, 2009)

Las siguientes son propiedades fundamentales o atributos de seguridad del SS:

- **Confidencialidad.** El software debe asegurar que cualquiera de sus características (incluidas sus relaciones con su ambiente de ejecución y sus usuarios), los activos que administra y/o su contenido son accesibles sólo para las entidades autorizadas e inaccesibles para el resto.
- **Integridad.** El software y los activos que administra son resistentes y flexibles a la subversión (modificaciones no autorizadas del código, los activos administrados, la configuración o el comportamiento del software por parte de entidades autorizadas). Esta propiedad se debe preservar durante el desarrollo del software y su ejecución.
- **Disponibilidad.** El software debe estar operativo y accesible a sus usuarios autorizados (humanos o procesos) siempre que se lo requiera; y desempeñarse con una performance adecuada para que los usuarios puedan realizar sus tareas en forma correcta y dar cumplimiento a los objetivos de la organización que lo utiliza. Para las entidades que actúan como usuarios (generalmente asociadas con los usuarios finales) se requieren dos propiedades adicionales:
- **Accountability.** Todas las acciones relevantes relacionadas con la seguridad de una entidad que actúa como usuario se deben registrar y trazar a fin de poder establecer responsabilidades; la trazabilidad debe ser posible tanto durante la ocurrencia de las acciones registradas como a posteriori.

- **No Repudio.** La habilidad de prevenir que una entidad que actúa como usuario desmienta o niegue la responsabilidad de acciones que han sido ejecutadas. Asegura que no se pueda subvertir o eludir la propiedad Accountability.

En consecuencia, los efectos de vulnerar la seguridad del software se pueden describir en términos de los efectos sobre estas propiedades fundamentales.

6.7.1 Análisis de Metodologías para el desarrollo de software seguro

En el artículo ***Metodologías para desarrollar software seguro*** (Brito Abundis, 2013), menciona que la clave para desarrollar un software seguro, está en el proceso de desarrollo utilizado. Debido a que es en el proceso, en donde se produce el producto, el cual pueda resistir o sostenerse a ataques ya anticipados, recuperarse rápidamente y mitigar el daño causado por los ataques que no pueden ser eliminados o resistidos. Manifiesta que si los desarrolladores contaran con un mejor equipo serían capaces de reconocer lo que implica su diseño y su posibilidad de implementación, con esto se podría evitar los defectos relacionados con la seguridad en software.

Entre las metodologías que existen para desarrollar software seguro, se ve que cada una de ellas posee una serie de pasos enfocados en buscar seguridad y resistencia a ataques. Entre ellas se analizará: Software Assurance Maturity Model (OpenSAMM) de OWASP, Microsoft Security Development Lifecycle (Microsoft SDL), Building Security In Maturity Model (BSIMM).

- **Metodología OpenSAMM**

  En el sitio web oficial (OPENSAMM Software Assurance Maturity Model, 2009) , manifiesta sobre la metodología OpenSAMM (Software Assurance Maturity Model) que en español significa modelo de madurez para el aseguramiento de software, es un marco de trabajo abierto para ayudar a las organizaciones a formular e implementar una estrategia de seguridad para Software que sea adecuada a las necesidades específicas que está enfrentado la organización.

Los recursos proveídos por el SAMM ayudarán a:

- Evaluar las prácticas de seguridad en Software existentes en la organización
- Construir un programa de seguridad en Software balanceado en iteraciones bien definidas
- Demostrar mejoras concretas en el programa de aseguramiento de Software
- Definir y medir las actividades relacionadas con seguridad en la organización

SAMM fue definido para ser flexible de manera que pueda ser utilizado por organizaciones pequeñas, medianas o grandes que utilicen cualquier estilo de desarrollo. Además, este modelo puede ser aplicado en toda la organización, en una sola línea de negocio o incluso en un proyecto en particular.

SAMM fue construido sobre los siguientes principios:

- Cambios de comportamiento de una organización a través del tiempo. Un programa de seguridad para Software exitoso debería ser creado en pequeños ciclos que entreguen ganancias tangibles en el aseguramiento de Software, al mismo tiempo, debe trabajar incrementalmente hacia metas de largo plazo.
- No hay una sola receta que funcione para todas las organizaciones. Un marco de seguridad en Software debe ser flexible y permitir a las organizaciones personalizar sus opciones basándose en su tolerancia a riesgo y la manera en la cual construye y usa el Software.
- Los lineamientos relacionados a las actividades de seguridad deben ser específicos. Todos los pasos en la construcción y medición del programa de aseguramiento deben ser simples, bien definidos y medibles. Este modelo también provee plantillas de planes de implementación para tipos comunes de organizaciones.

  Las bases de este modelo están construidas alrededor de las funciones de negocio relacionadas al desarrollo de Software, se incluyen una serie de prácticas relacionadas a cada función como se muestra en el siguiente diagrama:

Figura 6.1 SAMM Descripción
Fuente: OPENSAMM Software Assurance Maturity Model

Como se puede observar esta metodología está orientada a establecer estrategias de seguridad en el desarrollo y es adaptable a las necesidades de la empresa. Además que puede ser aplicada a cualquier proyecto por más pequeño que este sea. Tiene la característica de que puede ser flexible y tolerante a cambios, ya sea a corto o largo plazo.

- **Metodología SDL**

En el artículo ***Building More Secure Commercial Software: The Trustworthy Computing Security Development Lifecycle*** (Lipner, 2005), menciona que la seguridad es un requisito fundamental para los proveedores de software, impulsado por las fuerzas del mercado, la necesidad de proteger las infraestructuras críticas, y la necesidad de construir y preservar la confianza generalizada en la informática.

El principal desafío para todos los proveedores de software es crear un software más seguro que requiera menos actualización a través de parches y la administración de seguridad sea menos costosa. Para la industria del software, la clave para satisfacer la demanda actual de seguridad mejorada es implementar procesos repetibles que sean confiables hasta cierto grado.

Por lo tanto, los proveedores de software deben pasar a un proceso de desarrollo de software más estricto que se enfoque en mayor medida en la seguridad. Tal proceso tiene la intención de minimizar el número de vulnerabilidades de seguridad existentes en el diseño, codificación y documentación; y detectar y eliminar esas vulnerabilidades tan pronto en el ciclo de vida del desarrollo como sea posible.

La necesidad de un proceso de este tipo es mayor para el software empresarial y de consumo que probablemente se utilizará para procesar los datos recibidos desde el Internet, para controlar sistemas críticos que puedan ser atacados, o para procesar información de identificación personal.

Existen tres facetas para construir software más seguro: proceso repetible, educación de ingeniería, y métricas y responsabilidad. Este documento se enfoca en el aspecto del proceso repetible del SDL, aunque sí discute la educación de ingeniería y proporciona algunas métricas generales que muestran el impacto hasta la fecha de la aplicación de un subconjunto del SDL.

El SDL implica modificar los procesos de una organización de desarrollo de software mediante medidas integradas que conducen a una mejor seguridad del software. Este documento resume esas medidas y describe la forma en que se integran en un ciclo de vida de desarrollo de software típico. La intención de estas modificaciones no es modificar totalmente el proceso, sino agregar puntos de control de seguridad y entregables de seguridad bien definidos.

Este documento describe, a un nivel muy alto, los conceptos del SDL, que se muestran en la Figura 6.2.

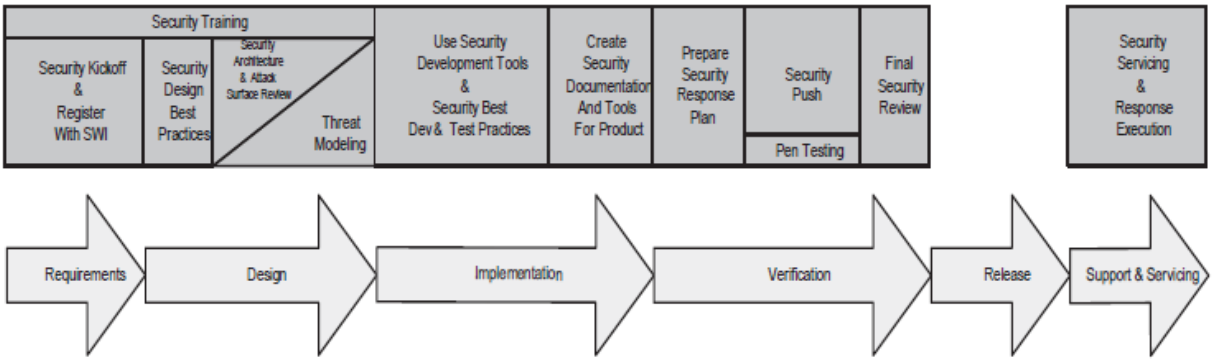


Figura 6.2 Mejoras de SDL para el proceso de Desarrollo de Microsoft
Fuente: Building More Secure Commercial Software: The Trustworthy Computing Security Development Lifecycle

- **Metodología BSIMM (Building Security In Maturity Model)**

Según el estudio realizado (Robles Gomez, 2011) sobre la Metodología BSIMM indica que BSIMM pretende ser una guía para la construcción y la evolución de una iniciativa de seguridad de software. Requiere de una planificación cuidadosa e implica un cambio de organización amplio, de esta manera se inculca la seguridad del software en la organización. Pero señalando claramente los objetivos y metas, y por las prácticas de seguimiento con indicadores adaptados a su propia iniciativa, que metódicamente se puede construir la seguridad del software en las prácticas de su organización de desarrollo de software.

Al utilizar el BSIMM como una guía para su propia iniciativa de seguridad de software, puede aprovechar la experiencia de muchos años capturada en el modelo.

Software de seguridad siempre debe ser impulsado por la gestión de riesgo empresarial. Para evolucionar los programas y alcanzar un software de seguridad se debe llevar a cabo las actividades descritas en el BSIMM, sin un coste excesivo.

Las iniciativas de seguridad de software se han creado con algunos de los objetivos de alto nivel en la mente. El BSIMM es apropiado si las metas de negocio global de software de seguridad incluyen:

- ✓ Decisiones informadas sobre la gestión de riesgos.
- ✓ Claridad sobre lo que es "lo que hay que hacer" para todos los involucrados en el software de seguridad.
- ✓ Reducción de costes a través de procesos estándar y repetible.
- ✓ Aumento de la calidad del código.

Comparativo de Metodologías de desarrollo de software seguro

Las 3 metodologías citadas y documentadas nos permiten tener una idea clara de las estrategias que cada una aplica en el desarrollo de software seguro, con un cuadro comparativo se puede evidenciar los ítems importantes y conocer más acerca de la metodología propuesta:

Característica	OpenSAMM	SDL	BSIMM
Adapta a necesidades de la organización	SI	SI	SI
Se basa en iteraciones	SI	SI	NO
Recursos aplicados	Desarrolladores, arquitectos, administradores, QA tester, Auditores de seguridad, dueños del negocio, personal de apoyo, operaciones	Tester y líderes de equipo	Constructores (desarrolladores, arquitectos, administradores) Tester, Operaciones, Administradores, Ejecutivos y mandos medios (dueños del negocio, administradores del producto)
Numero Fases/actividades	20 practicas	5 fases	109 actividades
Acceso libre	SI	SI	SI
Aplicable a metodologías ágiles	SI	SI	NO
Consume menos tiempo	NO	SI	NO
Requiere menos recursos	NO	SI	NO
Modelado de amenazas	NO	SI	NO

Tabla 6.1. Comparativo de Metodologías de desarrollo de software seguro
Elaborado por: Investigador

Luego de realizar el cuadro comparativo entre las metodologías investigadas y conforme a las características que cada una posee, se considera óptima la metodología SDL por ser la que menos recursos necesita, la cual indica que sólo es necesario un Tester y los líderes de equipo para la implementación de este ciclo de vida.

Otro aspecto a considerar y que es muy importante es el tiempo, esto quiere decir el esfuerzo que se requiere para la implementación y adaptación del ciclo de vida. Para determinar esto se consideró el número de actividades, prácticas, fases o procedimientos que tiene cada una de las metodologías y la flexibilidad de omitir alguna actividad o adaptarla según las necesidades de la organización, Como se

puede observar en el cuadro comparativo SDL es la menos tiempo requiere y la más flexible, ya que consta de 5 fases.

Por tanto se descarta a las demás metodologías ya que son menos flexibles en cuanto a la ejecución de sus actividades, ellas requieren muchas más revisiones y verificaciones de código muy rigurosas, por tanto demandan de más esfuerzo al implementar.

6.7.2 Metodología SDL para el desarrollo de software seguro

En el artículo ***Secure Software Development Life Cycle Processes: A Technology Scouting Report*** (Noopur, 2005)**,** menciona que el Ciclo de vida de desarrollo seguro (SDL) de confianza de Microsoft es un proceso que han adoptado para el desarrollo de software necesarios para resistir los ataques de seguridad.

El proceso agrega una serie de actividades centradas en la seguridad y entregables a cada fase del proceso de desarrollo de software de Microsoft. Estas actividades de seguridad y entregas incluyen la definición de los requisitos de características de seguridad y actividades de aseguramiento durante la fase de requisitos, el modelado de amenazas para identificación del riesgo de seguridad durante la fase de diseño del software, el uso de herramientas de análisis estático de escaneo de código y revisiones de códigos durante la implementación, y pruebas centradas en la seguridad, incluida la "prueba de fuzz" durante la fase de prueba. Una actividad de seguridad adicional incluye una revisión final del código nuevo y heredado durante la fase de verificación. Finalmente, durante la fase de lanzamiento, una revisión de seguridad final es llevada a cabo por el Equipo de Seguridad de Microsoft Central, un equipo de expertos de seguridad que también están disponibles para el equipo de desarrollo de productos durante el ciclo de vida de desarrollo y tienen un rol definido en el proceso general.

Microsoft ha aumentado el SDL con entrenamiento de seguridad obligatorio para su personal de desarrollo de software, métricas de seguridad y con experiencia en seguridad disponible a través del equipo de Seguridad de Microsoft Central.

Microsoft informa resultados alentadores de productos desarrollados utilizando el SDL, medido por el número de boletines de seguridad críticos e importantes emitidos por Microsoft para un producto después de su lanzamiento.

En el libro titulado ***Delivery and adoption of Cloud Computing Services in Contemporary Organizations*** (Chang, Walters, & Wills, 2015), indica que SDL tiene un valor agregado para el modelado de seguridad ya que viene con una herramienta de modelado de amenazas que es integral para analizar los requisitos, genera amenazas y riesgos. Algunos de los beneficios de SDL son:

- Soporte completo de Lifecycle para desarrollo basado en seguridad.
- Se han utilizado en el desarrollo de Windows Vista y SQL Server
- Reducir el número de vulnerabilidades de software
- Reducir el costo total del desarrollo (es posible ahorrar 30 veces si las vulnerabilidades de seguridad se solucionan anticipadamente)

En el libro titulado ***THE SECURITY DEVELOPMENT LIFECYCLE: SDL, a process for developing demonstrably more secure software*** (Howard & Lipner, 2006), menciona que los principales beneficios de SDL para los desarrolladores internos son la reducción de la privacidad y la exposición a la confiabilidad.

Los beneficios de la seguridad para las aplicaciones internas son difíciles de cuantificar. La privacidad tiene un componente de riesgo que entienden los gerentes superiores y los gerentes de riesgos, y la confiabilidad tiene un componente de acuerdo al tiempo de actividad y nivel de servicio que los gerentes también entienden.

El mandato SDL se aplica a cualquier software que cumple estos criterios:

- El software se usa regularmente en una empresa, negocio, agencia gubernamental u otra organización.
- El software se usa regularmente para procesar información personal o confidencial.
- El software se usa regularmente para conectarse a Internet. (Este requisito no cumple el software que interactúa con Internet solo para actualizar su código

o bases de datos conectándose a un servidor de Internet operado por Microsoft).

Factores que afectan el costo de SDL

Los siguientes factores ayudan a implementar SDL de manera efectiva:

- La implementación del SDL es más barata si se aplica a un proyecto que está creando un producto o aplicación desde cero. Su capacidad de elegir idiomas, estándares de codificación y herramientas con la seguridad en mente, y para evitar errores en lugar de corregirlos, genera eficiencia en tiempo y esfuerzo.
- De manera similar, aplicar el SDL a una cantidad de "código heredado" (escrito antes de considerar la seguridad) es costoso. Los equipos tienen que encontrar problemas, realizar cambios y asegurarse de que sus cambios no causen ningún problema, como la compatibilidad con versiones anteriores, con el correcto funcionamiento del producto o componente.
- A partir de los dos elementos anteriores, se deduce que la segunda y posteriores versiones del código que ha pasado por el SDL deberían ser menos costosas que la primera. El primer lanzamiento de SDL paga el precio, en tiempo y calendario, para limpiar la mayoría de los problemas latentes, y los lanzamientos posteriores pueden concentrarse en evitar que se ingresen los defectos y en buscar clases de vulnerabilidades recientemente descubiertas.
- Será más fácil aplicar el SDL a un lenguaje que produzca código administrado, como C#, Microsoft Visual Basic .NET o Java, que a un código no administrado como C o C ++. Aunque no se garantiza que el código administrado sea seguro, existen clases de vulnerabilidades de seguridad que los desarrolladores no pueden introducir en el código administrado. Por ejemplo, no es posible escribir un desbordamiento de búfer en C#, como en C o C ++, aunque todavía es posible que un desarrollador escriba una aplicación en C# que sea vulnerable a un ataque de inyección SQL. El hecho de que no sea posible incluir ciertos tipos de vulnerabilidades en una aplicación escrita en código administrado ahorra el esfuerzo que de otro modo se requeriría para buscarlos, encontrarlos y eliminarlos.

- Las herramientas son casi siempre más eficaces y eficientes que una búsqueda manual de vulnerabilidades de implementación. Esto no significa, que tales herramientas sean una solución y encuentren todos los errores de seguridad. Las herramientas pueden analizar grandes cantidades de código rápidamente, más rápido de lo que podría hacerlo un humano, pero las herramientas no son un reemplazo para los humanos.
- Las capacitaciones ayudan a reducir el costo de la seguridad y motivan a los diseñadores y desarrolladores a producir software más seguro en primer lugar. Las revisiones del código, las herramientas y las pruebas encontrarán menos problemas, y la implementación del SDL será más rápida y económica.
- Los diseños seguros reducen la cantidad de errores a nivel de código que resultan en vulnerabilidades de seguridad y reducen la gravedad de las vulnerabilidades que quedan. Como resultado, revisiones de códigos, herramientas y pruebas encontrarán menos vulnerabilidades de seguridad que deben abordarse.

Eficacia de los elementos del SDL

El SDL se compone de una gran cantidad de subprocesos de componentes que se distribuyen a lo largo del ciclo de vida de desarrollo de software. Al equipo de SDL se le ha pedido que priorice esos subprocesos en términos de eficacia, los cuales tienen el mayor beneficio, y lo que se ha intentado y se ha encontrado menos efectivo.

El SDL sigue siendo un proceso relativamente nuevo en Microsoft, por lo que hasta ahora no ha habido casos en los que se haya eliminado un componente del proceso.

- **Fases de la Metodología SDL**

En el artículo ***Building More Secure Commercial Software: The Trustworthy Computing Security Development Lifecycle*** (Lipner, 2005), menciona que una organización que busca desarrollar software seguro debe asumir la responsabilidad de garantizar que su población de ingeniería reciba la educación adecuada. En

Microsoft, todo el personal involucrado en el desarrollo de software debe pasar por un entrenamiento anual de "actualización de seguridad".

Fase de Requisitos:

La necesidad de considerar la seguridad "desde cero" es un principio fundamental del desarrollo de sistemas seguros. Durante la fase de requisitos, el equipo del producto se pone en contacto con el equipo de seguridad central para solicitar la asignación de un asesor de seguridad que actúa como punto de contacto, recurso y guía a medida que avanza la planificación.

Entre las funciones del asesor de seguridad:

- Ayudar al equipo del producto revisando los planes
- Aporta con recomendaciones y asegurando que el equipo de seguridad planifique los recursos apropiados para respaldar el cronograma del equipo del producto.
- El asesor de seguridad es el punto de contacto del equipo de producto y el equipo de seguridad desde el inicio del proyecto hasta la finalización de la revisión de seguridad final y el lanzamiento del software.

Esta fase es la oportunidad para que el equipo de producto considere cómo se integrará la seguridad en el proceso de desarrollo, identifique objetivos clave de seguridad y, de lo contrario, maximice la seguridad del software y minimice la interrupción de planes y cronogramas. Como parte de este proceso, el equipo debe considerar cómo se integrarán las características de seguridad y las medidas de seguridad de su software con otro software que probablemente se utilizará junto con su software.

Fase de diseño:

Identifica los requisitos generales y la estructura del software. Desde una perspectiva de seguridad, los elementos clave de la fase de diseño son:

- Definir la arquitectura de seguridad y las pautas de diseño: definir la estructura general del software desde una perspectiva de seguridad, e identificar aquellos componentes cuyo correcto funcionamiento es esencial para la seguridad del sistema.

- Documentar los elementos de la superficie de ataque del software dado que el software no logrará una seguridad perfecta, es importante que solo las funciones que serán utilizadas por la gran mayoría de los usuarios estén expuestas por defecto a todos los usuarios y que esas características se instalen con el nivel de privilegio mínimo factible.
- Conducir modelos de amenazas. Utilizando una metodología estructurada, el proceso de modelado de amenazas identifica las amenazas que pueden dañar a cada activo y la probabilidad de que se haga daño (una estimación del riesgo), y ayuda a identificar contramedidas que mitiguen el riesgo.

Fase de Implementación

Durante la fase de implementación, el equipo del producto codifica, prueba e integra el software. Los pasos tomados para eliminar fallas de seguridad o evitar su inserción inicial reducen significativamente la probabilidad de que las vulnerabilidades de seguridad lleguen a la versión final del software.

Los elementos del SDL que se aplican son:

- Desde la tarea del modelo de amenaza, entienda cuáles son los componentes de alto riesgo.
- Aplicar estándares de codificación y prueba.
- Aplicar herramientas de prueba de seguridad, incluidas herramientas de prueba de fuzz.
- Aplicar herramientas de análisis de código de análisis estático.
- Llevar a cabo revisiones del código de seguridad

Fase de verificación

Es el punto en el que el software está funcionalmente completo y entra en la prueba beta. Microsoft introdujo el impulso de seguridad durante la fase de verificación de Windows Server 2003 y varias otras versiones de software a principios de 2002. El objetivo principal del impulso de seguridad es revisar el código que se desarrolló o actualizó durante la fase de implementación y el "código heredado" que no fue modificado.

Fase de lanzamiento

Durante la fase de lanzamiento, el software debe estar sujeto a una Revisión Final de Seguridad ("FSR").

El FSR es una revisión independiente del software realizado por el equipo de seguridad central para la organización. Las tareas incluyen la revisión de errores que inicialmente se identificaron como errores de seguridad, pero en análisis posteriores se determinó que no tenían impacto en la seguridad, para garantizar que el análisis se realizara correctamente. Un FSR también incluye una revisión de la capacidad del software para resistir las vulnerabilidades recientemente reportadas que afectan software similar. Un FSR para una versión de software importante requerirá pruebas de penetración y, potencialmente, el uso de contratistas externos de revisión de seguridad para complementar el equipo de seguridad.

Fase de Soporte y Servicio

A pesar de la aplicación del SDL durante el desarrollo, las prácticas de desarrollo de última generación aún no admiten software de envío que esté completamente libre de vulnerabilidades, y existen buenas razones para creer que nunca lo harán. Los equipos de productos deben prepararse para responder a las vulnerabilidades descubiertas recientemente en el software de envío.

Parte del proceso de respuesta implica la preparación para evaluar los informes de vulnerabilidades y lanzar avisos de seguridad y actualizaciones cuando corresponda. El otro componente del proceso de respuesta es llevar a cabo una autopsia de cada vulnerabilidad reportada y tomar las medidas necesarias. Según la naturaleza y la gravedad de las vulnerabilidades notificadas, esta acción puede incluir la repetición de fases anteriores del proceso para abordar nuevas amenazas.

En el documento publicado por Microsoft titulado ***Implementación simplificada del proceso SDL de Microsoft*** (Implementación simplificada del proceso SDL de Microsoft, 2010)**,** indica que el proceso SDL de Microsoft es un conjunto de actividades de seguridad obligatorias, que se presentan en el orden en que deben

llevarse a cabo y se agrupan por cada una de las fases de un ciclo de vida de desarrollo de software tradicional. Muchas de las actividades descritas aportarían ciertos beneficios en materia de seguridad si se implementaran de manera independiente. Sin embargo, la experiencia en Microsoft demuestra que las actividades de seguridad realizadas como parte de un proceso de desarrollo de software aportan mayores beneficios que las actividades implementadas de manera poco sistemática o de modo ad hoc. A continuación las actividades que involucra implementar la metodología SDL según Microsoft.

El autor trata a las Fases de SDL como **Actividades de seguridad simplificadas de SDL**

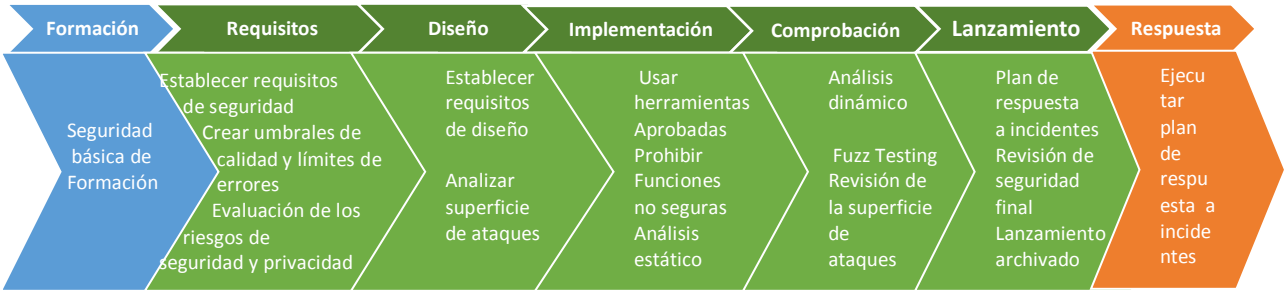


Figura 6.3 Ciclo de vida de desarrollo de software de Microsoft simplificado
Fuente: Investigador

Requisitos anteriores a SDL: formación en materia de seguridad

Procedimiento 1 de SDL: requisitos de formación

En la formación de requisitos trata de que el equipo de desarrollo debe estar bien formado; es decir, deben ser capacitados sobre conceptos básicos y estar al tanto de las últimas tecnologías sobre seguridad y privacidad. Se recomienda que las personas que están implicadas directamente con el desarrollo deben asistir al menos una vez al año a capacitaciones sobre seguridad en el código fuente.

Procedimiento:

Para cumplir con este procedimiento el responsable de TI realiza anualmente una planificación sobre las capacitaciones del personal a su cargo. En el área de

desarrollo debe incluir temas de capacitación sobre seguridad de desarrollo de software.

Primera fase: requisitos

En esta primera fase se establecen los requisitos de seguridad que tendrá el proyecto, así como la creación de los límites de errores que no son otra cosa que los umbrales de calidad que permitirán integrar la seguridad en el proceso de desarrollo. Además, aquí se evalúan los riesgos de seguridad y privacidad que se tomará en cuenta para maximizar la seguridad del software. A continuación, los procedimientos:

Procedimiento 2 de SDL: requisitos de seguridad

En este paso se considera a la seguridad y la privacidad como un aspecto fundamental en el desarrollo de un sistema seguro. Aquí es donde se definen los requisitos de fiabilidad de un proyecto de software. Con esto, el equipo de desarrollo puede identificar los principales hitos y resultados, además de integrar la seguridad y la privacidad de tal manera que los planes y programaciones no tengan gran impacto. Se puede también aquí especificar requisitos de seguridad mínimos, especificar e implementar un sistema de seguimiento de los elementos de trabajo y de las vulnerabilidades de seguridad.

Procedimiento 3 de SDL: umbrales de calidad y límites de errores

Se usan umbrales de calidad y límites de errores para establecer niveles mínimos aceptables de calidad en cuanto a la seguridad y privacidad. Definir esto en el inicio del proyecto es necesario para cuando existan problemas de seguridad, así los equipos podrán identificar y corregir los errores de seguridad durante el desarrollo. Primero el equipo de proyecto debe negociar los umbrales de calidad (por ejemplo, todas las advertencias del compilador se deben evaluar y corregir antes de proteger el código) para cada fase de desarrollo; seguido, el asesor de seguridad debe aprobarlos y si es necesario puede agregar aclaraciones específicas del proyecto así como requisitos de seguridad más estrictos.

Un límite de errores es un umbral de calidad que se debe aplicar a todo el proyecto de desarrollo de software. Son utilizados para definir los umbrales de gravedad de las vulnerabilidades de seguridad.

Procedimiento 4 de SDL: evaluación de los riesgos de seguridad y privacidad

Realizar procesos de evaluación de riesgos de seguridad (SRA) y de privacidad (PRA), son necesarios para identificar los aspectos funcionales del software, los cuales requieren una revisión exhaustiva. Las evaluaciones contener:

1. (Seguridad) Partes del proyecto que van a requerir modelos de riesgos antes del lanzamiento.
2. (Seguridad) Las partes del proyecto que van a requerir revisiones del diseño de seguridad antes del lanzamiento.
3. (Seguridad) Partes del proyecto (si procede) que van a requerir pruebas de penetración por parte de un grupo externo al equipo de proyecto y acordado mutuamente.
4. (Seguridad) Requisitos de análisis o de pruebas adicionales que el asesor de seguridad considera necesarios para mitigar los riesgos de seguridad.
5. (Seguridad) Ámbito específico de los requisitos en materia de pruebas de exploración de vulnerabilidades mediante datos aleatorios.
6. (Privacidad) El nivel de impacto sobre la privacidad que puede ser alto, medio o bajo

Segunda fase: diseño

En esta fase se definen las directrices del diseño y requisitos del software. Se analiza la superficie de ataques lo que permitirá detectar las instancias en las que el software es más susceptible de recibir ataques. Se realiza un modelado de amenazas para identificar las amenazas que pueden dañar al software. Los procedimientos son:

Procedimiento 5 de SDL: requisitos de diseño

Esta etapa incluye la creación de especificaciones de diseño en cuestión de seguridad y privacidad, la revisión de las especificaciones y la especificación de requisitos mínimos de diseño criptográfico. Las especificaciones de diseño deben describir las características de seguridad o de privacidad que van a estar directamente expuestas a

los usuarios, por ejemplo, la autenticación del usuario para obtener acceso a datos específicos o que necesitan autorización del usuario para usar una característica con un alto riesgo para la privacidad. Además, todas las especificaciones de diseño deben describir cómo se implementa de manera segura toda la funcionalidad proporcionada por una característica o función determinada. Se recomienda validar las especificaciones de diseño con la descripción funcional de la aplicación.

Procedimiento 6 de SDL: reducción de la superficie de ataques

Se relaciona con los modelos de riesgos, la reducción de la superficie de ataques significa reducir el riesgo, minimizando las oportunidades de los atacantes de encontrar un posible punto débil o una posible vulnerabilidad. Para reducir la superficie de ataques, se cierra o se restringe el acceso a los servicios del sistema, se aplica el principio de privilegios mínimos o se usan en la medida de lo posible defensas por capas.

Procedimiento 7 de SDL: modelos de riesgos

Los modelos de riesgos son utilizados en entornos donde existe un riesgo significativo para la seguridad. Este procedimiento permite a los equipos de desarrollo considerar, documentar y describir de manera estructurada las implicaciones para la seguridad de los diseños. Los modelos de riesgos también permiten estudiar los problemas de seguridad en el nivel de los componentes o aplicaciones. La creación de un modelo de riesgos es un trabajo de equipo que implica a los administradores de programas o proyectos, desarrolladores y evaluadores; además, es la principal tarea de análisis de seguridad que se realiza en la fase de diseño del software.

Modelado de Amenazas

Según el artículo titulado ***MODELADO DE AMENAZAS, UNA TÉCNICA DE ANÁLISIS Y GESTIÓN DE RIESGO ASOCIADO A SOFTWARE Y APLICACIONES*** (Barba Olivares, 2017)**.** Define al modelado de amenazas es una técnica de comprobación cuyo objetivo es ayudar a identificar y planificar de forma correcta, la mejor manera de mitigar las amenazas de una aplicación mediante un

enfoque moderno de análisis de gestión de riesgos, y la implementación de medidas o controles que contribuyan a mejorar la seguridad.

Etapas del Modelado de Amenazas

Según el modelo propuesto por Microsoft, se tiene las siguientes etapas:

Etapa 1: Conformar un grupo de análisis de riesgo:

Aquí se conforma un equipo de trabajo para dar seguimiento a las fases del modelado; y encargado de identificar:

- Los objetivos de seguridad: Determinar los objetivos ayudará a clarificar y cuantificar el esfuerzo que se debe dedicar en las actividades subsiguientes.
- Crear una descripción general de la aplicación, con la cual se trata de identificar los actores y características más importantes que interactúan con la aplicación, esto con el fin de lograr reconocer cuales podrían llegar a ser las amenazas más significativas (Barba Olivares, 2017).

Etapa 2: Descomponer la aplicación e identificar componentes claves

Se aplica DFD (diagrama de flujo de datos) con el fin de conocer la arquitectura de la aplicación y descomponerla, de tal manera que se pueda identificar las principales funcionalidades y los módulos susceptibles a provocar un mayor impacto en la seguridad (Vulnerabilidades conocidas) (Barba Olivares, 2017).

Etapa 3: Determinar las amenazas de cada componente de la aplicación

En esta etapa se trata de adoptar el punto de vista del atacante, basándose no solo en las características propias de un componente de la aplicación, sino también en la interacción con otros componentes. Para esta etapa (de acuerdo con el modelo propuesto por Microsoft), se encuentra el método de clasificación STRIDE (Barba Olivares, 2017).

Método de clasificación STRIDE.

Este término es el acrónimo de “Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, Elevation of privilege".

Para seguir el método STRIDE, se descompone el sistema en diferentes componentes, seguido de analizar cada uno, con el objetivo de comprobar si es susceptible de sufrir amenazas; después se proponen acciones que traten de mitigarlas y se repite el proceso hasta llegar a una situación cómoda con las amenazas restantes (Barba Olivares, 2017).

- ✓ Spoofing Identity (Suplantación de identidad): Esta categoría de amenazas indica básicamente que los usuarios no deberían ser capaces de hacerse pasar por otros usuarios. Es importante enfocarse en un adecuado sistema de autenticación y poner mucha atención en los temas de uso de sesión con el fin de evitar robos por medio del uso de éstas.
- ✓ Tampering with Data (Manipulación de datos): Esto se da por la carencia de medidas preventivas como, es el filtrado y la validación de los datos enviados y recibidos de los usuarios de la aplicación. En el caso específico de las aplicaciones desarrolladas para un entorno web, se puede dar a manera de ejemplo, las vulnerabilidades conocidas, tales como la inyección de sentencias SQL o el XSS; en general, en aquellos casos en los que las aplicaciones proporcionan al usuario, datos que no son obtenidos directamente de la propia aplicación, se realiza algún tipo de cálculo con ellos y posteriormente se almacenan, teniendo en cuenta que estos datos pueden ser susceptibles de una manipulación efectuada por un usuario malintencionado que disponga de las herramientas adecuadas.
- ✓ Repudiation (Repudio): Aquí se trata de establecer un nivel adecuado del seguimiento de las acciones realizadas por los usuarios de la aplicación; con el fin de evitar que aparezcan situaciones no deseadas se debe intentar garantizar él no repudio de los usuarios.
- ✓ Information Disclosure (Revelación de información): La existencia de vulnerabilidades en una aplicación que permitan extraer información sensible, es un factor claro de riesgo que en ocasiones puede derivar en pérdidas económicas, así como también, en la disminución de la confianza y reputación de cara a posibles o ya existentes usuarios, clientes, proveedores o inversores.

- ✓ Denial of Service (Denegación de servicio): A la hora de diseñar una aplicación, o en el momento de añadir una nueva funcionalidad, es conveniente evitar aquellas situaciones que puedan devengar en la consecución de un ataque de denegación de servicio. Por esta razón es conveniente proporcionar un uso racional de los recursos con los que interactúa la aplicación, ya sea evitando cálculos complejos, búsquedas intensivas o el acceso a inserción de archivos de gran tamaño a usuarios no permitidos.
- ✓ Elevation of Privilege (Elevación de privilegios): En el caso de que la aplicación o el sistema proporcionen diferentes niveles de privilegio en función de los distintos tipos de usuarios, todas las acciones que lleven al uso de privilegios deben ser filtradas a través de un mecanismo adecuado de autorización. Este método de validación de los privilegios deberá ser suficientemente robusto para impedir un posible escalamiento de privilegios.

Etapa 4: Asignar valor a cada amenaza

Se trata de calcular el riesgo asociado a cada amenaza, esto nos permite identificar qué respuesta dar para cada amenaza y priorizarlas según el riesgo que representa para la aplicación. Para esta etapa (de acuerdo con el modelo propuesto por Microsoft) se encuentra el método de puntuación DREAD (Barba Olivares, 2017).

Método de puntuación DREAD.

Una vez que se tiene identificada la lista de amenazas, el siguiente paso consiste en puntuarlas de acuerdo con el riesgo que suponen. Esto permite priorizar las actuaciones a efectuar para mitigar el riesgo. Comúnmente un riesgo se puede cuantificar, como el resultado de multiplicar la probabilidad de que la amenaza se produzca por el impacto (Barba Olivares, 2017).

La utilización del método DREAD se basa en las siguientes cuestiones:

Criterio	Definición
Damage potential (Daño potencial)	¿Cuál es el daño que puede originar la vulnerabilidad si llega a ser explotada?
Reproducibility (Reproducibilidad)	¿Es fácil reproducir las condiciones que propicien el ataque?
Exploitability (Explotabilidad)	¿Es sencillo llevar a cabo el ataque?
Affected users (Usuarios afectados)	¿Cuantos usuarios se verían afectados?
Discoverability (Descubrimiento)	¿Es fácil encontrar la vulnerabilidad?

Tabla 6.2. Criterio DREAD
Elaborado por: Investigador

Se establece un sistema de puntuación para calificar a los riesgos con las siguientes ponderaciones:

Riesgo	Puntuación	Rango
Alto	3	12-15
Medio	2	8-11
Bajo	1	5-7

Tabla 6.3. Calificación Riesgos DREAD
Elaborado por: Investigador

Etapa 5: Definir cómo responder a las amenazas

Frente a las amenazas se pueden dar diferentes respuestas, basándose en el valor del riesgo asociado a cada una, por esta razón, finalmente después de haber puntuado el riesgo de la aplicación, se tendrá una lista priorizada de amenazas para mitigar; como regla general se deberá trabajar en aquellas amenazas que supongan un mayor riesgo primero (Barba Olivares, 2017).

Herramientas para Modelado De Amenazas (Barba Olivares, 2017)

En la actualidad existen algunas herramientas de apoyo (proporcionadas por las principales entidades) que ayudarán a construir un modelado de amenaza de una manera sencilla. Dentro de las principales se encuentran:

- SDL threat modeling tool (SDL).
- Threat Analysis and Modeling tool (TAM).
- Consultative Objective Risk Analysis System (CORAS).

Estas herramientas proporcionan bibliotecas de ataques predefinidos y recomiendan cuales deberían ser las mitigaciones eficaces para los diferentes tipos de ataque asociado a cada amenaza.

Threat Modeling Tool (TMT)

Threat Modeling Tool (TMT) es un elemento central de Microsoft Security Development Lifecycle (SDL).

- Permite a los arquitectos de software identificar y mitigar posibles problemas de seguridad en una etapa temprana, cuando son relativamente fáciles y rentables de resolver. Como resultado, reduce considerablemente el costo total de desarrollo.
- La herramienta es diseñada teniendo en cuenta a expertos que no son de seguridad, facilitando el modelado de amenazas para todos los desarrolladores al proporcionar una guía clara sobre la creación y el análisis de modelos de amenazas.
- La herramienta SDL funciona mediante el uso de representaciones gráficas, la cual agiliza la construcción de modelado de amenazas.
- La herramienta da soporte con adaptaciones construidas de tal forma, que los componentes se representan como burbujas de procesos, los roles de usuarios dependencias externas como entidades, y los datos de forma directa.

Tercera fase: implementación

En esta fase se realizan pruebas con herramientas que permiten el análisis de código estático. Se aplica estándares de codificación para prevenir la inclusión de errores y

se produzcan vulnerabilidades. Se realiza una revisión general del código. Entre los procedimientos, se tiene:

Procedimiento 8 de SDL: usar herramientas aprobadas

Todos los equipos de desarrollo deben definir y publicar una lista de las herramientas aprobadas y de las comprobaciones de seguridad asociadas, como las advertencias y las opciones del compilador o del vinculador. Esta lista la debe aprobar el asesor de seguridad del equipo de proyecto. Se recomienda que las herramientas a utilizar por los equipos de desarrollo, sean de versiones más reciente a fin de aprovechar las nuevas protecciones y funciones de análisis de seguridad.

Procedimiento 9 de SDL: prohibir funciones no seguras

Los equipos de proyecto deben analizar todas las funciones y API que se van a utilizar en un proyecto de desarrollo de software y descartar las que consideran inseguras. Luego deben utilizar archivos de encabezado (por ejemplo, banned.h y strsafe.h), compiladores más recientes o herramientas de análisis de código para comprobar si hay funciones prohibidas en el código (incluido código heredado si procede) y reemplazarlas por otras alternativas que sean seguras.

Procedimiento 10 de SDL: análisis estático

El análisis estático del código fuente ayuda al equipo de seguridad a revisar el código de seguridad de forma escalable y contribuye a asegurar que se observan las directivas de codificación segura. Sin embargo se debe tomar en cuenta que esto no reemplaza a la revisión de código manual, por tanto se debe ampliar las herramientas de análisis por otras o la revisión humana.

Cuarta fase: comprobación

En esta fase ya se tiene una versión beta del software y se pueden aplicar el análisis dinámico, pruebas de fuzz. También se procede con las revisiones al código de seguridad a parte de las realizadas en la fase anterior para asegurar que se cumplan con los requisitos. Entre los procedimientos:

Procedimiento 11 de SDL: análisis dinámico de los programas

Se requiere la comprobación de los programas de software en tiempo de ejecución para asegurar que su funcionalidad sea acorde con el diseño. Aquí se puede identificar fallas en la memoria, se analiza problemas con los roles de usuario u otros tipos de problemas de seguridad. El proceso SDL usa herramientas en tiempo de ejecución como AppVerifier, junto con otras técnicas como pruebas de exploración de vulnerabilidades mediante datos aleatorios, para lograr los niveles de cobertura deseados de las pruebas de seguridad.

Procedimiento 12 de SDL: pruebas de exploración de vulnerabilidades mediante datos aleatorios

Es un tipo de pruebas de análisis dinámico, que buscar producir errores en los programas introduciendo datos aleatorios o de formato incorrecto en una aplicación. Esta estrategia se deriva del uso previsto de la aplicación y de sus especificaciones funcionales y de diseño. La cantidad y la duración de este tipo de pruebas dependerá del criterio del asesor de seguridad.

Procedimiento 13 de SDL: revisión de los modelos de riesgos y de la superficie de ataques

Una vez completado el código de la aplicación es muy importante que se revisen los modelos de riesgos y la medición de la superficie de ataques, puesto que en el transcurso del desarrollo pueden existir cambios en las especificaciones funcionales y de diseño. De esta forma se toma en cuenta los nuevos cambios y es posible mitigar los riesgos que producen los cambios.

Quinta fase: lanzamiento

En esta última fase se prepara un plan de respuesta a incidentes el cual podrá ser utilizado para cuando ocurra algún incidente y se registre de acuerdo a un formato indicado. También se realiza una revisión final del software de todas las actividades de seguridad que se han aplicado. Entre los procedimientos:

Procedimiento 14 de SDL: plan de respuesta a incidentes

De acuerdo a los requisitos de SDL al poner en producción un software se debe incluir un plan de respuesta a incidentes. El plan de respuesta a incidentes debe incluir:

- Un plan de respuesta a emergencias en el que se identifica el personal de ingeniería, marketing, comunicaciones y administración que representa el primer punto de contacto en caso de una emergencia de seguridad.
- Contactos con capacidad para tomar decisiones y disponibilidad las 24 horas del día y los 7 días de la semana.
- Planes de servicios de seguridad para código heredado de otros grupos dentro de la organización.
- Planes de servicios de seguridad para código de terceros bajo licencia, incluidos nombres de archivo, versiones, código fuente, datos de contacto de terceros y permiso contractual para realizar cambios (si procede).

Procedimiento 15 de SDL: revisión de seguridad final

La revisión de seguridad final es una inspección de todas las actividades de seguridad realizadas con una aplicación de software antes del paso a producción. Esta revisión la lleva a cabo el asesor de seguridad con ayuda del personal de desarrollo habitual y de los responsables de los equipos de seguridad y de privacidad. La revisión consiste en un estudio de los modelos de riesgos, las solicitudes de excepciones, los resultados de las herramientas y el rendimiento teniendo en cuenta los umbrales de calidad y los límites de errores previamente determinados. La revisión de seguridad final da lugar a uno de estos tres resultados:

- **Revisión de seguridad final superada.** Se han corregido y mitigado todos los problemas de seguridad y de privacidad identificados en la revisión de seguridad final.
- **Revisión de seguridad final superada con excepciones.** Se han corregido y mitigado todos los problemas de seguridad y de privacidad identificados en la revisión de seguridad final y/o todas las excepciones se han resuelto de modo satisfactorio. Los problemas que no se pueden resolver (por ejemplo, vulnerabilidades debido a problemas heredados del "nivel de diseño") se registran y se corrigen en la siguiente versión.

- **Revisión de seguridad final con remisión a una instancia superior.** Si un equipo no cumple todos los requisitos de SDL y el asesor de seguridad y el equipo de proyecto no alcanzan un acuerdo aceptable, el asesor de seguridad no podrá aprobar el proyecto, por lo que no se podrá realizar el paso a producción. Los equipos deberán intentar cumplir en la medida de lo posible todos los requisitos de SDL antes de proceder con el paso a producción del proyecto o remitir el asunto a instancias superiores para que tomen una decisión al respecto.

Procedimiento 16 de SDL: lanzamiento o archivado

Las versiones RTM (Release To Manufacturing) y RTW (Release To Web) dependen de la finalización del proceso SDL. El asesor de seguridad asignado a la versión debe certificar (mediante la revisión de seguridad final y otros datos) que el equipo de proyecto ha cumplido los requisitos de seguridad. De manera similar, para todos los productos que tienen al menos un componente con el nivel de impacto en la privacidad P1, el asesor de privacidad del proyecto debe certificar que el equipo de proyecto ha cumplido los requisitos de privacidad para que se pueda enviar el software.

Actividades de seguridad opcionales

Estas actividades se las puede ejecutar cuando se determina que el software es de carácter crítico. Puede ser propuesta por el asesor de seguridad con el fin de asegurar un mayor nivel de análisis de seguridad para determinados componentes de software. Entre esas actividades se tiene:

Revisión de código manual

La revisión de código manual lo puede realizar los miembros del equipo de seguridad y/o el asesor de seguridad, no solo con el uso de herramientas de vulnerabilidades sino centrarse en los componentes “críticos” de una aplicación.

Pruebas de penetración

Las pruebas de penetración consisten en un análisis de seguridad de caja blanca de un sistema de software que llevan a cabo profesionales en el ámbito de la seguridad simulando las acciones de un hacker. Estas pruebas tienen como objetivo detectar posibles vulnerabilidades debidas a errores de codificación, errores en la configuración del sistema u otros puntos débiles de la implementación operativa.

Análisis de vulnerabilidades de aplicaciones similares

En algunos casos, los análisis de vulnerabilidades de aplicaciones de software similares pueden arrojar luz sobre los problemas de diseño o implementación con los que se puede encontrar el software en fase de desarrollo.

6.7.3 Propuesta de implementación de la Metodología SDL

El aplicativo que será utilizado para la aplicación de la Metodología SDL es Chequera Virtual. Este sistema está desarrollado en la plataforma Visual Studio 2003 con el lenguaje de programación C#. Utiliza como motor de base de datos SQL Server 2012.

La presente propuesta de implementación de la Metodología SDL está formada por 5 fases más la Formación o Entrenamiento que es un proceso previo al inicio del desarrollo de la metodología.

Previo al inicio de las fases se debe considerar el entrenamiento o capacitación que deben recibir los programadores anualmente sobre seguridad para poder desarrollar un software seguro y en lo posible se reduzca en nivel de riesgo. En la primera fase que corresponde a los **Requisitos**, se va detallando los requerimientos mínimos de seguridad que va a tener la aplicación, contemplando los estándares de buenas prácticas de programación, así como la evaluación de los riesgos. En la segunda fase de **Diseño** se establecen validaciones en entrada de datos y encriptamiento de la información que se considera crítica y que viaja desde el cliente hacia el servidor y viceversa, se incluye también en análisis de riesgos; para ellos se cuenta con un modelado de amenazas que también posee sus etapas hasta llegar a determinar el nivel de riesgo y dar una solución a las amenazas detectadas. En la tercera fase de **Implementación** se certifica el uso de buenas prácticas de programación y se realiza una breve revisión al código, se aplica también herramientas de análisis de código

estático como: Fxcop, BinScope y Code Analysis. En la cuarta fase de **Comprobación** se realiza una serie de pruebas de exploración de vulnerabilidades (Fuzz Testing) en tiempo de ejecución, insertando datos para producir errores y que puedan ser controlados a tiempo. Para la quinta fase de **Lanzamiento** se indica quiénes van a conformar el equipo de respuesta a incidentes; de igual forma se propone un plan de respuesta a incidentes para lo cual se cuenta con un formulario.

El aplicativo Chequera Virtual está expuesto al internet y puede ser utilizado por todos los socios que mantengan cuentas en la Cooperativa. El aplicativo dispone de transacciones tanto dentro de la misma entidad, como hacia otras instituciones financieras. Adicional se puede realizar renovaciones de pólizas, recargas de celular, realizar créditos en línea y consultar sobre estados de cuenta, pólizas y créditos.

A continuación un diagrama sobre el proyecto Chequera Virtual, en donde se muestra los elementos involucrados:

- La parte del acceso al sistema a través del Internet por medio de un navegador.
- Validaciones de datos de entrada (autenticación)
- El servidor Windows 2003 Server donde se encuentra alojado el aplicativo, el cual contiene: interfaz, el negocio, el acceso a los datos.
- Servidor de base de datos con el motor SQL Server 2012
- Roles de acceso a la base por parte del aplicativo

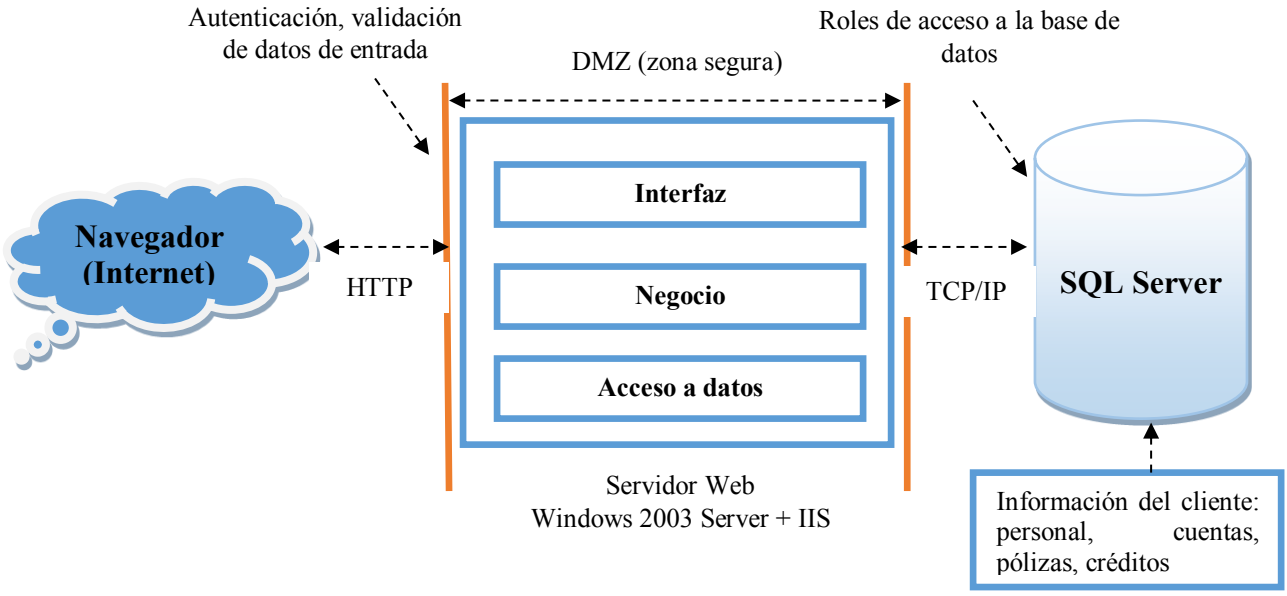


Figura 6.4 Descripción general de la aplicación
Elaborado por: Investigador

Formación o Entrenamiento

Este es el primer procedimiento que se realiza previo a dar inicio con las fases de la metodología SDL.

Aquí se resalta que los programadores de la Cooperativa han recibido capacitaciones al menos una vez al año sobre seguridad, en lo que se refiere al Ciclo de Desarrollo Seguro SDL. Esto se ha llevado a cabo con la empresa consultora LOGICSTUDIO de la ciudad de Quito, que desde hace tiempo atrás se ha encargado de capacitar al personal de desarrollo de la Cooperativa.

Se toma como ejemplo el tema de la capacitación denominado Desarrollo Seguro de Aplicaciones SDL. Este entrenamiento fue diseñado para desarrolladores con experiencia en construcción de soluciones tecnológicas y que necesitan formalizar sus técnicas de ingeniería de software para desarrollar soluciones seguras y confiables.

Se utiliza conceptos basados en Security Development Lifecycle SDL dentro del proceso formal de construcción de soluciones de tecnología, además de la realización de ejercicios prácticos para mostrar la interacción de los componentes de desarrollo con la metodología de seguridad revisada.

Primera Fase: Requisitos

En esta fase se establecerán los requisitos mínimos de seguridad y privacidad. Como requisitos iniciales tenemos:

- Establecer la longitud mínima para las contraseñas
- Solicitud de cambio de contraseña en el primer inicio de sesión
- Expiración de las contraseñas
- Combinación de números, letras y caracteres especiales para la creación de la contraseña
- Bloqueo de usuario luego de un reiterados intentos de ingresos fallidos
- No repetición de ultimas contraseñas utilizadas

- Almacenamiento cifrado de contraseñas, utilizando algoritmos de encriptación
- Desconexión de las sesiones luego de un tiempo establecido

Umbrales de Calidad:

- Al momento de realizar la compilación del proyecto, no se debe dejar sin resolver ninguna advertencia que el compilador presente, por tanto todas esas advertencias deben ser evaluadas y corregidas a tiempo.
- Controlar que no exista duplicidad de código, para esto se debe utilizar la herencia en las clases.
- No respetar estándares de programación y mejores prácticas
- Se debe incluir comentarios en el código fuente para especificar la funcionalidad de los métodos y/o funciones.

Evaluación de Riesgos: Se evalúa riesgos como:

Inyección SQL.- Esto por lo general ocurre cuando se tiene campos de entrada de texto. En este caso se puede:

- Emplear el uso de herramientas de análisis de código
- Realizar un análisis de código automatizado
- Verificar que se separe la información de la consulta en el uso de intérpretes
- Evitar consultas dinámicas
- Hacer uso de APIs seguras para que no se tenga que hacer uso de intérpretes

Dependiendo la acción que vaya a realizar el usuario por lo general una transferencia, ahí se requiere ingresar otros datos que son confidenciales. Estos datos deben ser validados y controlados de manera que no se pueda ser víctima de inyección SQL.

Perdida de autenticación y gestión de sesiones.- Se refiere al manejo de contraseñas y sesiones de usuario. Se puede proteger:

- Encriptando las credenciales de los usuarios utilizando hash o algoritmos de cifrado.
- Estableciendo tiempos de duración de las sesiones de usuarios.

En el caso que estamos aplicando tenemos campos de entrada de texto para el inicio de sesión a la Chequera Virtual en donde se ingresa el usuario y contraseña. Esta información debe viajar encriptada.

Segunda Fase: Diseño

En esta fase se establecen requerimientos de diseño:

- Aquí se va a aplicar validaciones para los datos de entrada en donde se especifiquen controles que aseguren la validez de los datos ingresados mediante el aplicativo Chequera Virtual. Las validaciones son:
    - En los campos que son obligatorios, incluir el control *RequiredFiledValidator*
    - Identificar los campos que son numéricos y aplicar el control *RegularExpressionValidator*
    - Para que no existe claves personales duplicadas, aplicar el control *CustomValidator*
    - Para la clave de transferencia en el campo de texto se debe especifica el rango de caracteres, para esto utilizamos *RangeValidator*
- Control de procesamiento interno para minimizar los riesgos de fallas de procesamiento.
- Se implementará controles criptográficos cuando se trate del envío de mensajes que contengan información confidencial. En este caso se aplica la encriptación a:
    - Clave personal de Chequera Virtual
    - Pregunta secreta
    - Clave de transferencia

Modelado de Amenazas

Identificación de Amenazas

Con la visión general del funcionamiento del aplicativo Chequera Virtual, el personal de desarrollo puede ir identificando las principales amenazas a la que puede estar expuesto el aplicativo. A continuación un diagrama de Chequera Virtual en donde se denotan ciertas amenazas:

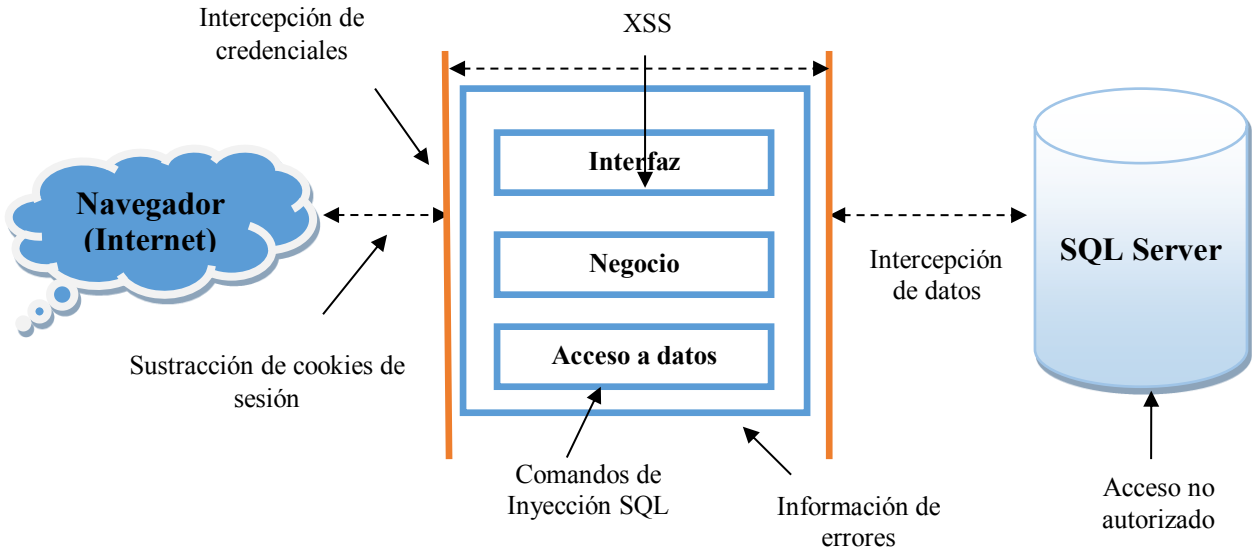


Figura 6.5 Diagrama de Amenazas
Elaborado por: Investigador

Entre las amenazas de acuerdo al diagrama y funcionamiento del aplicativo se tiene:

Amenaza 1: La contraseña del usuario será almacenado utilizando el método de cifrado hash. Aquí se detecta un ataque hacia la base donde están almacenadas las claves de los usuarios.

Amenaza 2: Pueden suplantar la identidad del usuario con la captura de cookies de sesión de usuarios, ya que el SSL implementado en la página web no evita que existan robos de cookies de sesiones y que las identidades de los usuarios sean suplantadas.

Amenaza 3: En los campos de entrada de texto se puede identificar el ingreso de inyección de comandos SQL que es posible que tenga graves afectaciones a la base de datos.

Amenaza 4: También la inyección de código por los conocidos ataques Cross-Site Scripting (XXS)

Amenaza 5: La información extra que se emite cuando existen errores en la aplicación pueden revelar nombres internos de la base de datos haciendo referencia a campos y/o tablas.
Amenaza 6: Los atacantes pueden encontrar las llaves de cifrado de los datos sensibles de los usuarios de la Chequera Virtual.

Etapas del Modelado de amenazas

Etapa 1: Conformar un grupo de análisis de riesgo
Para llevar a cabo el modelado de amenazas, se cuenta con la siguiente descripción de la aplicación:

El aplicativo Chequera Virtual proporciona a los clientes de la Cooperativa el servicio de transferencias tanto en la misma institución como a otras instituciones financieras (interbancarias), pagos de tarjetas, renovación de pólizas y generación de créditos en línea.

Los elementos que intervienen en la aplicación son: El usuario del sistema, que son los clientes los cuales acceden al aplicativo haciendo uso de cualquier navegador que tenga acceso a Internet. El aplicativo se encuentra en la dirección www.oscus.coop en la opción Chequera Virtual. En la pantalla principal de la Chequera Virtual ingresan el usuario y contraseña. Se realiza la petición a la base de datos para confirmar las credenciales. Una vez ingresado al sistema se puede realizar cualquier tipo de consulta y/o transacción.

La plataforma en la que se encuentra desarrollado es Visual Studio 2003, se encuentra alojado en un servidor Windows 2003 Server. El motor de base de datos es SQL Server 2012.

Etapa 2: Descomponer la aplicación e identificar componentes claves

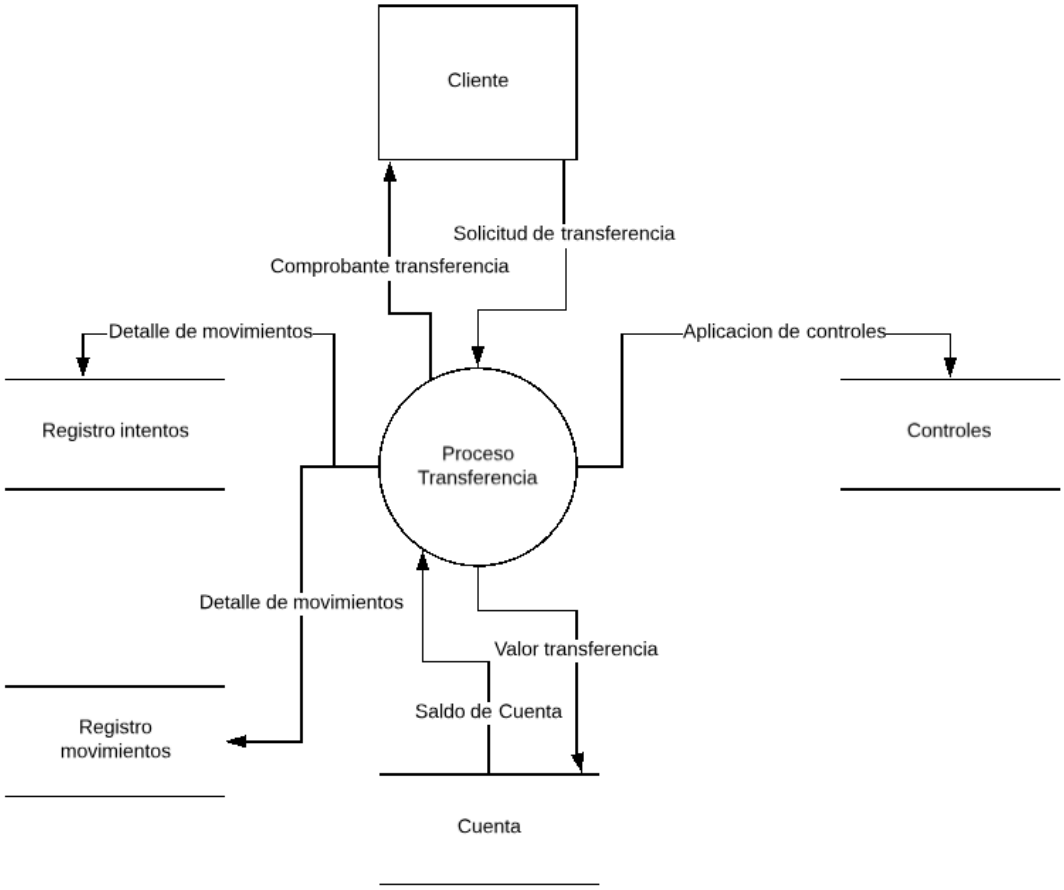


Figura 6.6 Diagrama de Flujo de Datos sin descomponer de proceso de Transferencia
Elaborado por: Investigador

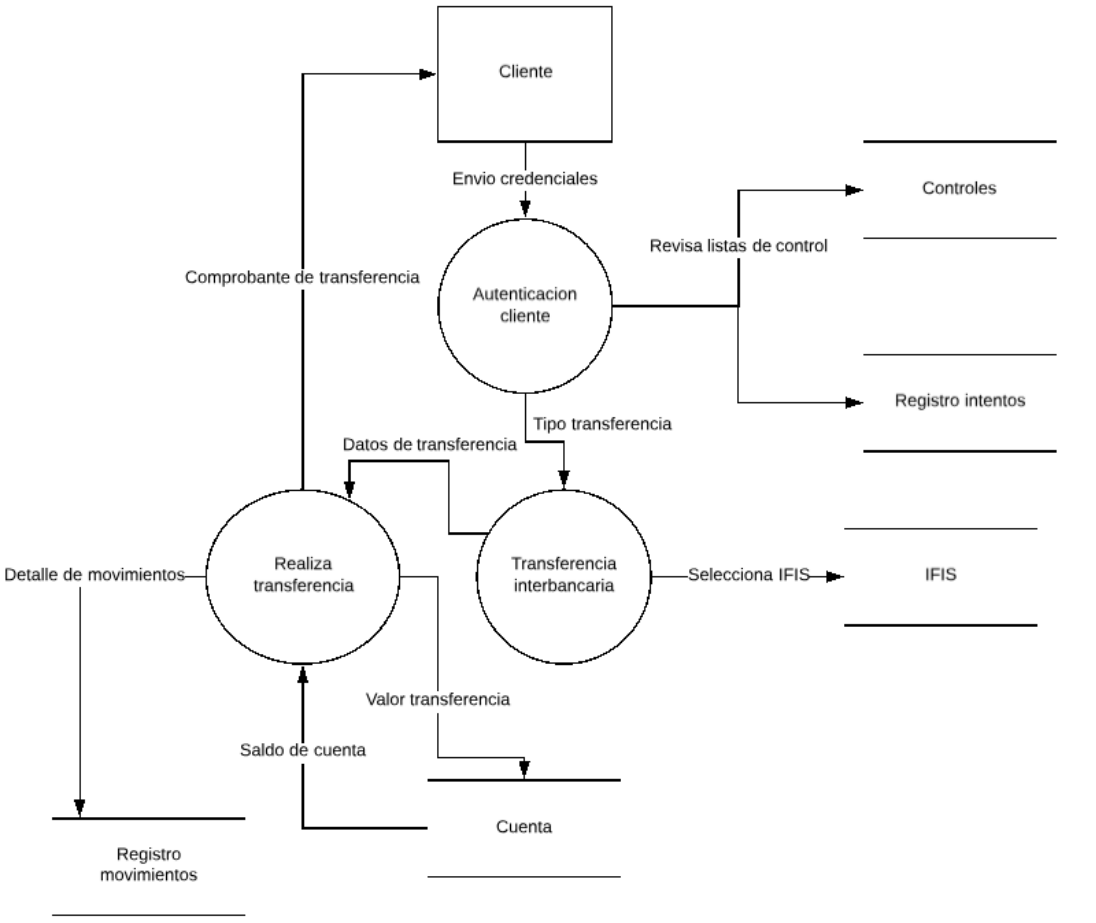


Figura 6.7 Diagrama de Flujo de Datos descompuesto de proceso de Transferencia

Elaborado por: Investigador

Etapa 3: Determinar las amenazas de cada componente de la aplicación

Se procede a clasificar las amenazas detectadas en la aplicación, de acuerdo a las categorías del STRIDE.

Categoría	Manipulación de datos
Descripción	Inyección de comandos SQL
Objetivo de la amenaza	Explotar datos en la base de datos
Nivel de riesgo	-
Técnicas de ataque	El atacante introduce comandos SQL en los campos de entrada de texto utilizado para formar una sentencia SQL.

Contramedidas	- Filtrar el contenido del campo de entrada de texto. - Utilizar un procedimiento almacenado que utilice parámetros para acceder a la base de datos. - Validar las entradas de texto, limitar la longitud del campo y verificar que cumpla con una serie de caracteres. - Utilizar API segura para minimizar el uso completo de intérpretes.

Tabla 6.4: Método de Clasificación STRIDE – Amenaza 1
Elaborado por: Investigador

Categoría	Suplantación de identidad
Descripción	Pérdida de autenticación y gestión de sesiones de usuario
Objetivo de la amenaza	Asumir la identidad de otros usuarios (interceptar sesiones de otros usuarios)
Nivel de riesgo	-
Técnicas de ataque	El atacante utiliza filtraciones y vulnerabilidades en las funciones de autenticación y gestión de sesiones.
Contramedidas	- Fijar un tiempo de desconexión (expiración) de las sesiones. - Re-autenticar a los usuarios cuando realicen transacciones de alto nivel. - Utilizar controles de manejo de sesiones que ofrece la tecnología ASP.Net - Los datos sensibles se deben guardar en una sesión o servidor.

Tabla 6.5: Método de Clasificación STRIDE – Amenaza 2
Elaborado por: Investigador

Categoría	Manipulación de datos
Descripción	Ataque de Cross-Site Scripting (XSS)
Objetivo de la amenaza	Explotar el intérprete del navegador

Nivel de riesgo	-
Técnicas de ataque	El atacante envía cadenas de texto que son secuencias de comandos hacia el navegador
Contramedidas	- Utilizar Ajax para actualizar dinámicamente la página. - Validar la entrada de datos y que los datos cumplan con las reglas del negocio. - Utilizar políticas de seguridad de contenido

Tabla 6.6: Método de Clasificación STRIDE – Amenaza 3
Elaborado por: Investigador

Categoría	Repudio
Descripción	Negar acciones realizadas por los usuarios
Objetivo de la amenaza	Dañar la integridad del sistema
Nivel de riesgo	-
Técnicas de ataque	- Cargar archivos dañinos al sistema - Eliminar archivos esenciales del sistema
Contramedidas	- Implementar logs para evidencias las acciones del usuario antes de efectuar transacciones.

Tabla 6.7: Método de Clasificación STRIDE – Amenaza 4
Elaborado por: Investigador

Categoría	Divulgación de información
Descripción	Exposición de información sensible en mensajes de error de la aplicación.
Objetivo de la amenaza	Obtener información de la base de datos como nombres de tablas, campos.
Nivel de riesgo	-
Técnicas de ataque	Ingreso de comandos maliciosos o aplicación de teclas combinadas para producir un error a nivel de aplicación.
Contramedidas	- Manejo de excepciones

	- Personalizar los mensajes de error

Tabla 6.8: Método de Clasificación STRIDE – Amenaza 5
Elaborado por: Investigador

Categoría	Denegación de servicio
Descripción	Sobrecarga de peticiones al sistema
Objetivo de la amenaza	Provocar caídas de un servicio o un sitio web
Nivel de riesgo	-
Técnicas de ataque	- Carga de archivos de gran tamaño - Ejecución intensiva de consultas hacia la base
Contramedidas	- Evitar cálculos complejos - Evitar búsquedas intensivas - Evitar que se carguen archivos de gran tamaño - En el caso de reiteración de contraseñas fallidas, se podría ejecutar un bloqueo de usuario.

Tabla 6.9: Método de Clasificación STRIDE – Amenaza 6
Elaborado por: Investigador

Categoría	Elevación de privilegios
Descripción	Asumir la identidad de un usuario con privilegios.
Objetivo de la amenaza	El atacante escala privilegios y accede a funciones no autorizadas del sistema.
Nivel de riesgo	-
Técnicas de ataque	Modifica la URL o algún parámetro a una función con privilegios.
Contramedidas	- Implementar un módulo de autorizaciones de acuerdo a las transacciones que vaya a realizar - Asignación de privilegios de acceso al sistema - Incluir logs para pistas de auditoría.

	- Negar los accesos por defecto a las páginas del sistema

Tabla 6.10: Método de Clasificación STRIDE – Amenaza 7
Elaborado por: Investigador

Etapa 4: Asignar valor a cada amenaza

De acuerdo al modelo propuesto por Microsoft

Determinar el nivel de riesgo de acuerdo al Método de puntuación DREAD

Se procede a puntuar la amenazas una a una para obtener su nivel de riesgo: Las amenazas identificadas son:

- ✓ Inyección de comandos SQL
- ✓ Pérdida de autenticación y gestión de sesiones de usuario
- ✓ Ataque de Cross-Site Scripting (XSS)
- ✓ Negar acciones realizadas por los usuarios
- ✓ Exposición de información sensible en mensajes de error de la aplicación
- ✓ Sobrecarga de peticiones al sistema
- ✓ El atacante asume la identidad de un usuario con privilegios

Amenaza	D	R	E	A	D	Total	Puntuación
Inyección de comandos SQL	3	3	3	2	3	14	Alto

Tabla 6.11: Puntuación DREAD - Amenaza 1
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Inyección de comandos SQL, se obtiene un total de 14; lo que significa que el nivel de riesgo es ALTO, de acuerdo a la tabla 6.11.

Amenaza	D	R	E	A	D	Total	Puntuación
Pérdida de autenticación y gestión de sesiones de usuario	3	3	3	2	3	14	Alto

Tabla 6.12: Puntuación DREAD - Amenaza 2
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Pérdida de autenticación y gestión de sesiones de usuario, se obtiene un total de 14; lo que significa que el nivel de riesgo es ALTO, de acuerdo a la tabla 6.12.

Amenaza	D	R	E	A	D	Total	Puntuación
Ataque de Cross-Site Scripting (XSS)	3	3	2	2	3	13	Alto

Tabla 6.13: Puntuación DREAD - Amenaza 3
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Ataque de Cross-Site Scripting (XSS), se obtiene un total de 13; lo que significa que el nivel de riesgo es ALTO, de acuerdo a la tabla 6.13.

Amenaza	D	R	E	A	D	Total	Puntuación
Negar acciones realizadas por los usuarios	2	2	2	1	1	8	Medio

Tabla 6.14: Puntuación DREAD - Amenaza 4
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Negar acciones realizadas por los usuarios, se obtiene un total de 8; lo que significa que el nivel de riesgo es MEDIO, de acuerdo a la tabla 6.14.

Amenaza	D	R	E	A	D	Total	Puntuación
Exposición de información sensible en mensajes de error de la aplicación	3	3	3	2	3	14	Alto

Tabla 6.15: Puntuación DREAD - Amenaza 5
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Exposición de información sensible en mensajes de error de la aplicación, se obtiene un total de 14; lo que significa que el nivel de riesgo es ALTO, de acuerdo a la tabla 6.15.

Amenaza	D	R	E	A	D	Total	Puntuación
Sobrecarga de peticiones al sistema	3	2	3	3	2	13	Alto

Tabla 6.16: Puntuación DREAD - Amenaza 6
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Sobrecarga de peticiones al sistema, se obtiene un total de 143 lo que significa que el nivel de riesgo es ALTO, de acuerdo a la tabla 6.16.

Amenaza	D	R	E	A	D	Total	Puntuación
Asumir la identidad de un usuario con privilegios	3	3	2	2	1	11	Medio

Tabla 6.17: Puntuación DREAD - Amenaza 7
Elaborado por: Investigador

Con la aplicación de método de puntuación DREAD para la amenaza Asumir la identidad de un usuario con privilegios, se obtiene un total de 11; lo que significa que el nivel de riesgo es MEDIO, de acuerdo a la tabla 6.17.

Asignación de nivel de riesgo

Una vez identificado el nivel de riesgo se procede a actualizar el resultado del riesgo en las amenazas que se encuentran en las categorías del STRIDE.

Categoría	Manipulación de datos
Descripción	Inyección de comandos SQL
Objetivo de la amenaza	Explotar datos en la base de datos
Nivel de riesgo	ALTO

Técnicas de ataque	El atacante introduce comandos SQL en los campos de entrada de texto utilizado para formar una sentencia SQL.
Contramedidas	- Filtrar el contenido del campo de entrada de texto. - Utilizar un procedimiento almacenado que utilice parámetros para acceder a la base de datos. - Validar las entradas de texto, limitar la longitud del campo y verificar que cumpla con una serie de caracteres. - Utilizar API segura para minimizar el uso completo de intérpretes.

Tabla 6.18: Resultado Nivel de Riesgo – Amenaza 1
Elaborado por: Investigador

Categoría	Suplantación de identidad
Descripción	Pérdida de autenticación y gestión de sesiones de usuario
Objetivo de la amenaza	Asumir la identidad de otros usuarios (interceptar sesiones de otros usuarios)
Nivel de riesgo	ALTO
Técnicas de ataque	El atacante utiliza filtraciones y vulnerabilidades en las funciones de autenticación y gestión de sesiones.
Contramedidas	- Fijar un tiempo de desconexión (expiración) de las sesiones. - Re-autenticar a los usuarios cuando realicen transacciones de alto nivel. - Utilizar controles de manejo de sesiones que ofrece la tecnología ASP.Net - Los datos sensibles se deben guardar en una sesión o servidor.

Tabla 6.19: Resultado Nivel de Riesgo – Amenaza 2
Elaborado por: Investigador

Categoría	Manipulación de datos
Descripción	Ataque de Cross-Site Scripting (XSS)
Objetivo de la amenaza	Explotar el intérprete del navegador
Nivel de riesgo	ALTO
Técnicas de ataque	El atacante envía cadenas de texto que son secuencias de comandos hacia el navegador
Contramedidas	- Utilizar Ajax para actualizar dinámicamente la página. - Validar la entrada de datos y que los datos cumplan con las reglas del negocio. - Utilizar políticas de seguridad de contenido

Tabla 6.20: Resultado Nivel de Riesgo – Amenaza 3
Elaborado por: Investigador

Categoría	Repudio
Descripción	Negar acciones realizadas por los usuarios
Objetivo de la amenaza	Dañar la integridad del sistema
Nivel de riesgo	MEDIO
Técnicas de ataque	- Cargar archivos dañinos al sistema - Eliminar archivos esenciales del sistema
Contramedidas	- Implementar logs para evidencias las acciones del usuario antes de efectuar transacciones.

Tabla 6.21: Resultado Nivel de Riesgo – Amenaza 4
Elaborado por: Investigador

Categoría	Divulgación de información
Descripción	Exposición de información sensible en mensajes de error de la aplicación.
Objetivo de la amenaza	Obtener información de la base de datos como nombres de tablas, campos.

Nivel de riesgo	ALTO
Técnicas de ataque	Ingreso de comandos maliciosos o aplicación de teclas combinadas para producir un error a nivel de aplicación.
Contramedidas	- Manejo de excepciones - Personalizar los mensajes de error

Tabla 6.22: Resultado Nivel de Riesgo – Amenaza 5
Elaborado por: Investigador

Categoría	Denegación de servicio
Descripción	Sobrecarga de peticiones al sistema
Objetivo de la amenaza	Provocar caídas de un servicio o un sitio web
Nivel de riesgo	ALTO
Técnicas de ataque	- Carga de archivos de gran tamaño - Ejecución intensiva de consultas hacia la base
Contramedidas	- Evitar cálculos complejos - Evitar búsquedas intensivas - Evitar que se carguen archivos de gran tamaño - En el caso de reiteración de contraseñas fallidas, se podría ejecutar un bloqueo de usuario.

Tabla 6.23: Resultado Nivel de Riesgo – Amenaza 6
Elaborado por: Investigador

Categoría	Elevación de privilegios
Descripción	Asumir la identidad de un usuario con privilegios.
Objetivo de la amenaza	El atacante escala privilegios y accede a funciones no autorizadas del sistema.
Nivel de riesgo	MEDIO

Técnicas de ataque	Modifica la URL o algún parámetro a una función con privilegios.
Contramedidas	- Implementar un módulo de autorizaciones de acuerdo a las transacciones que vaya a realizar - Asignación de privilegios de acceso al sistema - Incluir logs para pistas de auditoría. - Negar los accesos por defecto a las páginas del sistema

Tabla 6.24: Resultado Nivel de Riesgo – Amenaza 7
Elaborado por: Investigador

Etapa 5: Definir cómo responder a las amenazas

Una vez obtenido el valor del riesgo que compete a cada amenaza, con lo cual se determina la prioridad que tienen de acuerdo al nivel

- ✓ Inyección de comandos SQL: (Nivel ALTO)
  - Filtrar el contenido del campo de entrada de texto.
  - Utilizar un procedimiento almacenado que utilice parámetros para acceder a la base de datos.
  - Validar las entradas de texto, limitar la longitud del campo y verificar que cumpla con una serie de caracteres.
  - Utilizar API segura para minimizar el uso completo de intérpretes.

- ✓ Pérdida de autenticación y gestión de sesiones de usuario: (Nivel ALTO)
  - Fijar un tiempo de desconexión (expiración) de las sesiones.
  - Re-autenticar a los usuarios cuando realicen transacciones de alto nivel.
  - Utilizar controles de manejo de sesiones que ofrece la tecnología ASP.Net
  - Los datos sensibles se deben guardar en una sesión o servidor.

- ✓ Ataque de Cross-Site Scripting (XSS): (Nivel ALTO)
  - Utilizar Ajax para actualizar dinámicamente la página.
  - Validar la entrada de datos y que los datos cumplan con las reglas del negocio.
  - Utilizar políticas de seguridad de contenido

- ✓ Exposición de información sensible en mensajes de error de la aplicación: (Nivel ALTO)
  - Manejo de excepciones
  - Personalizar los mensajes de error

- ✓ Sobrecarga de peticiones al sistema: (Nivel ALTO)
  - Evitar cálculos complejos
  - Evitar búsquedas intensivas
  - Evitar que se carguen archivos de gran tamaño
  - En el caso de reiteración de contraseñas fallidas, se podría ejecutar un bloqueo de usuario

- ✓ Negar acciones realizadas por los usuarios: (Nivel MEDIO)
  - Implementar logs para evidencias las acciones del usuario antes de efectuar transacciones

- ✓ El atacante asume la identidad de un usuario con privilegios: (Nivel MEDIO)
  - Implementar un módulo de autorizaciones de acuerdo a las transacciones que vaya a realizar
  - Asignación de privilegios de acceso al sistema
  - Incluir logs para pistas de auditoría
  - Negar los accesos por defecto a las páginas del sistema

Herramienta Threat Modeling Tool para el modelado de amenazas

Se puede hacer uso de herramientas para el modelado de amenazas como por ejemplo Threat Modeling Tool (TMT), que permite construir una representación gráfica del modelado mediante el uso de DFD (Diagrama de Flujo de Datos).

En este caso se construye un diagrama sobre la operatividad del aplicativo Chequera Virtual.

El aplicativo Chequera Virtual El cual está desarrollado con una arquitectura de 3 capas: Navegador Web, el aplicativo que se encuentra expuesto en internet y la base de datos.

Los servidores con el que cuenta el aplicativo para su funcionalidad es: un Servidor Web Windows2003 Server y el servidor de base de datos SQL Server 2012.

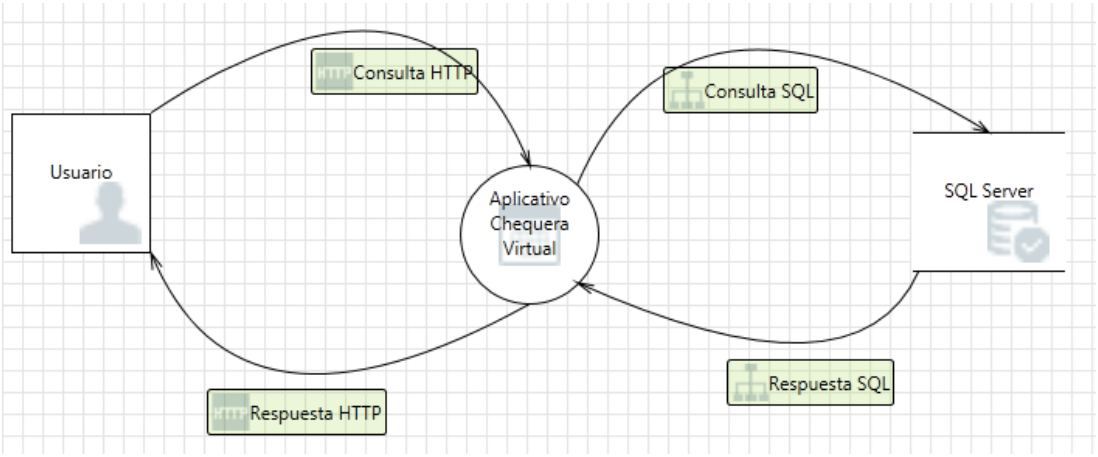


Figura 6.8 Diagrama del aplicativo Chequera Virtual, construido en la herramienta TMT
Elaborado por: Investigador

El resultado de análisis de riesgo generado con la herramienta TMT luego del diseño, es el siguiente:

Threat List

ID	Diagram	Changed By	Last Modified	State	Title	Category	Description	Justification	Interaction	Priority
0	Diagram 1		Generated	Not Started	Spoofing the U...	Spoofing	Usuario may b...		Consulta HTTP	High
1	Diagram 1		Generated	Not Started	Cross Site Scri...	Tampering	The web server...		Consulta HTTP	High
2	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Aplicativo Che...		Consulta HTTP	High
3	Diagram 1		Generated	Not Started	Spoofing of De...	Spoofing	SQL Server ma...		Consulta SQL	High
4	Diagram 1		Generated	Not Started	Potential SQL I...	Tampering	SQL injection i...		Consulta SQL	High
5	Diagram 1		Generated	Not Started	Potential Exces...	Denial Of Servi...	Does Aplicativ...		Consulta SQL	High
6	Diagram 1		Generated	Not Started	Spoofing of So...	Spoofing	SQL Server ma...		Respuesta SQL	High
7	Diagram 1		Generated	Not Started	Cross Site Scri...	Tampering	The web server...		Respuesta SQL	High
8	Diagram 1		Generated	Not Started	Persistent Cros...	Tampering	The web server...		Respuesta SQL	High
9	Diagram 1		Generated	Not Started	Weak Access C...	Information Di...	Improper data...		Respuesta SQL	High

Figura 6.9 Resultado análisis de riesgo generado automática por la herramienta TMT
Elaborado por: Investigador

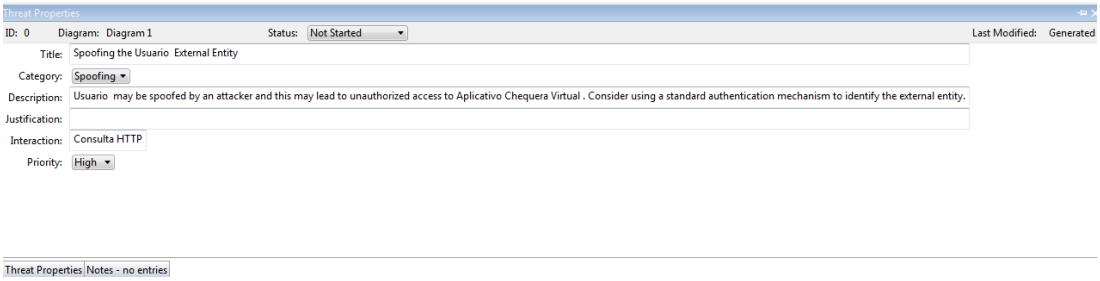


Figura 6.10 Resultado análisis de riesgo (detalle) generado automáticamente por la herramientaTMT
Elaborado por: Investigador

Se puede también obtener un reporte en formato HTML que la herramienta proporciona, se muestra de la siguiente manera:

Threat Modeling Report

Created on 9/11/2018 9:25:52 PM

Threat Model Name: Modelado de Amenzas Chequera Virtual

Owner: Ruth Aleaga

Reviewer:

Contributors:

Description:

Assumptions:

External Dependencies:

Threat Model Summary:

Not Started	10
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	10
Total Migrated	0

Diagram: Diagram 1

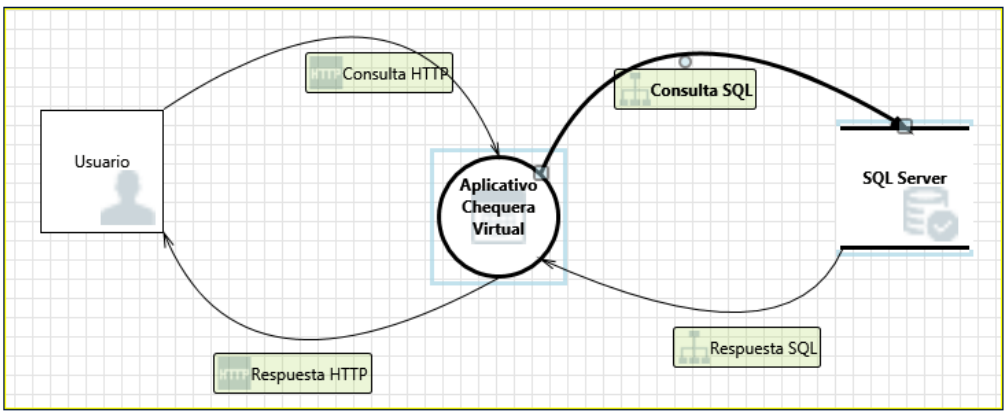
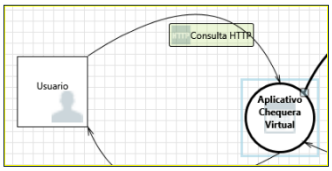


Diagram 1 Diagram Summary:

Not Started	10
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	10
Total Migrated	0

Interaction: Consulta HTTP



1. Spoofing the Usuario External Entity [State: Not Started] [Priority: High]

Category: Spoofing
Description: Usuario may be spoofed by an attacker and this may lead to unauthorized access to Aplicativo Chequera Virtual . Consider using a standard authentication mechanism to identify the external entity.
Justification: <no mitigation provided>

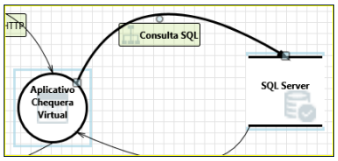
2. Cross Site Scripting [State: Not Started] [Priority: High]

Category: Tampering
Description: The web server 'Aplicativo Chequera Virtual ' could be a subject to a cross-site scripting attack because it does not sanitize untrusted input.
Justification: <no mitigation provided>

3. Elevation Using Impersonation [State: Not Started] [Priority: High]

Category: Elevation Of Privilege
Description: Aplicativo Chequera Virtual may be able to impersonate the context of Usuario in order to gain additional privilege.
Justification: <no mitigation provided>

Interaction: Consulta SQL



4. Spoofing of Destination Data Store SQL Server [State: Not Started] [Priority: High]

Category: Spoofing
Description: SQL Server may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of SQL Server . Consider using a standard authentication mechanism to identify the destination data store.
Justification: <no mitigation provided>

5. Potential SQL Injection Vulnerability for SQL Server [State: Not Started] [Priority: High]

Category: Tampering
Description: SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. Any procedure that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. Even parameterized data can be manipulated by a skilled and determined attacker.
Justification: <no mitigation provided>

6. Potential Excessive Resource Consumption for Aplicativo Chequera Virtual or SQL Server [State: Not Started] [Priority: High]

Category: Denial Of Service
Description: Does Aplicativo Chequera Virtual or SQL Server take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.
Justification: <no mitigation provided>

Interaction: Respuesta SQL

7. Spoofing of Source Data Store SQL Server [State: Not Started] [Priority: High]

Category: Spoofing
Description: SQL Server may be spoofed by an attacker and this may lead to incorrect data delivered to Aplicativo Chequera Virtual . Consider using a standard authentication mechanism to identify the source data store.
Justification: <no mitigation provided>

8. Cross Site Scripting [State: Not Started] [Priority: High]

Category: Tampering
Description: The web server 'Aplicativo Chequera Virtual ' could be a subject to a cross-site scripting attack because it does not sanitize untrusted input.
Justification: <no mitigation provided>

9. Persistent Cross Site Scripting [State: Not Started] [Priority: High]

Category: Tampering
Description: The web server 'Aplicativo Chequera Virtual ' could be a subject to a persistent cross-site scripting attack because it does not sanitize data store 'SQL Server ' inputs and output.
Justification: <no mitigation provided>

10. Weak Access Control for a Resource [State: Not Started] [Priority: High]

Category: Information Disclosure
Description: Improper data protection of SQL Server can allow an attacker to read information not intended for disclosure. Review authorization settings.
Justification: <no mitigation provided>

Figura 6.11 Informe en formato HTML, generado desde la herramienta TMT
Elaborado por: Investigador

El resultado de la herramienta arroja de acuerdo a interacciones:

Consulta HTTP: Aquí se identifica las categorías:

- **Spoofing Identity (Suplantación de identidad):** El usuario puede ser engañado por un atacante y esto puede llevar a un acceso no autorizado al Aplicativo Chequera Virtual. Considere usar un mecanismo de autenticación estándar para identificar la entidad externa.
- **Tampering with Data (Manipulación de datos):** El servidor web Chequera Virtual' podría estar sujeto a un ataque de scripts entre sitios, ya que no elimina información no confiable

- **Elevation of Privilege (Elevación de privilegios):** El Aplicativo Chequera Virtual puede suplantar el contexto de usuarios para obtener privilegios adicionales.

Consulta SQL: Aquí se identifica las categorías:

- **Spoofing Identity (Suplantación de identidad):** SQL Server puede ser falsificado por un atacante y esto puede provocar que los datos se escriban en el destino del atacante en lugar de en SQL Server. Considere utilizar un mecanismo de autenticación estándar para identificar el almacén de datos de destino.
- **Tampering with Data (Manipulación de datos):** La inyección SQL es un ataque en el que el código malicioso se inserta en cadenas que luego se pasan a una instancia de SQL Server para su análisis y ejecución. Cualquier procedimiento que construya sentencias de SQL debe revisarse para detectar vulnerabilidades de inyección, ya que SQL Server ejecutará todas las consultas sintácticamente válidas que reciba. Incluso los datos parametrizados pueden ser manipulados por un atacante experto y determinado.
- **Denial of Service (Denegación de servicio):** ¿Aplicativo Chequera Virtual o SQL Server toma pasos explícitos para controlar el consumo de recursos? Los ataques de consumo de recursos pueden ser difíciles de tratar, y hay veces que tiene sentido dejar que el sistema operativo haga el trabajo. Tener cuidado de que las solicitudes de recursos no se bloqueen y de que se agoten.

Respuesta SQL: Aquí se identifica las categorías:

- **Spoofing Identity (Suplantación de identidad):** SQL Server puede ser falsificado por un atacante y esto puede llevar a datos incorrectos entregados al Aplicativo Chequera Virtual. Considere usar un mecanismo de autenticación estándar para identificar el almacén de datos de origen.
- **Tampering with Data (Manipulación de datos):** El servidor web Chequera Virtual' podría estar sujeto a un ataque de scripts entre sitios porque no desinfecta las entradas no confiables.
- **Tampering with Data (Manipulación de datos):** El servidor web Chequera Virtual' podría estar sujeto a un ataque persistente de secuencias de comandos

entre sitios, ya que no desinfecta las entradas y la salida del servidor de datos 'SQL Server'.

- **Information Disclosure (Revelación de información):** La protección de datos inadecuada de SQL Server puede permitir que un atacante lea información que no está destinada a ser revelada. Revise los ajustes de autorización.

Tercera Fase: Implementación

En esta fase de implementación se aplica los siguientes elementos de SDL:

Estándares de codificación.- Los estándares ayudan a los programadores a minimizar los errores y que se produzcan vulnerabilidades. Entre las normas de codificación tenemos:

Indentación: Consiste en dejar 8 espacios a la izquierda. Ej:

```
//Lista de cuentas del cliente
foreach(Cuenta itemCuenta in listaCuentaCaptacion){
       if(itemCuenta.ProductoCuenta.AceptaRetiro){
              unaClaveAccesoTransferencia =
       ClaveAccesoTransferencia.GetRegistroCta(itemCuenta.Secuencial);
              if(unaClaveAccesoTransferencia != null){
                     unaClaveAccesoTransferencia.FechaUltimoIngreso =
DateTime.Now;

                     unaClaveAccesoTransferencia.Guardar();
              }
              break;
       }
}
```

Nombrado de funciones: Deben ser escritas en minúsculas y separadas con guión bajo.

```
    return Sistema.ValorACadena_INTERNA(unValor, enDolares, " ", false);
```

Para los enumerados, se debe escribir con mayúsculas y separados por guión bajo

```
public enum TipoCuentaTransferencia {AHORROS, CORRIENTE, TARJETA_CREDITO,
OTROS};
```

Comentario en el código: El comentario es una breve descripción de los realiza el código escrito.

```
//verifica si el socio tiene aceptado el contrato de oscus online
AceptacionContratoOscusOnline unaAceptacion =
AceptacionContratoOscusOnline.GetRegistro(usuarioWeb);
if(unaAceptacion == null)
      this.Response.Redirect("wbfContratoOscusOnline.aspx");
```

Espacios dentro del código: Facilita la lectura del código al incluir espacios entre elementos

```
//Obtiene la lista de bancos
ArrayList listaBancos = Banco.GetRegistros(true);


foreach (Banco unBanco in listaBancos){
      this.ddlIFIS.Items.Add(new ListItem(unBanco.Nombre,
      unBanco.Secuencial.ToString()));
}
this.ddlIFIS.SelectedValue = "44";
```

Declaraciones: Deben realizarse las declaraciones en líneas separadas, de tal manera que se le pueda agregar un comentario a cada declaración

```
      private DateTime fecha; //Fecha de transferencia
      private string comentario; //Comentario de transferencia
      private string cuentaReceptora; //Cuenta que recibe transferencia
```

Líneas largas: Fraccionar las líneas después de una coma o un operador

```
      valorAUtilizarLinea = (valorTotalMovimiento - this.Disponible)
      / valorComision;
```

Declaración de clases: Sin separación entre en nombre del método, el paréntesis y los parámetros del método. La llave de apertura del método debe ir en la misma línea y la llave de cierre debe con la misma indentación.

```
public bool ValidaClaveUsuarios(int secuencialCuenta, string
unaClaveUsuarioIngreso, string unaClaveUsuarioAutoriza)
{
      //Codigo
}
```

Manejo de sentencias:

- **Simples:** Deben ir en una sola línea

```
int id = MovimientoCuentaOnline.GetIdentificador(unMovimiento);
this.txtSecuencial.Text = id.ToString();
```

- **Compuestas:** Van entre llaves (así contenga una sola sentencia), de este tipo son: *if, switch, for, foreach, while*.

```
//Controles de saldos
if(unaclaveacceso.MontoMaximo < Decimal.Parse(txtMontoGiro.Text)){
   throw new Exception("No tiene monto de transferencia revisar...");
}else{
   if(cuenta.MenosDelMinimo(Decimal.Parse(txtMontoGiro.Text))){
         throw new Exception("Valor a Transferir Deja a la Cuenta sin
         el Minimo...");
   }
}
```

Identificación de Vulnerabilidades

Se empleará herramientas de análisis de código sobre el aplicativo Chequera Virtual para identificar problemas de seguridad en el código, vulnerabilidades en la aplicación y cumplimiento de buenas prácticas.

Herramienta FxCop: Es una herramienta gratuita de análisis de código estático, la cual se encarga de verificar que se cumplan reglas y buenas prácticas de programación. El análisis se realiza sobre librerías (DLL) ya compiladas.

En este caso se tiene la DLL de OscusOnline compilada en .net, lenguaje de programación C#. Esta DLL contiene varios métodos que permiten el funcionamiento del aplicativo Chequera Virtual.

A continuación el listado de vulnerabilidades como resultado del escaneo con FxCop sobre OscusOnline.dll:

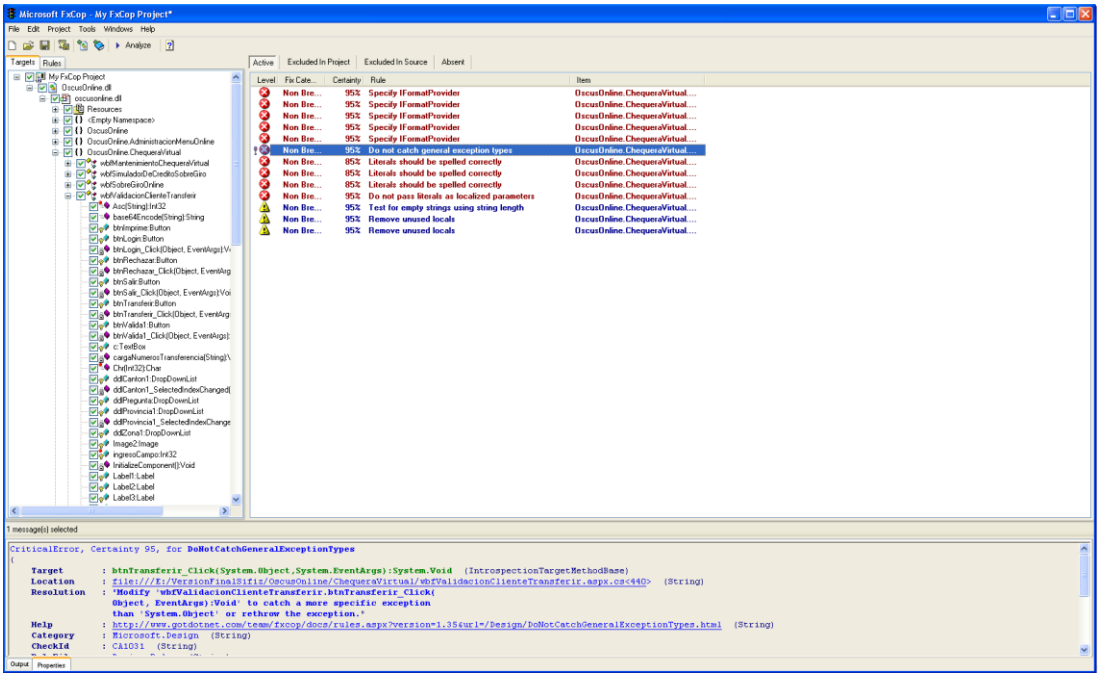


Figura 6.12 Errores detectados en método Transferir
Elaborado por: Investigador

Los errores detectados se encuentran identificados con rojo y también se han detectados warning. En cada uno de ellos aparece un detalle del error y una posible solución para contra restar el error. Entre los errores detectados se detallan:

1. El primer error detectado trata de que no se está manejando las excepciones de forma adecuada, pero ejemplo, el mensaje de error que va dentro del *catch* no se está desplegando.
2. El segundo error hace énfasis en que los mensajes que se visualizan en pantalla no deben ser asignados directamente en el control, si no que debería existir una tabla de parametrización de mensajes de donde se puede obtener dicha información.
3. El tercer error detectado es sobre las conversiones de tipo de datos, en este caso se está convirtiendo de tipo entero a string un dato, enviando un solo parámetro; lo que se sugiere en la herramienta es que se envíe el parámetro con un formato.

Todos los errores que arroja la herramienta se trata de los mismos casos que están en varias páginas, eventos, métodos. Por tanto según se va analizando uno a uno se puede ir solventando.

Error 1: El error fue detectado en el evento click del botón Transferir (btnTransferir_Click). En la pestaña *Problemas* muestra una resolución al error detectado e indica el empleo de excepciones.

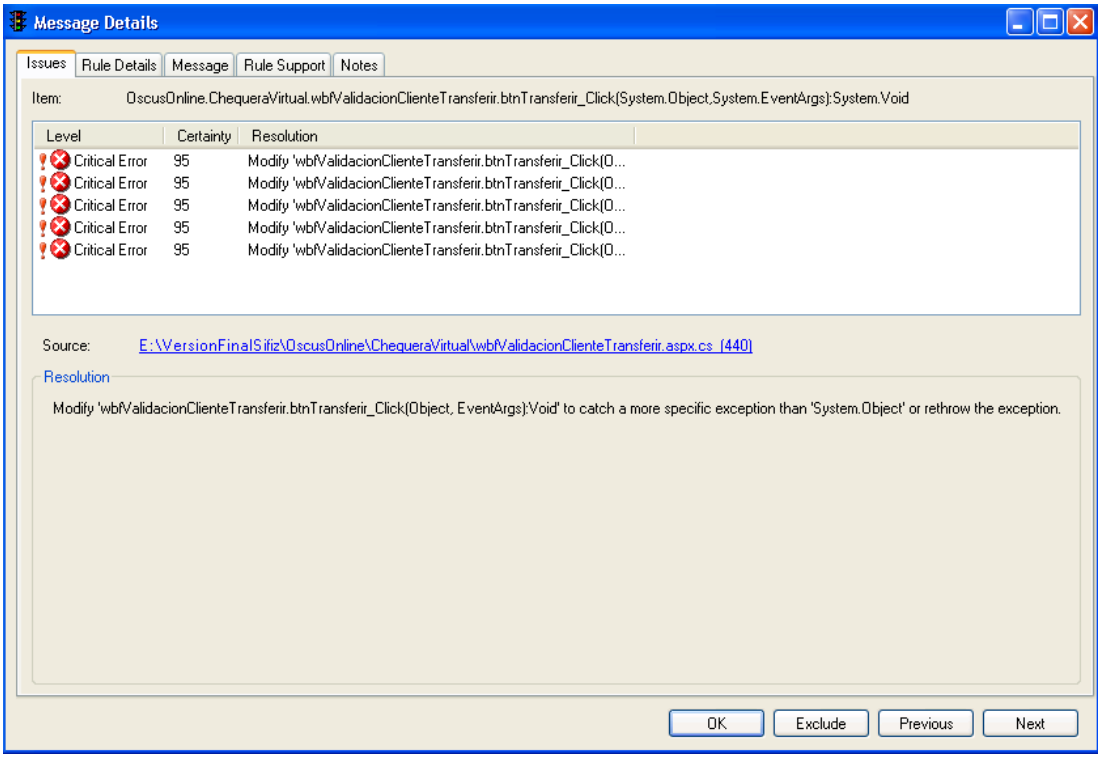


Figura 6.13 Error 1 detectado
Elaborado por: Investigador

En la pestaña de detalles de regla se especifica una información adicional en donde hace énfasis a que se debe capturar el error con el manejo de excepciones.
En los detalles de la regla también categoriza al error y da una información adicional de cómo se podría mitigar el error.

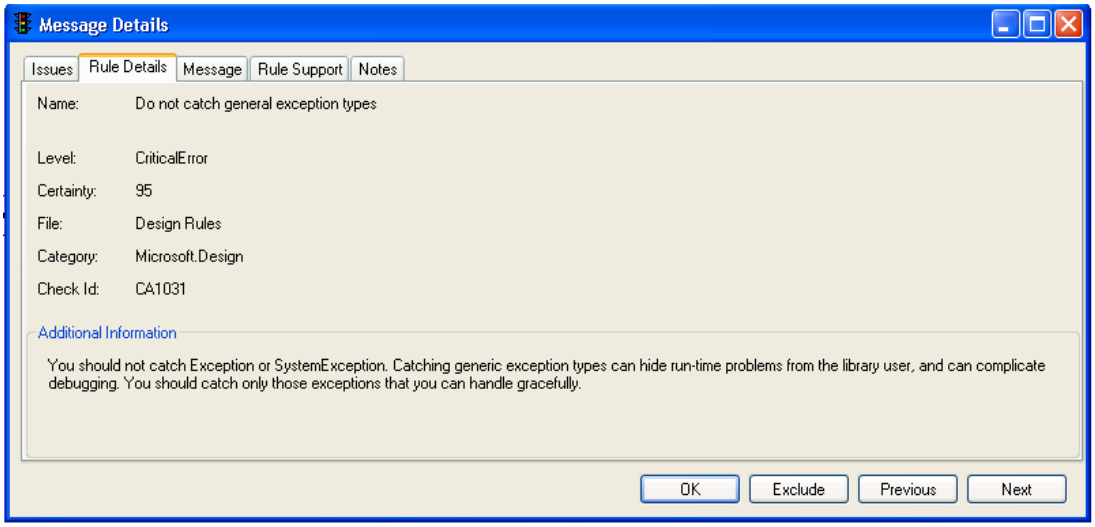


Figura 6.14 Detalle de Error 1
Elaborado por: Investigador

Solución:

Para solventar este error se utiliza el manejo de excepciones, desplegando el error generado en la sección del ***catch***

```
try
{
        //codigo
}
catch (Exception error)
{
        throw new Exception(error.Message);
}
```

Error 2: Se ha detectado también en el evento click del botón Transferir (btnTransferir_Click) que los mensajes que se emiten pueden ser tomados de alguna tabla parametrizada con mensajes.

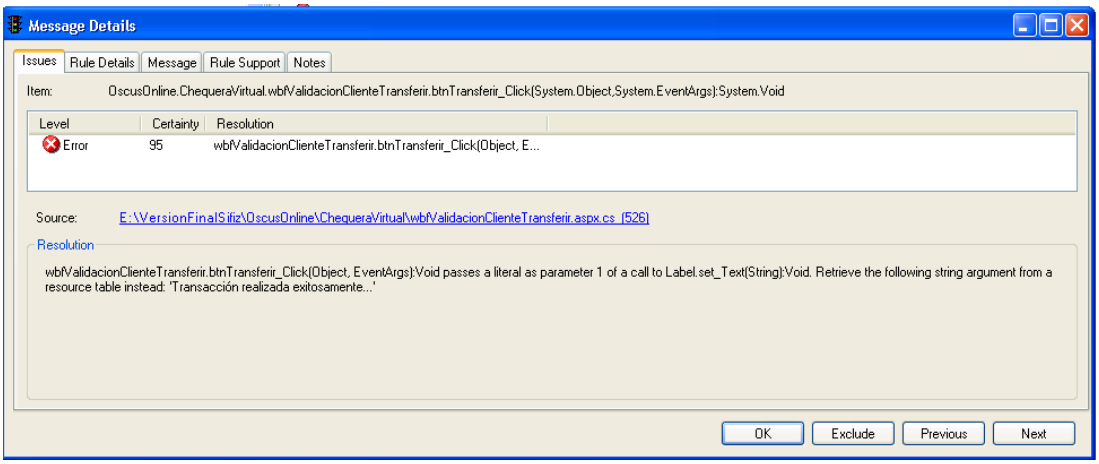


Figura 6.15 Error 2 detectado
Elaborado por: Investigador

Solución:

Para solventar este error ya no se asigna directamente el mensaje al control, si no se lee de una tabla creada y parametrizada con mensajes.

```
string parametro = "1";
this.lblMensaje.Text = GetRegistro(parametro);
```

Error 3: Error detectado en el evento click del botón Transferir (btnTransferir_Click). El detalle es acerca del manejo de conversiones de string a int (viceversa) con la utilización de *IFormatProvider*.

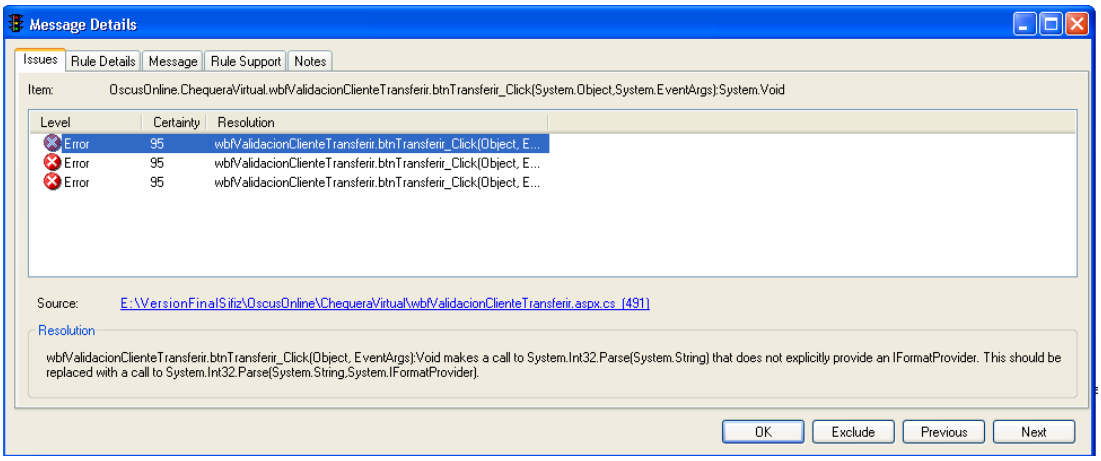


Figura 6.16 Error 3 detectado
Elaborado por: Investigador

Solución:

Para solventar este error se envía como parámetro adicional *CultureInfo.CurrentCulture* en la conversión de a *string*.

```
string numeroDocumento = "1873834";
int documento = Int32.Parse(numeroDocumento,
CultureInfo.CurrentCulture);
```

Herramienta BinScope: Es una herramienta disponible de forma gratuita de Microsoft, analiza archivos binarios (DLL) para detectar vulnerabilidades dentro de archivos binarios y comprobar que el código ha sido construido cumpliendo requisitos de seguridad de SDL.

Se envía a analizar la dll de OscusOnline mediante consola con los comandos que se encuentran a continuación en la imagen:

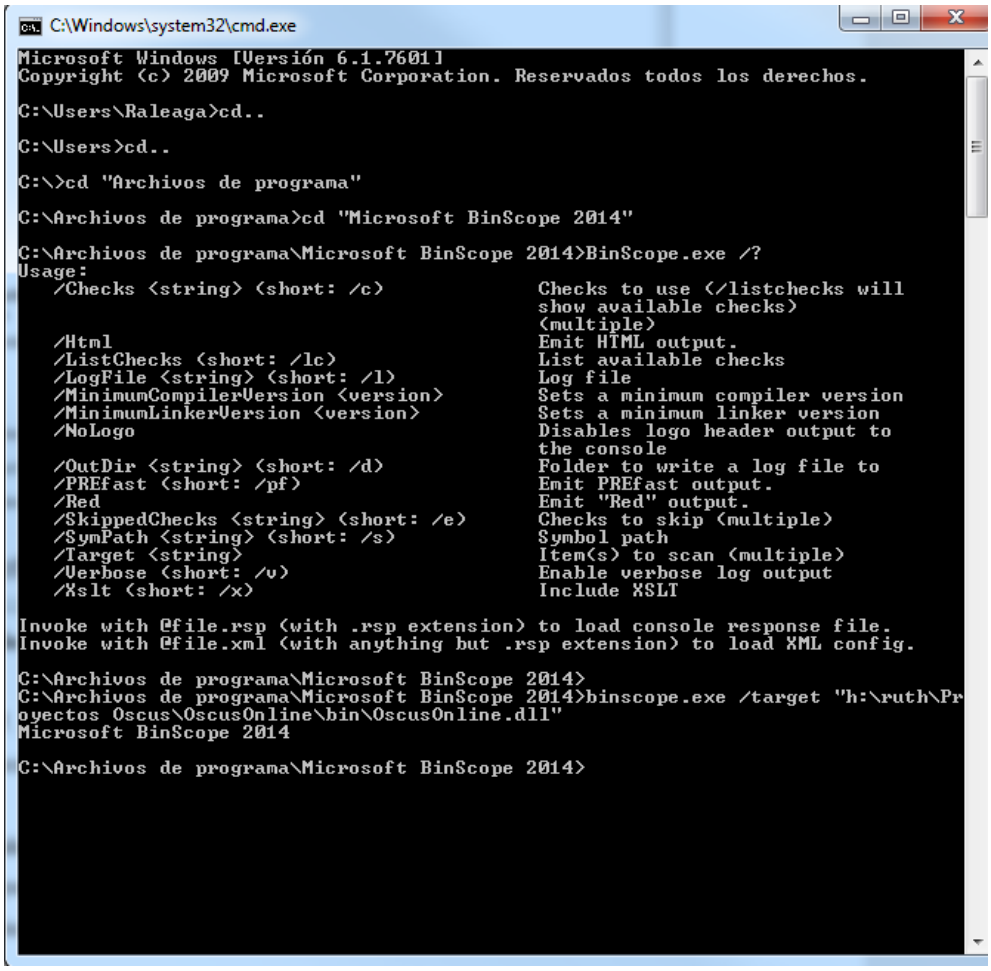


Figura 6.17 Escaneo vulnerabilidades con BinScope
Elaborado por: Investigador

Una vez ejecutado los comandos, se crea un archivo HTML en el cual muestra el informe de resultados de la revisión de las bibliotecas de vínculos dinámicos (DLL), con los errores identificados:

RESULTS FOR MICROSOFT BINSCOPE 2014 RUN ON MADESA01 AT 2018-10-02T00:11:05.9771005Z

Failed Checks

No failed checks.

Checks Executed:

- ATLVersionCheck
- ATLVulnCheck
- CompilerVersionCheck
- DBCheck
- DefaultGSCookieCheck
- ExecutableImportsCheck
- GSCheck
- GSFriendlyInitCheck
- GSFunctionSafeBuffersCheck
- HighEntropyVACheck
- NXCheck
- RSA32Check
- SafeSEHCheck
- SharedSectionCheck
- VB6Check
- WXCheck

All Scanned Items

- h:\ruth\Proyectos Oscus\OscusOnline\bin\OscusOnline.dll
  - path : h:\ruth\Proyectos Oscus\OscusOnline\bin\OscusOnline.dll
  - version : 1.0.0.0
  - versionString : File: h:\ruth\Proyectos Oscus\OscusOnline\bin\OscusOnline.dllInternalName: OscusOnline.dllOriginalFilename: OscusOnline.dllFileVersion: 1.0.0.0FileDescription: OscusOnlineProduct: OscusOnlineProductVersion: 1.0.0.0Debug: FalsePatched: FalsePreRelease: FalsePrivateBuild: FalseSpecialBuild: FalseLanguage: Independiente del idioma
  - length : 220160
  - key : C50DEBADF2A55A1366B8B1B78CA98D94F88960D9
  - description : h:\ruth\Proyectos Oscus\OscusOnline\bin\OscusOnline.dll
  - SymbolPath : h:\ruth\Proyectos Oscus\OscusOnline\bin\OscusOnline.pdb

Figura 6.18: Resultado con BinScope
Elaborado por: Investigador

Según el resultado de análisis con esta herramienta, no se han detectado vulnerabilidades dentro de los archivos binarios. En los siguientes escenarios:

- **ATLVersionCheck:** Verifica los encabezados de ATL que suelen ser para construir el binario sean bien conocidos.
- **ATLVulnCheck:** Detecta clases que implementan *iPersistStreamInit* que tienen entradas de mapas de propiedades potencialmente vulnerables.
- **GSCheck:** Verifica que el indicador del compilador / GS se utilizó para compilar todos los componentes del binario y muestra información detallada por objeto en el binario.
- **SafeSEHCheck:** Verifica que la imagen haya sido enlazada usando / SAFESEH. No usar / SAFESEH destruye la protección proporcionada por / GS.
- **CompilerVersionCheck:** Identifica imágenes que contienen módulos C o C ++ compilados con un compilador más antiguo que la versión requerida por el SDL.
- **DBCheck:** Comprueba si un binario ha optado por la función ASLR.
- **NXCheck:** Comprueba si un binario ha optado por la Prevención de ejecución de datos de hardware

Herramienta Code Analysis: Es una herramienta que se encuentra incluida dentro de Visual Studio. Funciona de forma similar a FxCop en cuanto al conjunto de reglas que se utilizan para la verificación, la diferencia es que esta herramienta es ejecutada de forma directa sobre el código fuente.

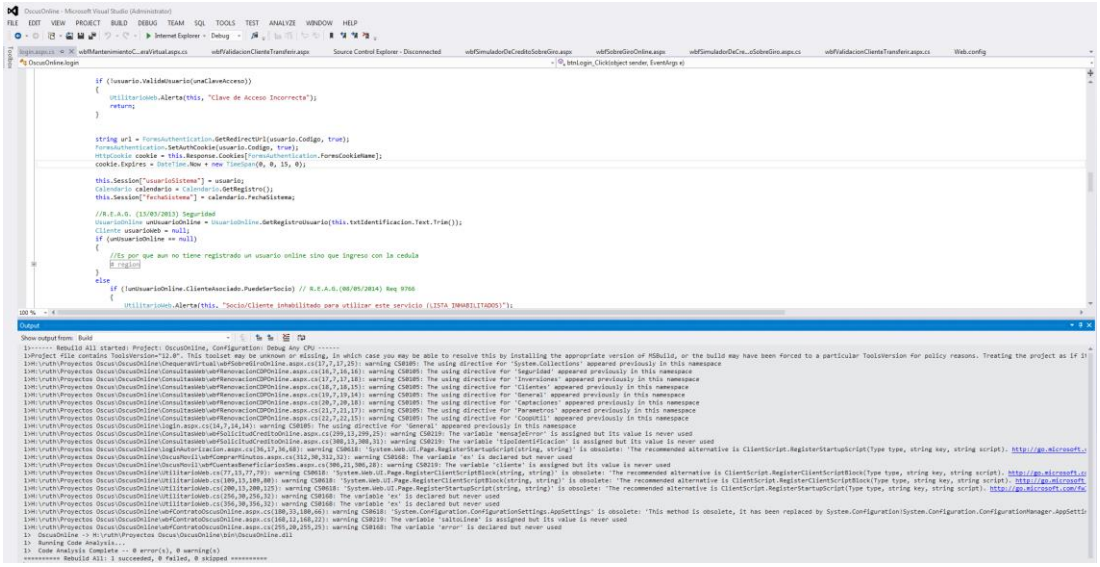


Figura 6.19 Análisis con Code Analysis
Elaborado por: Investigador

Como resultado del análisis se puede apreciar en la imagen que no existen errores, pero si algunas advertencias (warning). Se visualiza un detalle de la advertencia y la línea en donde se ha detectada la misma. A continuación la revisión de cada una de ellas:

Variable declarada y no utilizada

La advertencia indica que la variable ha sido declarada pero no es utilizada en ninguna parte del documento, a continuación la imagen:

```
string saltoLinea = "\n";

Usuario usuario = (Usuario) this.Session["usuarioSistema"];
Cliente usuarioWeb = (Cliente) this.Session["usuarioWeb"];
DateTime fechaSistema = (DateTime) this.Session["fechaSistema"];

string representantePrincipal = "_____";
```

Figura 6.20 Advertencia 1: Variable declarada y no utilizada
Elaborado por: Investigador

Solución:

Para solucionar esta advertencia se procede a eliminar la línea, una vez constatado que no se utiliza.

Variable excepción no utilizada

La advertencia indica que la variable *ex* ha sido declarada en el *catch* pero no es utilizada para lanzar el error capturado como se muestra en la imagen:

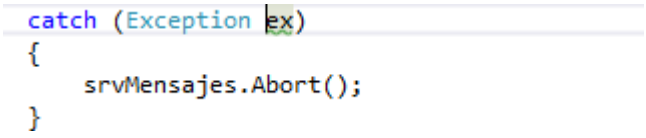


Figura 6.21 Advertencia 2: Variable excepción no utilizada
Elaborado por: Investigador

Solución:

Para solventar esta advertencia se incluye la línea *throw new Exception* con el mensaje descrito en la variable *ex*.

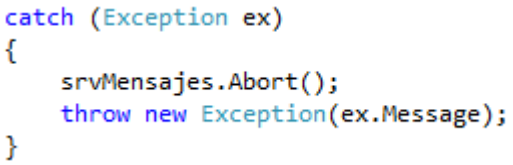


Figura 6.22 Solución Advertencia 2
Elaborado por: Investigador

Llamada obsoleta a funciones y lecturas de claves

En la imagen capturada muestra una advertencia que se encuentra incorrecta la llamada a la lectura de claves configuradas en el *appSettings* del archivo *web.config*.

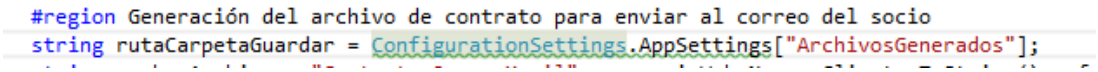


Figura 6.23 Advertencia 3: Llamada obsoleta a función
Elaborado por: Investigador

Solución:

Para dar solución a la advertencia se hace referencia con el siguiente código *System.Configuration.ConfigurationManager*:

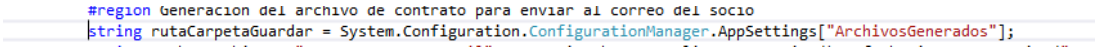


Figura 6.24 Solución Advertencia 3
Elaborado por: Investigador

De igual manera para registrar el script de inicio, se encuentra incorrecta la sintaxis de llamada:

```
this.RegisterStartupScript("Startup", scriptString);
Cliente usuarioWeb = (Cliente)this.Session["usuarioWeb"];
```

Figura 6.25 Advertencia 4
Elaborado por: Investigador

Solución:

Para dar solución a la advertencia se incluye las siguientes líneas de código:

```
Type cstype = this.GetType();
ClientScript.RegisterStartupScript(cstype, "Startup", scriptString);
```

Figura 6.26 Solución Advertencia 4
Elaborado por: Investigador

Declaración de librerías

En la siguiente advertencia hace referencia a las librerías declaradas que se encuentran duplicadas como se ve en la imagen:

```
using System.Web.UI;
using System.Web.UI.WebControls;

using Seguridad;
using Inversiones;
using Clientes;
using General;
using Captaciones;
using Parametros;
using CoopUtil;

using Seguridad;
using Inversiones;
using Clientes;
using General;
using Captaciones;
using Parametros;
using CoopUtil;
using System.Collections;
```

Figura 6.27 Advertencia 5: Declaración duplicada de librerías
Elaborado por: Investigador

Solución:

La solución a esta advertencia es eliminar las librerías que se encuentran duplicadas

```
using System.Web.UI;
using System.Web.UI.WebControls;

using Seguridad;
using Inversiones;
using Clientes;
using General;
using Captaciones;
using Parametros;
using CoopUtil;

using System.Collections;
```

Figura 6.28 Solución Advertencia 5
Elaborado por: Investigador

Para finalizar se ejecuta nuevamente el análisis con la herramienta Code Analysis y como resultado nos arroja que no se han encontrado vulnerabilidad dentro de la aplicación:

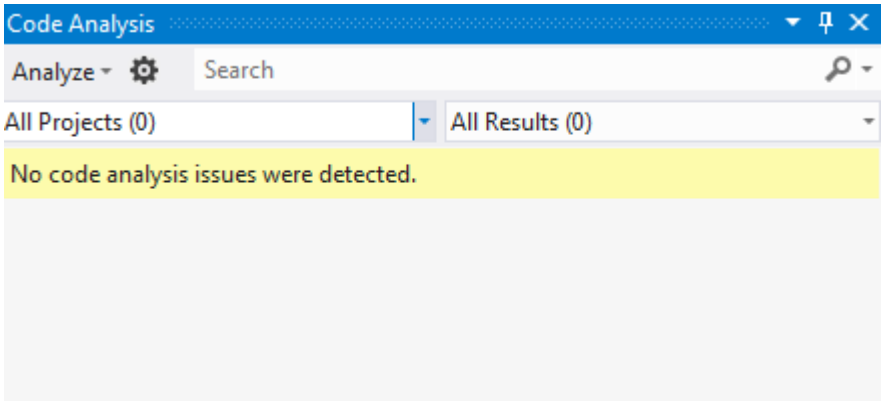


Figura 6.29 Resultado de análisis final Code Analysis
Elaborado por: Investigador

Cuarta Fase: Comprobación

Se realiza una revisión final en tiempo de ejecución del aplicativo Chequera Virtual. Pruebas introduciendo datos erróneos, de tal manera que se detecten fallos y poder corregirlos a tiempo.

Pruebas de exploración de vulnerabilidades (Fuzz testing)

A continuación se emplea un análisis dinámico sobre el aplicativo introduciendo datos erróneos para provocar errores. Entre ellos, se cita:

Datos erróneos ingresados en un campo de texto

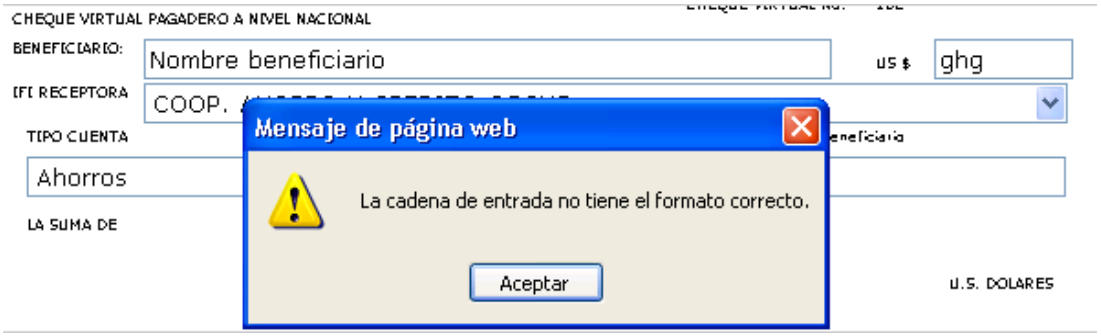


Figura 6.30 Datos erróneos ingresados en un campo de texto
Elaborado por: Investigador

Como se puede observar en la imagen el aplicativo no está controlando el tipo de dato ingresado en el campo US$ que requiere que sean numéricos y el mensaje no es personalizado. Para solventar esta vulnerabilidad se implementó una función *javascript* de manera que se pueda aplicar al campo de texto y se pueda controlar el ingreso de caracteres.

```
<script language="javascript">
            function solonumeros(letra)
            {
                  var tecla = window.event.keyCode;
                  if ((tecla < 48 || tecla >57))
                  {
                        window.event.keyCode=0;
                  }
                  return letra;
            }
</script>
```

Esta función *javascript* es aplicada al control, de la siguiente manera:

```
this.txtMonto.Attributes.Add("onkeypress","solonumeros('"+this.txtMo
nto.Text+"') ;");
```

Mensajes de error revela información de tablas de la base de datos

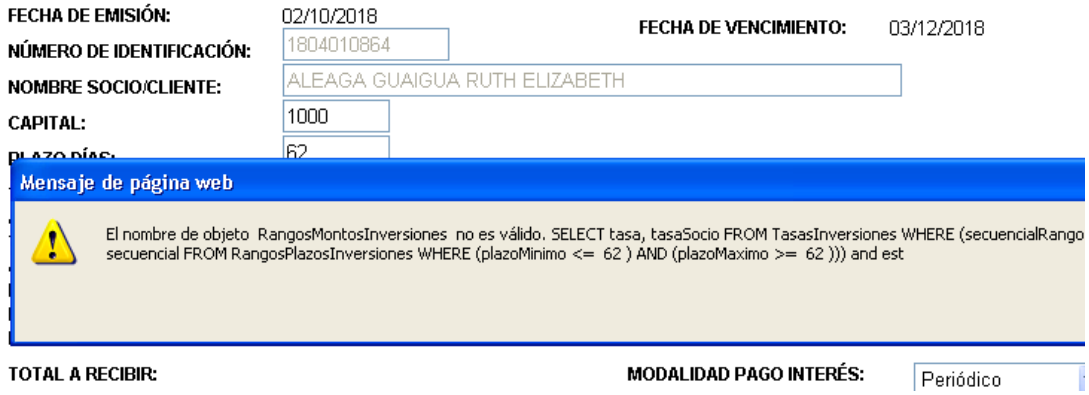


Figura 6.31 Mensajes de error revela información de tablas de la base de datos
Elaborado por: Investigador

El mensaje de error generado revela información de la base de datos y es una vulnerabilidad de riesgo alto ya que con esto el atacante conoce acerca de nombres de tablas y campos de la base. Estos mensajes de error deben ser personalizados, para lo cual se maneja el control de excepciones.

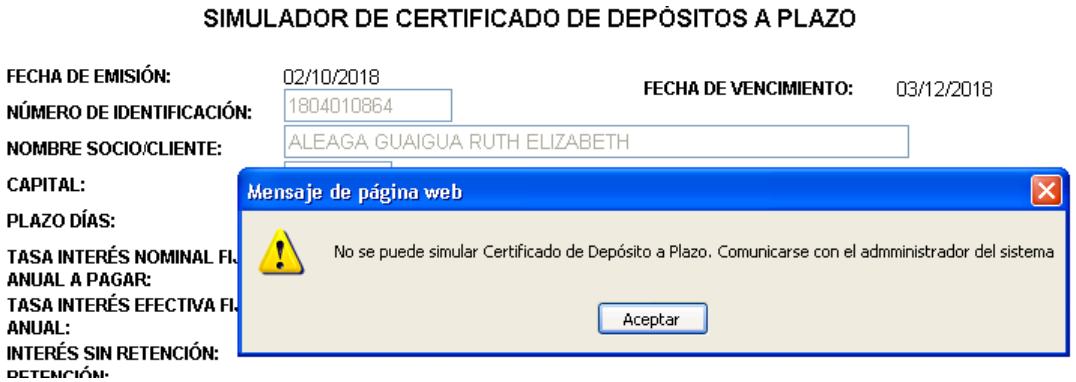


Figura 6.32 Mensaje de error personalizado
Elaborado por: Investigador

En la imagen se muestra personalizado el mensaje de error, incluyendo las siguientes líneas de código:

```
try
{
```

```
            //Codigo
        }
        catch (Exception error )
        {
            UtilitarioWeb.Alerta(this, "No se puede simular Certificado de
Depósito a Plazo. Comunicarse con el admministrador del sistema");
            }
```

De igual manera se debe controlar que no se visualicen errores que ocurren internamente en el servidor, también aplicando el control de excepciones.

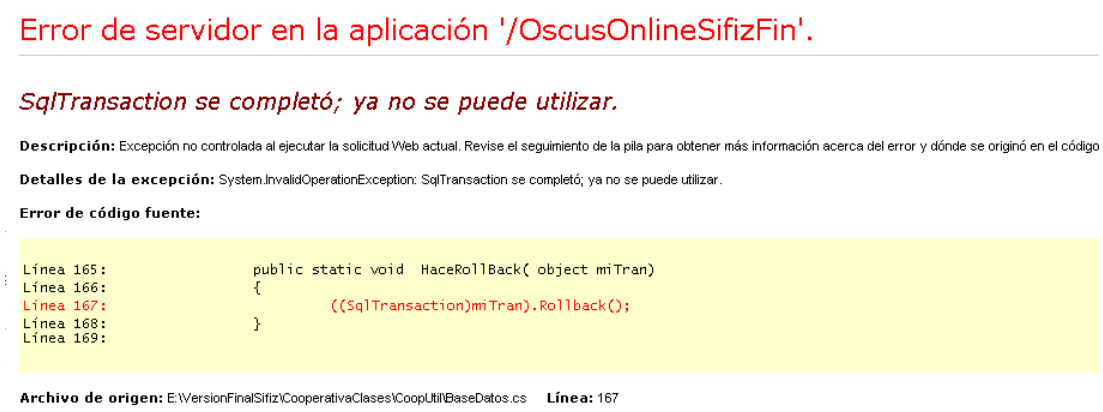


Figura 6.33 Mensaje de error despersonalizado
Elaborado por: Investigador

El error visualizado en la imagen se soluciona utilizando también el manejo de excepciones. El proceso debe estar dentro de la sección *try* y *catch*.

Quinta Fase: Lanzamiento

Para la elaboración del plan de respuesta a incidentes, se toma en cuenta los siguientes criterios:

- **El equipo de Respuesta a Incidentes:** En este caso las personas idóneas para cumplir este rol son:

Nombre	Cargo	Dirección	Teléfono
Juan Ortiz	Responsable de Seguridad Integral		
Marcelo Herrera	Oficial de		

	Seguridad Lógica		

Tabla 6.25: Equipo de Respuesta a Incidentes
Elaborado por: Investigador

Las personas descritas en la tabla 6.25 se encuentran capacitadas, cuentan con experiencia y poseen las herramientas necesarias para responder ante incidentes presentados, las cuales deberán estar disponibles 24/7.

- **Formulario de plan de respuesta a incidentes:** Este documento ayudará a tener organizada la información en caso de que ocurra una caída o fallo del sistema. Se debe hacer constar información detallada sobre el incidente.
  Documentar cada incidente servirá de apoyo para cuando se presenten futuros incidentes y se pueda ir almacenando en una bitácora (ver Anexo 2).

6.8 Conclusiones

- ✓ Durante todo el proceso que conlleva la aplicación de las fases de la metodología SDL demuestran que se debe pensar en todo momento acerca de la seguridad y privacidad del software.
- ✓ La aplicación de la metodología SDL en los proyectos de desarrollo ayudan significativamente a la mejora de la seguridad en lo que concierne a programación, basándose en una metodología ágil lo que significa que puede ser incremental y flexible.
- ✓ Los procesos que se llevan a cabo en cada fase aportan a obtener un producto de software confiable, por lo que la seguridad debe ser incluida desde la planificación inicial (la primera fase), de esta manera se consigue bases sólidas para no tener retrasos durante el cumplimiento del proceso que contempla la metodología SDL.
- ✓ La utilización de herramientas adecuadas para el escaneo del código fuente permiten al programador dar una solución oportuna ante las vulnerabilidades encontradas. De esta manera se reduce el nivel de riesgo que puede existir hacia el sistema.
- ✓ El modelado de amenazas dentro del proceso SDL ayuda a identificar los activos críticos con que cuenta la institución y mitigar los posibles problemas de seguridad.

6.9 Recomendaciones

- ✓ Se recomienda la aplicación de buenas prácticas de programación ya que son principios fundamentales que todo desarrollador de software de aplicar.
- ✓ Se recomienda que el equipo de desarrollo reciba capacitación al menos una vez al año acerca de seguridades en software y de esta manera se tenga aportes en conjunto con toda el área de desarrollo.
- ✓ Se recomienda seleccionar herramientas apropiadas para el análisis de riesgos ya que no todas se adaptan a los proyectos que se realizan.
- ✓ Se recomienda tener una documentación actualizada de cada proyecto plasmando los procesos que se han realizado en cada una de las fases de SDL.
- ✓ Se recomienda el registro de incidentes haciendo uso del formulario del plan de respuesta a incidentes (ver anexo 2), el cual se encuentra en un formato legible y fácil de llenar, el cual servirá de guía para futuras aplicaciones que se desarrollen.
- ✓ Se recomienda el uso de la herramienta TMT (Threat Modeling Tool) para el modelado de amenazas, de tal manera que se pueda generar un análisis de forma automática.

Bibliografía

OPENSAMM Software Assurance Maturity Model. (2009). Obtenido de http://www.opensamm.org

Implementación simplificada del proceso SDL de Microsoft. (2 de Febrero de 2010). Obtenido de https://www.microsoft.com/es-es/download/details.aspx?id=12379

Metodologías para desarrollar software seguro. (2013). Obtenido de http://recibe.cucei.udg.mx/revista/es/vol2-no3/computacion05.html

(2015). *REGLAMENTO DE LA GESTIÓN DE TECNOLOGÍA DE LA INFORMACIÓN Y COMUNICACIÓN INFORMATICA.*

Aguilera Días, V. (09 de 05 de 2012). *Desarrollo de software seguro: una visión con OpenSAMM.* Obtenido de https://www.isecauditors.com/sites/default/files/files/IDiA_Desarrollo_de_software_seguro_OpenSAMM.pdf

Alvarado, F. (2012). *Ingeniería del Software*. Obtenido de http://brfranciscoosunaiuty.blogspot.com/2012/07/proceso-de-desarrollo-de-software.html

Ascencio Mendoza, M., & Moreno Patiño, P. (2011). *Desarrollo de una Propuesta Metodológica para Determinar la Seguridad en una Aplicación Web.* Obtenido de http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/2511/0058A811.pdf

Barba Olivares, G. E. (2017). *Modelado de Amenazas.* Obtenido de http://polux.unipiloto.edu.co:8080/00004067.pdf

Brito Abundis, C. (3 de 12 de 2013). *Metodologías para desarrollar software seguro.* Obtenido de http://recibe.cucei.udg.mx/revista/es/vol2-no3/pdf/computacion05.pdf

Casares Charles, J. P. (Julio de 1999). *AMBIENTE PARA LA INSTRUCCIÓN VISUAL DE ALGORITMOS.* Obtenido de http://usablehack.com/amiva/AMIVA.pdf

Castellaro, M., Romaniz, S., Ramos, J., & Pessolani, P. (2009). *Hacia la Ingeniería de Software Seguro.* Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/21332/Documento_completo.pdf

Chang, V., Walters, R. J., & Wills, G. (2015). *Delivery and adoption of Cloud Computing Services in Contemporary Organizations.*

Cuenca Diaz, C. R. (s.f.). *Desarrollo Seguro: Principios y Buenas Prácticas*. Obtenido de https://www.owasp.org/images/9/93/Desarrollo_Seguro_Principios_y_Buenas_Pr%C3%A1cticas..pdf

d. (s.f.). Obtenido de https://www.isecauditors.com/sites/default/files/files/IDiA_Desarrollo_de_software_seguro_OpenSAMM.pdf

Geer, D. (2010). *Are Companies Actually Using Secure Development Life Cycles?*

Howard, M., & Lipner, S. (2006). *THE SECURITY DEVELOPMENT LIFECYCLE: SDL, a process for developing demonstrably more secure software.*

Humphrey, W. S. (Noviembre de 2000). *The Team Software Process (TSP)*. Obtenido de https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5287

LAGOS SANTELICES, A. I. (Junio de 2012). *MEJORA SISTEMÁTICA DEL PROCESO DE DESARROLLO DE SOFTWARE DE LA DIVISIÓN DE AUTOSERVICIO DE DTS.* Obtenido de http://repositorio.uchile.cl/bitstream/handle/2250/110974/cf-lagos_as.pdf

Lipner, S. B. (2005). *Building More Secure Commercial Software: The Trustworthy Computing Security Development Lifecycle.* Obtenido de http://cs.emis.de/LNI/Proceedings/Proceedings67/GI-Proceedings.67-3.pdf

Mendoza T., J. C. (Noviembre de 2008). *Demostración de cifrado simétrico y asimétrico*. Obtenido de http://dspace.ups.edu.ec/handle/123456789/8185

Noopur, D. (2005). *Secure Software Development Life Cycle Processes: A Technology Scouting Report.* Obtenido de http://www.dtic.mil/dtic/tr/fulltext/u2/a447047.pdf

Robles Gomez, E. (22 de 07 de 2011). *Ciclo de Vida de Desarrollo de Software Seguro en Metodologías Ágiles.* Obtenido de https://cimat.repositorioinstitucional.mx/jspui/bitstream/1008/411/1/ZACTE14.pdf

Roche Saldarriaga, J. P., & Suarez Arias, J. M. (13 de Febrero de 2009). *ANÁLISIS, DISEÑO, E IMPLEMENTACIÓN DE UN SOFTWARE, PARA LA ADMINISTRACIÓN DE PROYECTOS DE GRADO EN EL PROGRAMA DE SISTEMAS, APLICANDO UNA METODOLOGÍA ÁGIL.* Obtenido de http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/1316/0057565R673.pdf

Solano Campos, J. (2015). *Implementación de la metodología Six Sigma SDLC para la mejora de la calidad del proceso de desarrollo web.* Obtenido de http://www.repositorio.una.ac.cr/bitstream/handle/11056/10550/Documento_Proyecto Final.pdf

Terrence R., S., Jeffrey F., S., & Daniel R., S. (18 de Noviembre de 1997). *Cryptography system and method for providing cryptographic services for a computer application*. Obtenido de https://www.google.com/patents/US5689565

Anexos

ANEXO 1

ENCUESTA PARA PROGRAMADORES

Objetivo: Determinar el nivel de seguridad que existe en el código desarrollado dentro de la Cooperativa.

1. ¿Conoce sobre metodologías de desarrollo de software seguro?

   SI ☐ NO ☐

2. ¿Existe alguna metodología de seguridad que aplique en el desarrollo de sistemas dentro de la Cooperativa?

   SI ☐ NO ☐

3. ¿Aplica alguna técnica de encriptación a nivel de programación? En caso de aplicar, indique cual.

   SI ☐ NO ☐

   Técnica de encriptación: …….

4. ¿Aplica herramientas de análisis de código que valide la seguridad?

   SI ☐ NO ☐

5. ¿Se realizan auditorías al código desarrollado en la Cooperativa?

   SI ☐ NO ☐

6. ¿Se han detectado amenazas hacia el código por parte de intrusos?

SI ☐ NO ☐

7. ¿Considera que el software desarrollado es vulnerable a algún tipo de amenaza?

SI ☐ NO ☐

8. ¿Se preocupa de la funcionalidad del sistema antes que de la seguridad?

SI ☐ NO ☐

9. ¿Considera indispensable la aplicación de alguna metodología de seguridad en el desarrollo de software?

SI ☐ NO ☐

10. ¿Se ve afectado el negocio por la carencia de seguridad en los sistemas financieros que utilizan en cada institución?

SI ☐ NO ☐

ANEXO 2

OSCUS		COOPERATIVA DE AHORRO Y CREDITO OSCUS LTDA.	
		DESARROLLO DE TIC	
		FORMULARIO PLAN DE RESPUESTA A INCIDENTES	

DETALLE DEL INCIDENTE:					
Fecha/hora que sucedió el incidente	dd/mm/aaaa hh:mm:ss				
Fecha/hora que se reportó el incidente	dd/mm/aaaa hh:mm:ss				
Lugar del incidente	Nombre de oficina operativa				
Aplicativo/Sistema afectado	Nombre del aplicativo(sistema financiero, chequera virtual, etc.)				
Persona que reporta el incidente	Nombre de la persona / cargo				
Daños ocasionados por el incidente	Que afectaciones hubo (local o de toda la cooperativa)				
Se detecta vulnerabilidad?	SI ☐	NO ☐	Cual?		
Se detecta que ocasionó el incidente?	SI ☐	NO ☐	Cual?		
Se identifica al responsable del incidente?	SI ☐	NO ☐	Cual?		
Estado del incidente	Sucediendo ☐	Sucedió ☐	Sucede nuevamente ☐		

DETALLE DE SOLUCIÓN DEL INCIDENTE:	
Fecha/hora de investigación del incidente	dd/mm/aaaa hh:mm:ss
Nombre del responsable de investigación del incidente	Nombre completo
Fecha/hora de finalización del incidente	dd/mm/aaaa hh:mm:ss
Fecha/hora de finalización del impacto del incidente	dd/mm/aaaa hh:mm:ss
Descripción de las acciones tomadas para resolver el incidente	
Lista de evidencias obtenidas	
Acciones pendientes para resolver el incidente	
Conclusiones	
Recomendaciones	
Lecciones aprendidas del incidente	

DETALLE DE LA PERSONA MIEMBRO DEL EQUIPO DE RESUESTA A INCIDENTES:	
Nombre	
Departamento	
Teléfono	
Dirección	
Teléfono	
Correo electrónico	
Fecha	mm/dd/aaaa
Firma	