Ciclo de vida de desarrollo ágil de software seguro

Miguel Hernández Bejarano*
mhernandezb@libertadores.edu.co

Luis Eduardo Baquero Rey*
lebaqueror@libertadores.edu.co

Colección INVESTIGACIÓN

---

* Grupo de Investigación en Ingeniería Aplicada (GUIAS).

Catalogación en la Publicación Fundación Universitaria Los Libertadores

Hernández Bejarno, Miguel
Ciclo de vida de desarrollo ágil de software seguro Miguel Hernández Bejarano / Luis Eduardo Baquero Rey, – Bogotá: Fundación Universitaria Los Libertadores, 2020.
108 páginas, ilustraciones, gráficas; 26 cm (Colección Investigación)

ISBN: 978-958-5478-41-1 (impreso) | ISBN: 978-958-5478-44-2 (digital)

1. Seguridad en el software – 2. Metodologías ágiles – 3. Desarrollo de software seguro – 4. Seguridad informática. I Hernández Bejarano, Miguel, Autor II. Fundación Universitaria Los Libertadores.

SCDD 005.8 B222c –dc23 **FULL BIBLIOTECA**

Primera Edición: Bogotá, diciembre de 2020
©Fundación Universitaria Los Libertadores.

Cra. 16 No. 63A-68/ Tel. 2544750.
www.ulibertadores.edu.co

Juan Manuel Linares Venegas
Presidente del Claustro

Ángela María Merchán Basabe
Rectora

©*Miguel Hernández Bejarano*
©*Luis Eduardo Baquero Rey*
Autores

Jefferson Miguel Hernández Cáceres
Diagramación

Diego A. Martínez Cárdenas
Coordinador Editorial

Los autores declaran que esta investigación fue financiada por la Fundación Universitaria Los Libertadores en el marco de la Convocatoria de Investigaciones internas de la institución. Los conceptos emitidos en esta publicación son responsabilidad expresa de los autores y no comprometen de ninguna forma a la institución.
Se autoriza la reproducción del texto citando autor y fuente, únicamente con fines académicos. En caso distinto, se requiere solicitar autorización por escrito al editor. Escrito en LaTeX.

Agradecimientos

Los autores expresan especial agradecimiento a todas aquellas personas que de una u otra manera han contribuido para que esta obra haya sido posible, empezando por los estudiantes del semillero de investigación en seguridad informática donde nació la idea del proyecto, a los colegas del programa de Ingeniería de Sistemas, a los colegas del Grupo de Investigación en Ingeniería Aplicada (GUIAS), a los directivos de la Facultad de Ingeniería y Ciencias Básicas, y en general a la Fundación Universitaria Los Libertadores por brindar los espacios y el apoyo financiero necesario.

A Dios, a mi esposa Flor Ángela e hijos (Jefferson, Julieth y Oscar).
Miguel

A Dios, mi familia, colegas y estudiantes
Eduardo

Prólogo

Nos encontramos inmersos en una sociedad donde el consumo de datos sobre diversas plataformas conectados a través de Internet requiere de un alto nivel prestacional por parte de diversos sectores y actores tecnológicos, como proveedores de servicio, fabricantes, así como el sector de la industria del software. Soluciones basadas en aplicaciones que soportan solicitudes para el ingreso o consumo de datos en tiempo real sobre cualquier área: lúdica, consumo, ocio, trabajo, educación, entre otros. Esta variedad de soluciones demanda grandes desafíos, orientados a cada vez más altos niveles de prestaciones que puedan ofrecer, tales como la disponibilidad, robustez, interoperabilidad, integridad, etc., de cada uno de sus componentes, incluyendo aspectos trascendentales propios para lograr mitigar riesgos y vulnerabilidades de seguridad.

La realidad de la cantidad de procesos que se desprenden de soluciones informáticas es tan variada, que, para llevar a cabo su correcto desarrollo bajo especificaciones y estándares, requieren de un alto nivel de madurez en todos y cada uno de los procesos establecidos en las etapas de desarrollo de software. Es así como la presente obra, presenta una de esas realidades inmersas sobre un variado ecosistema de soluciones digitales basadas en software; enfocado puntualmente sobre estrategias y métodos para mitigar riesgos de seguridad dentro de procesos enmarcados en el ciclo de vida del desarrollo de software. Lo anterior, bajo diversas miradas que presenta el mercado informático a partir de metodologías ágiles. Indudablemente una apuesta que permite acercarnos a una de serie de consideraciones y criterios que todo equipo, empresa o casa matriz de desarrollo necesitan conocer e identificar, con el propósito de poder cubrir aspectos esenciales en el despliegue de aplicaciones y soluciones informáticas que logren ser distribuidas sobre el mercado.

Dentro de la presente obra, el lector podrá encontrar apartados orientados a destacar el panorama que presentan las metodologías de desarrollo ágil dentro del contexto de software seguro. Así mismo, la relevancia que presenta el desarrollo ágil, de la mano de estrategias y buenas prácticas para el desarrollo seguro del software. Finalmente, los autores exponen de manera detallada y validando mediante caso de estudio aplicado, una propuesta metodológica partiendo de los principios de ciclo de vida de desarrollo de software, y el uso de estrategias de seguridad a considerar en cada una de las fases planteadas.

Me place encontrar dentro de la literatura nacional, este tipo de apuestas orientadas a ofrecer espacios de análisis, reflexión y propuestas concretas asociadas a métodos de desarrollo de software ágil a partir de estrategias de seguridad, sobre todo, en el marco de una variada industria de software apalancada por emprendimientos de base tecnológica, donde el software como actor del ecosistema digital, es uno de los insumos valiosos para una sociedad que demanda cada vez más servicios en línea.

Paulo Alonso Gaona García, Ph.D

Introducción

El mercado es cada día más exigente en cuanto a desarrollo de software se refiere. Ya no es suficiente con que las casas desarrolladoras de programas informáticos cumplan a cabalidad con los requerimientos de funcionalidad que satisfacen las necesidades del cliente, sino que además se deben involucrar aspectos relevantes de los requisitos no funcionales que de alguna manera mitiguen los posibles riesgos a los que pueda estar expuesto el producto cuando este se encuentre en la etapa productiva, pues el número de vulnerabilidades es cada vez mayor, lo que pone en riesgo la información y, por ende, la continuidad del negocio de las organizaciones.

Las herramientas ágiles buscan dar soluciones informáticas oportunas con la participación activa y directa del cliente, y pueden ofrecer mayor seguridad a las aplicaciones desarrolladas al involucrarlas con un ciclo de vida de desarrollo de software seguro. Esta obra da cuenta de los resultados obtenidos durante una investigación aplicada a la integración de un ciclo de vida de desarrollo de software seguro (S-SDLC), de fundamentos teóricos y herramientas existentes para el desarrollo ágil de este tipo de proyectos y de buenas prácticas de seguridad de la información en los desarrollos de productos software. Para ello, se propuso una metodología implementada bajo las circunstancias del planteamiento de un caso de estudio hipotético, que podría ser real.

Es conveniente reseñar la concepción del proyecto desde el semillero de investigación en seguridad informática, que posteriormente se planteó en el marco de la convocatoria interna de proyectos de investigación institucional en la línea de investigación de Innovación y Emprendimiento, como parte integrante del Grupo de Investigación en Ingeniería Aplicada (GUIAS).

El objetivo fundamental en la seguridad del software es la construcción de aplicaciones informáticas de mejor calidad, más robustas y libres de fallos, que implementen el principio de resiliencia, es decir, que sigan funcionando correctamente ante ataque malicioso (McGraw, 2006).

En respuesta a la creciente tasa de daños y perjuicios causados a las empresas por explotación de las vulnerabilidades de seguridad en los productos de software, se requiere de mayor esfuerzo para desarrollar software más seguro (Keramati y Mirian-Hosseinabadi, 2008).

Para tal efecto, este documento se presenta en seis capítulos, de la siguiente manera:

En el capítulo 1 se realiza un estudio comparativo de diferentes ciclos de vida de desarrollo de software seguro y se determinan sus ventajas y desventajas, para luego establecer la conveniencia de su aplicabilidad con metodologías ágiles de desarrollo de software, partiendo de la flexibilidad y las características que ofrecen, lo que producirá un análisis de resultados y su posterior discusión. Ello servirá de insumo para la industria en este campo.

En el capítulo 2 se estudian los principales elementos y técnicas ágiles implicadas en el desarrollo de un producto software, a fin de determinar los principales elementos aportantes para el desarrollo de la investigación.

El capítulo 3 trata sobre las buenas prácticas y los estándares de seguridad de la información, como la norma ISO 27001 y los Common Criteria (CC); estudia organizaciones como Open Web Application Security Project (OWASP), que aportan esfuerzos importantes para el desarrollo de aplicaciones web seguras; compara herramientas para el desarrollo de software seguro y para el análisis de código, como UMLsec, e identifica las propiedades que debe cumplir un software seguro y las amenazas a dicha seguridad.

En el capítulo 4 se presentan las fases del ciclo de vida de desarrollo de software propuesto, la metodología adoptada, el caso de estudio planteado y su desarrollo en los términos ya enunciados, adoptando el marco de trabajo SCRUM.

Finalmente, en los capítulos 5 y 6 se hace un análisis de los resultados obtenidos y se relacionan las principales conclusiones producto del desarrollo del proyecto, respectivamente.

Con esta investigación se busca adoptar una buena práctica que dé respuesta a requisitos de seguridad que deben contemplar los equipos de desarrollo ágil de software, con el fin de construir productos más estables y robustos, fundamentales en materia de seguridad informática, en pro de garantizar la confidencialidad, disponibilidad e integridad de un producto software. Entretanto, también pretende servir de apoyo a la academia en la formación de nuevos profesionales, como un insumo a tener en cuenta en las líneas de Ingeniería de Software y Seguridad de la Información.

Índice general

1. Ciclos de vida de desarrollo de software seguro **6**
1.1. Introducción . . . 6
1.2. Tipos de ciclos de vida de desarrollo de *software* seguro . . . 6
1.2.1. McGraw's Seven Touchpoints . . . 7
1.2.2. Microsoft Security Development Lifecycle (SDL) . . . 8
1.2.3. Correctness by Construction (CbyC) . . . 9
1.2.4. Comprehensive, Lightweight Application Security Process (CLASP) . . . 10
1.2.5. Software Assurance Maturity Model . . . 11
1.2.6. Team Software Process (TSP) . . . 12
1.2.7. Otras metodologías . . . 16
1.3. Características de los *S-SDLC* . . . 16
1.4. Modelado de amenazas . . . 19

2. El desarrollo ágil en el software **21**
2.1. Introducción . . . 21
2.1.1. El manifiesto ágil . . . 21
2.1.2. Las técnicas ágiles y la madurez de la industria del software . . . 22
2.1.3. Scrum . . . 23
2.1.4. Metodología XP . . . 24
2.1.5. Kanban . . . 24
2.1.6. OpenUP/Basic . . . 27
2.1.7. SD3+C . . . 28

3. Buenas prácticas en seguridad de la información **29**
3.1. Introducción . . . 29
3.2. Estándares . . . 29
3.2.1. Norma ISO 27001 . . . 29
3.2.2. Common Criteria (CC) . . . 30
3.3. Organizaciones . . . 30
3.3.1. Common Weakness Enumeration (CWE) . . . 30
3.3.2. Web Application Security Consortium (WASC) . . . 30
3.3.3. The Open Web Application Security Project (OWASP) . . . 30
3.3.4. Secure Software Programmer . . . 33
3.4. Desarrollo de software seguro . . . 34
3.4.1. Security Requirements Engineering Process (SREP) . . . 34
3.5. Buenas prácticas de seguridad del *software* . . . 34
3.6. Herramientas de desarrollo de *software* seguro . . . 34
3.6.1. Casos de mal uso (casos de abusos) . . . 34
3.6.2. UMLsec . . . 34
3.6.3. Herramientas para análisis de código . . . 35
3.7. Seguridad en el *software* . . . 35

3.7.1. Propiedades de un *software* seguro . . . 35
3.7.2. Amenazas a la seguridad del *software* . . . 36

4. Metodología y caso de estudio **37**
4.1. Introducción . . . 37
4.2. S-SDLC propuesto . . . 37
4.2.1. Fase de Capacitación . . . 37
4.2.2. Fase de Construcción de los requerimientos y Casos de abuso . . . 38
4.2.3. Fase de Análisis de riesgos . . . 38
4.2.4. Fase de Diseño . . . 38
4.3. Fase de Codificación y pruebas . . . 38
4.3.1. Fase de Implementación . . . 38
4.3.2. Fase de Seguimiento . . . 38
4.4. Fase de Auditoría de seguridad . . . 38
4.5. Metodología adoptada . . . 39
4.6. Caso de estudio . . . 42
4.7. Desarrollo del caso de estudio . . . 42
4.7.1. Fase de Capacitación . . . 42
4.7.2. Fase de Construcción de los requerimientos y Casos de abuso . . . 43
4.7.3. Ejecución del *sprint* . . . 50
4.7.4. Fase de Análisis de riesgos . . . 55
4.7.5. Fase de Diseño . . . 57
4.7.6. La seguridad en las bases de datos . . . 59
4.7.7. Seguridad en motores de bases de datos . . . 61
4.7.8. Fase de Codificación y pruebas . . . 69
4.7.9. Algunas pruebas . . . 91
4.8. La seguridad *framework* . . . 91
4.8.1. *Framework* Java . . . 91
4.8.2. *Framework* PHP . . . 97
4.9. Fase de implementación . . . 99
4.9.1. Salvaguardas de los sistemas operativos . . . 99
4.9.2. Arquitectura de la aplicación . . . 100
4.10. Fase de Seguimiento . . . 103
4.11. Fase de Auditoría de seguridad . . . 104

5. Análisis de resultados **107**

6. Conclusiones **108**

Bibliografía **109**

Índice de figuras

1.1. Touchpoints de Gary McGraw . . . 8
1.2. Ciclo de vida de Microsoft SDL . . . 8
1.3. Ciclo de vida de Microsoft SDL . . . 9
1.4. Ciclo de vida de Microsoft SDL . . . 10
1.5. Relación de los niveles de vistas CLASP . . . 11
1.6. Organización SAMM . . . 15
1.7. *STRIDE* . . . 20

2.1. *Valores del manifiesto ágil* . . . 21
2.2. *Principios del manifiesto ágil* . . . 22
2.3. *Enfoque SD3+C* . . . 28

3.1. *Top 10 de OWASP* . . . 31
3.2. *Vulnerabilidades asociadas al Top 10* . . . 32
3.3. *OWASP Cloud-10* . . . 33

4.1. *Fases del S-SDLC propuesto* . . . 37
4.2. *Scrum* . . . 39
4.3. *Ciclo Scrum* . . . 39
4.4. *Scrum adoptado para el proyecto* . . . 40
4.5. *Product backlog* . . . 44
4.6. *Historia de usuario Ingreso al sistema* . . . 44
4.7. *Historia de usuario Registro* . . . 44
4.8. *Historia de usuario Renovación de contrato* . . . 45
4.9. *Historia de usuario Realizar pago* . . . 45
4.10. *Historia de usuario Consultar temas* . . . 45
4.11. *Historia de usuario Buscar revistas* . . . 45
4.12. *Caso de abuso Ingreso al sistema* . . . 45
4.13. *Historia de usuario HU01 - Iteración 3* . . . 47
4.14. *Ajuste historia de usuario HU02* . . . 48
4.15. *Planeación de historia de usuario* . . . 49
4.16. *Historia de usuario HU03 - Iteración 3* . . . 50
4.17. *Sprint propuestos* . . . 51
4.18. *Tablero Kanban para historias de usuario* . . . 52
4.19. *Tablero Kanban incluyendo seguridad* . . . 52
4.20. *Diagrama de clases del usuario* . . . 58
4.21. *Diagrama de clases de la revista* . . . 59
4.22. *Seguridad en Oracle* . . . 61
4.23. *Tabla de Usuario* . . . 62
4.24. *Modelo E-R para Suscriptor* . . . 64
4.25. *Modelo E-R para la Revista* . . . 65
4.26. *Capas de la aplicación* . . . 69

4.27. *Estadísticas de los lenguajes de programación* 70
4.28. *To de los leguajes* 71
4.29. *Seguridad en .Net* 72
4.30. *Seguridad en PHP* 73
4.31. *Seguridad en Java* 74
4.32. *Pruebas de ingreso al sistema* 76
4.33. *Captcha* 82
4.34. *Apache Shiro* 92
4.35. *Java de cifrado simplificado* 94
4.36. *Spring Security* 95
4.37. *jGuard* 96
4.38. *Codelnigter* 97
4.39. *Laravel* 98
4.40. *Autenticación y autorización en Symfony* 98
4.41. *Aspectos de seguridad en profundidad* 99
4.42. *Elementos de una arquitectura segura* 100
4.43. *Internet Information Server (IIS)* 101
4.44. *Servidor Apache* 101
4.45. *Servidor NGINX* 102
4.46. *Servidores web más utilizados* 102

Índice de tablas

1.1. *Relación de actividades sugeridas para el desarrollo CLASP* 12
1.2. *S-SDLC* vs. actividades de ingeniería de requisitos y diseño 17
1.3. *Actividades de implementación y verificación* 18
1.4. *S-SDLC vs. recursos y usos a nivel empresarial* 19

2.1. *Características generales en las metodologías ágiles* 23
2.2. *Características comunes en metodologías ágiles* 25
2.3. *Seguridad en XP y Scrum* 26
2.4. *Flexibilidad del tablero Kanban* 27

4.1. Metodologías ágiles integradas en el S-SDLC propuesto 42
4.2. *Historia de usuario del caso de estudio* 43
4.3. *Epopeya para sistema de logueo* 43
4.4. *Identificación de riesgos* 55
4.5. *Valores cuantitativos del riesgo* 55
4.6. *Matriz de riesgos* 56
4.7. Tarjetas CRC 57
4.8. Categorización de los lenguajes de programación 70
4.9. *Pruebas a historias de usuario* 76
4.10. *Pruebas de caja blanca* 77
4.11. *Pruebas de caja negra - Ingreso al sistema* 78
4.12. Pruebas de caja negra - Búsqueda de revistas 78
4.13. Pruebas de caja negra - Registrar un suscriptor 93
4.14. *Cronograma de trabajo* 104
4.15. *Checklist de auditoría* 105

Capítulo 1

Ciclos de vida de desarrollo de software seguro

1.1. Introducción

En el proceso de desarrollo de *software* normalmente se involucra lo que se conoce como ciclo de vida de desarrollo de *software*, donde se agotan una serie de fases o etapas, con sus respectivas ventajas y desventajas según el paradigma adoptado. Hay diversos modelos y algunos de ellos tienen un enfoque de desarrollo de *software* seguro. En este capítulo se estudian varios modelos y se realiza un análisis comparativo para resaltar sus principales características, que a la vez pueden ser considerados para los fines de la investigación.

1.2. Tipos de ciclos de vida de desarrollo de *software* seguro

Actualmente, la seguridad es un requerimiento no funcional que puede ser definido como la calidad y capacidad del *software* al momento de enfrentarse a la explotación de las vulnerabilidades causadas por las aplicaciones. Por eso, se han diseñado una serie de metodologías incluidas en el proceso de desarrollo de *software* seguro, en pro de eliminar o mitigar dichas vulnerabilidades e integrar la seguridad como un componente básico en la arquitectura de cualquier producto de *software*(Kang J & Park J. H. 2018). Existen varias metodologías que establecen una serie de pasos o etapas denominadas ciclo de vida de desarrollo de *software* seguro (S-SDLC, por sus siglas en inglés), que promueven un *software* más seguro y capaz de resistir a ataques. A continuación, se describen algunas de ellas:

1.2.1. McGraw's Seven Touchpoints

Gary McGraw (2006) plantea siete puntos clave en el desarrollo de un producto de *software* seguro, establecidos en 2004 por el Instituto de Ingeniería Eléctrica y Electrónica (IEEE, por sus siglas en inglés), conocidos como el modelo de Gary McGraw y Cigital. Es considerado uno de los tres pilares de la seguridad de *software*, ya que une los aspectos técnicos y prácticos del desarrollo y de esa manera hace énfasis en el empleo de buenas prácticas. Para ello, recomienda tener en cuenta una revisión externa y los siguientes aspectos:

- **La revisión del código**. Los proyectos de *software* producen al menos un código fuente por artefacto. A nivel de código, la atención se centra en los errores de ejecución y en identificar las vulnerabilidades que puedan afectar el desarrollo del producto. Para brindar una mayor protección al *software* a prueba de intrusiones, es necesario completar todas las fases del modelo.

- **Análisis de riesgos de arquitectura**. En el ámbito del diseño y la arquitectura, un sistema debe ser coherente y presentar un frente de seguridad unificada que se constituye por el trabajo en equipo entre los arquitectos, diseñadores y analistas. De esta manera se genera una documentación desde diferentes ópticas. Al implementar un proceso de análisis de riesgos y seguridad se brindan al arquitecto pautas para el inicio de la construcción, sin descartar una probable aparición de amenazas en cualquier etapa del proyecto. Estos análisis deben ser constantes durante todo el proceso de construcción de la aplicación de *software*.

- **Las pruebas de penetración**. Es una etapa muy útil, especialmente si el análisis de riesgos de arquitectura está orientado a las pruebas, y se realiza mediante intrusiones de *hacking* ético controlado, con el fin de identificar inconvenientes que se pudieran presentar a futuro, cuyo propósito es implementar procesos para evitar y prevenir los posibles ataques.

- **Pruebas de seguridad basadas en el riesgo**. Cubren dos estrategias: la prueba de funcionalidad de seguridad con técnicas estándar de pruebas funcionales y la prueba de seguridad basada en el riesgo con base en patrones de ataque, ya que estos no siempre son evidentes. Estas pruebas de seguridad deben contemplar actividades orientadas a la identificación, análisis y seguimiento del riesgo.

- **La construcción de los casos de abuso**. Según Gary McGraw (2006), es una forma directa y cercana de entrar en la mente del atacante, pues la creación de los casos de abuso requiere una cobertura explícita para identificar, revisar y fortalecer lo que debe ser protegido, de qué o quién y por cuánto tiempo.

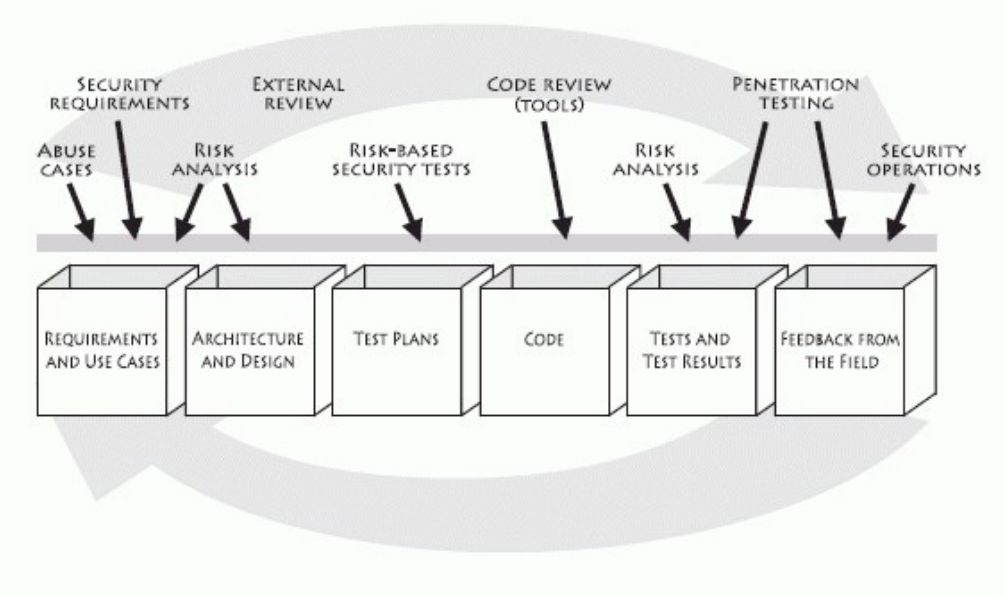
- **Los requisitos de seguridad**. Aunque en el inicio de todo proyecto de *software* se determinan los requerimientos funcionales y no funcionales desde la percepción del usuario, estos deben estar enfocados al proceso de seguridad. Además, deben ser validados y actualizados periódicamente, con el fin de restablecer y renovar los protocolos y procesos dispuestos para este fin.

- **Las operaciones de seguridad**. Los ataques ocurren, independientemente del esfuerzo de diseño e implementación, por lo que el monitoreo del comportamiento del *software* es una técnica defensiva esencial. Igualmente, el comportamiento de seguridad del *software* debe ser monitoreado de manera recurrente y continua, lo cual se ha convertido actualmente en la principal forma de defensa de un sistema de información. Su punto de partida se encuentra en la identificación de riesgos.

- **Revisión externa**. Es una evaluación imparcial y objetiva realizada por personal externo al proyecto.

La **Figura 1.1** ilustra los siete puntos propuestos por Gary McGraw.

Figura 1.1: Touchpoints de Gary McGraw



Fuente: http://www.swsec.com/resources/touchpoints/

1.2.2. Microsoft Security Development Lifecycle (SDL)

El proceso SDL de Microsoft (2016) es una metodología compuesta por 16 actividades divididas en las fases que componen la técnica y que están encaminadas al mejoramiento del desarrollo de *software*. Fue propuesta en 2004 por Microsoft e incluye un proceso de modelado de amenazas, que busca identificar vulnerabilidades a nivel de código.

Con el propósito de optimizar el acoplamiento según el tipo de desarrollo, SDL cuenta con dos versiones de ejecución:

SDL versión rígida, enfocada en equipos de proyectos y desarrollo de productos de gran envergadura donde los cambios son mínimos.

SDL versión ágil, cuyos desarrollos son incrementales con aumento en la frecuencia de ejecución y seguimiento de actividades, haciendo énfasis en su seguridad. Recomendada para desarrollos web.

Como parte diferencial de las metodologías de desarrollo de *software* seguro, SDL cuenta con una etapa de "entendimiento de seguridad" y "aseguramiento", en la cual se da seguimiento a cada uno de los resultados del proceso.

La **Figura 1.2** ilustra las fases del ciclo de vida Microsoft Security Development Lifecycle (SDL).

Fuente: https://www.microsoft.com/en-us/sdl/

Figura 1.2: Ciclo de vida de Microsoft SDL

La **Figura 1.3** describe las fases de esta metodología. Fuente: Microsoft.

Figura 1.3: Ciclo de vida de Microsoft SDL

1.2.3. Correctness by Construction (CbyC)

Esta metodología busca la creación de código de manera correcta desde su inicio, procurando corregir y eliminar errores desde su ingreso, para lo cual se apalanca en procesos rigurosos de seguridad con requerimientos muy detallados.

En una primera entrega, la metodología CbyC presenta una visión completa del sistema con funcionalidad en general limitada, pero que permitirá los ajustes y mejoras en cada iteración (Hall y Chapman, 2002).

De acuerdo con Amey (2006), corresponde a un método práctico para desarrollar aplicaciones de *software* que requieren un nivel de seguridad crítico y que además sea demostrable. A continuación, se describen las principales actividades de las fases involucradas en este ciclo de vida:

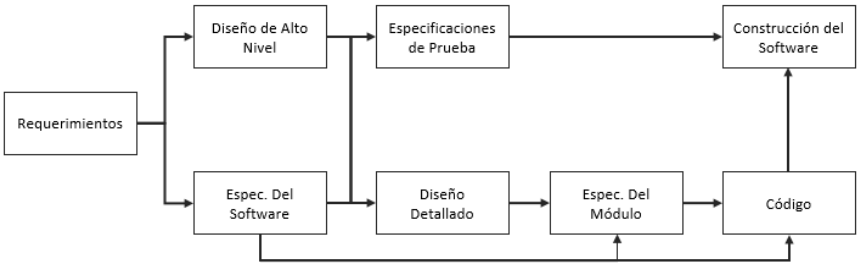
- **Fase de requerimientos**. Se especifica el propósito, las funciones, los requerimientos de usuario y se tiene en cuenta la definición de requerimientos no funcionales expresados en los diagramas de clases. Cada requerimiento debe pasar por el proceso de validación de seguridad a la amenaza correspondiente.
- **Fase de diseño de alto nivel**. Detalla la estructura interna del sistema, como la estructura de la base de datos y los mecanismos para las transacciones y comunicaciones, teniendo como punto principal los

requerimientos no funcionales en cuanto a su ámbito de seguridad en cada punto álgido.

- **Fase de especificación del *software***. Es el espacio donde se fundamentan las especificaciones de la interfaz de usuario (*look* and *feel*), mediante el desarrollo de un prototipo para la posterior validación con el cliente.
- **Fase de diseño detallado**. Se define la serie de módulos y actividades y la funcionalidad referida.
- **Fase de especificación de los módulos**. Define el estado y el comportamiento encapsulado en cada módulo de la aplicación de *software*, apoyado en los procesos que contienen información.
- **Fase de codificación**. Escenario para impedir cualquier ambigüedad posible, incluyendo también el código de la aplicación de *software*. En esta fase se induce el desarrollo de pruebas con el fin de disminuir y eliminar los errores presentes en el código, y se podrá definir de forma paralela si debe ser creada una nueva versión del código.
- **Fase de especificación de pruebas**. Como particularidad en la metodología CbyC, no es necesario ejecutar un proceso de pruebas de unidad ni caja blanca, puesto que su foco central es realizar estas validaciones a nivel de sistema y con miras al cumplimiento de las especificaciones, comportamientos y requerimientos no funcionales.
- **Fase de *software* del software CbyC**. Utiliza la metodología de tipo ágil. En la primera entrega se cuenta con una estructura de todas las interfaces y elementos de comunicación, con una funcionalidad muy limitada. La metodología presenta un enfoque técnico con el cual pretende disminuir los defectos, aumentando considerablemente su capacidad para minimizar fallas, lo que beneficia su implementación en el desarrollo de *software*.

La **Figura 1.4** describe el ciclo de vida CbyC, correspondiente a la secuencia establecida para la construcción de sus fases, cuyas pautas emplean la metodología de desarrollo ágil, en el cual se reciben retroalimentaciones a medida que se entrega y valora el producto en construcción. Lo anterior permite adelantar actividades en paralelo como el diseño de alto nivel y la especificación del software, especificaciones de prueba y diseño detallado.

Figura 1.4: Ciclo de vida de Microsoft SDL



Fuente: http://recibe.cucei.udg.mx/revista/es/vol2-no3/computacion05.html

1.2.4. Comprehensive, Lightweight Application Security Process (CLASP)

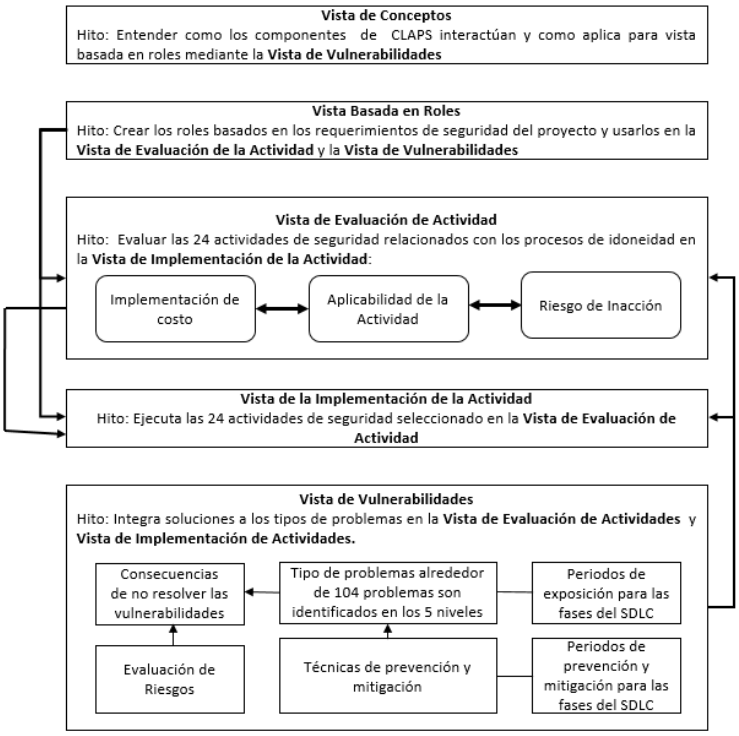
El uso de esta metodología dirige a los desarrolladores en el proceso de revisión y validación desde las primeras etapas del ciclo de vida del desarrollo de *software*, de manera que este proceso se convierta en estructurado (The Open Web Application Secutity Project [OWASP], 2016).

Facilita su integración a cualquier desarrollo de *software*, articulando cada una de las actividades aplicadas a uno o más roles previamente identificados: gerente del proyecto, arquitecto, especificador de requerimientos, diseñador, implementador (equipos de desarrollo), *tester* (grupo de pruebas), auditor de seguridad (De Win,

B. Scandariatoc R. Buyens, K. Grégoire, J. Joosen, W. , 2009)

Vista CLASP. La metodología plantea 104 fallas de seguridad agrupadas en 5 niveles de vistas, que tienen consigo actividades relacionadas en la **Figura 1.5**.

Figura 1.5: Relación de los niveles de vistas CLASP



Fuente: Owasp.org

Las 24 acciones indicadas en las vistas de implementación y evaluación de actividades para la metodología se describen en la Tabla 1.1.

Recursos CLASP. Como apoyo a los procesos de planificación y ejecución de actividades, CLASP presenta una serie de recursos o herramientas que permiten el acceso a los diferentes artefactos.

Caso de uso de vulnerabilidad. Escenario donde se describen los puntos en los cuales se presenta vulnerabilidad en las aplicaciones de software, que pretende dar al usuario una vista fácil al respecto y establecer claras relaciones de causa y efecto.

1.2.5. Software Assurance Maturity Model

El Software Assurance Maturity Model (SAMM) es un marco abierto para ayudar a las organizaciones a formular y aplicar una estrategia para la seguridad del software que se adapte a los riesgos específicos que enfrenta la organización. Los recursos proporcionados por SAMM ayudarán a:

- Evaluar una organización de seguridad de *software*.
- Construir un programa de seguridad de *software* equilibrado mediante estrategias de seguridad con enfoque en las iteraciones que se puedan definir.
- Demostrar mejoras concretas para un programa de garantía de la seguridad.

Tabla 1.1: *Relación de actividades sugeridas para el desarrollo CLASP*

Actividad	Propósito	Rol	Frecuencia
Direccionar los problemas de seguridad reportados.	Certificar que los riesgos identificados sean considerados y direccionados de manera correcta.	Diseñador.	Cuando se requiera.
Diseños de clase con propiedad de seguridad.	Elaborar políticas de seguridad para los campos de datos individuales.	Diseñador.	Una vez por iteración.
Aplicar los diseños de seguridad del proyecto.	Reforzar el diseño de aplicaciones mediante la implementación de diseños de seguridad. Identificar los riesgos de seguridad que puedan ser proporcionados por terceros.	Diseñador.	Al menos una vez por iteración.

- Definir y medir las actividades relacionadas con la seguridad dentro de una organización. SAMM presenta cuatro funciones principales, cada una compuesta por tres prácticas de seguridad, como lo muestra la **Figura 1.6**.

SAMM define cuatro funciones críticas de negocio de alto nivel, a saber:

Gobernabilidad. Está centrada en la definición de la estrategia y en los procesos y políticas relacionados con la gestión del S-SDLC en una organización.

Construcción. Se refiere a los procesos y actividades que la organización debe seguir para el desarrollo de la aplicación, lo cual incluye la administración del producto, la recolección de requerimientos, las especificaciones de la arquitectura a alto nivel, la definición del diseño detallado y la implementación de la aplicación.

Verificación. Se enfoca en los procesos relacionados con la revisión y pruebas de los artefactos producidos durante el *desarrollo* del *software*; incluye aseguramiento de calidad y diferentes tipos de pruebas.

Implementación. Son las actividades relacionadas con la liberación de la aplicación.

Desarrollo. Espacio donde se aborda la gestión de vulnerabilidades, el fortalecimiento del entorno y las actividades operativas.

1.2.6. Team Software Process (TSP)

Proporciona un marco de procesos y métodos disciplinados para la aplicación de los principios de ingeniería de *software* en el equipo a nivel individual. El uso de TSP ayuda a las organizaciones a establecer una práctica de ingeniería madura y disciplinada que produce software seguro y confiable en menos tiempo y con costos más bajos (Software Engineering Institute [SEI], 2010).

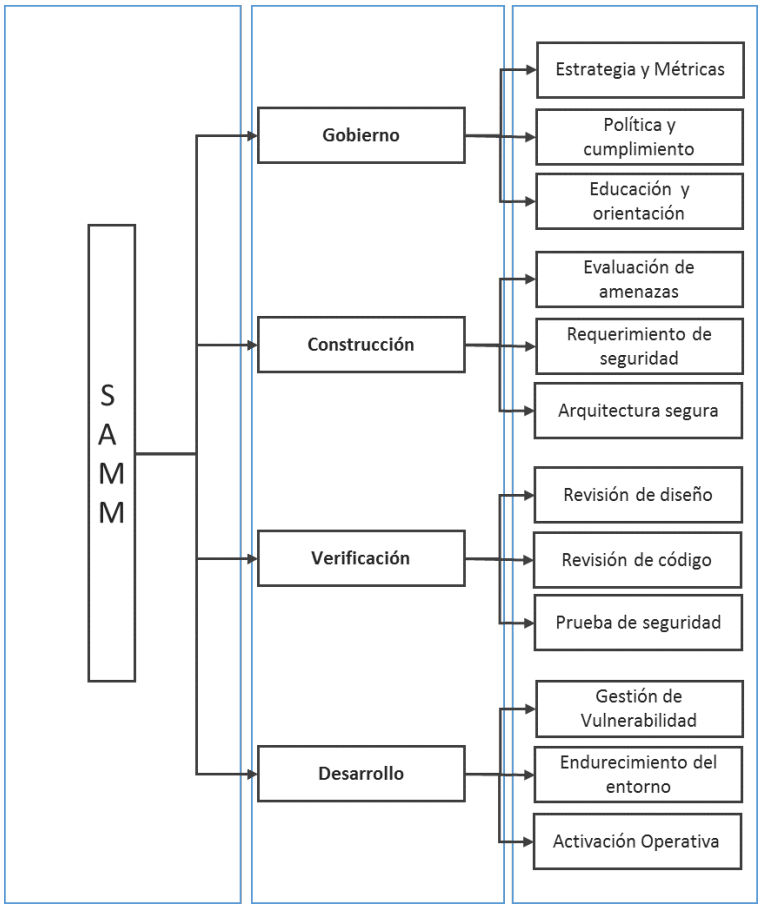
Algunos de los aspectos más importantes de TSP son: Planeación de calidad. El equipo desarrolla una estimación del número de defectos que serán inyectados durante el desarrollo y luego construye un plan para eliminar esos defectos. Una de las principales diferencias filosóficas es que los equipos TSP intentan eliminar la gran mayoría de los defectos, si no todos, antes de las pruebas del sistema de software.

Planificación y estimación. Son realizadas por los miembros individuales del equipo orientado por un entrenador TSP, lo cual les permite mayor apropiación de sus planes y compromisos. Las estimaciones se vuelven más precisas a medida que cada miembro del equipo mejora sus habilidades de estimación, a través

Actividad	Propósito	Rol	Frecuencia
Construir guía operacional de seguridad.	Proporcionar a los interesados documentación en medidas operacionales de seguridad. Suministrar los documentos necesarios que formalicen el proceso de seguridad de la aplicación.	Implementador.	Una vez por iteración.
Diseño de los casos de uso indebido.	Informar a los interesados sobre los riesgos y las razones de las decisiones relevantes que han sido tomadas para la seguridad.	Especificador de requerimientos.	Según sea necesario, sin importar las veces en cada ejecución.
Documentación de requerimientos de seguridad.	Documentar a nivel de negocio los requisitos funcionales de seguridad.	Especificador de requerimientos.	Una vez por iteración y cuando sea necesario.
Identificar superficie de ataque.	Especificar estructuradamente cada punto de entrada *(input)* del programa. Facilita el proceso de análisis.	Diseñador.	Recomendado al terminar la fase de diseño y continuar en la fase de elaboración.
Identificar la política general de seguridad.	Proporciona los requerimientos de seguridad del negocio. Compara el nivel de seguridad de varios productos de la org.	Especificador de requerimientos.	Una vez por iteración.
Identificar los recursos y límites de la confianza.	Identificar la base estructurada para interpretar los requisitos de seguridad del sistema	Arquitecto.	Una vez por iteración.
Identificar los roles de los usuarios y las capacidades de los recursos.	Precisar los roles del sistema, así como las capacidades y recursos a los que tendrá acceso cada uno.	Arquitecto.	Una vez por iteración.
Identificar, implementar y realizar pruebas de seguridad.	Identificar riesgos que no han sido evidenciados en la fase de implementación o que han sido identificados en el proceso operativo. Se comporta como mecanismo de captura de fallos.	*TESTER*	Generalmente, repetidas veces durante cada iteración.
Implementar interfaz de contratos.	Identificar los errores a nivel de fiabilidad desde el inicio del proceso.	Implementador.	Cuando son modificados los métodos o las funciones.
Programa institucional de sensibilización de seguridad.	Asegurar que los miembros del proyecto podrán enfrentar los inconvenientes de seguridad con la suficiente eficacia para volverla un objetivo mediante la entrega de cuentas.	Gerente del proyecto.	En marcha.

Actividad	Propósito	Rol	Frecuencia
Integrar la seguridad en el origen del proceso de admiración.	Automatizar el análisis de seguridad a nivel de los procesos de aplicación y la recolección de datos.	Implementador.	Cuando sea necesario.
Controlar las estadísticas de seguridad	Llevar control métrico de los casos de seguridad presentados.	Gerente del proyecto	En marcha.
Generar firma del código.	Proporcionar a los interesados la forma de validar el origen y la integridad del **software** entregado.	Implementador.	Una vez liberada la aplicación o desarrollo.
Hacer análisis de seguridad de los requisitos y diseño (modelado de amenazas).	Identificar las amenazas, mitigar los riesgos y fortalecer los requisitos de seguridad establecidos.	Auditor de seguridad.	Cuando sea necesario.
Ejecutar monitoreo de seguridad en la fuente.	Encontrar las vulnerabilidades a nivel de **software**.	Auditor de seguridad.	Incremental, al final de cada iteración.
Investigar y evaluar la seguridad de las soluciones tecnológicas.	Evaluar los niveles de seguridad que se puedan presentar por terceros.	Diseñador.	Cuando sea necesario.
Especificar la configuración y nivel de seguridad de las BD.	Especificar los niveles de seguridad para los usuarios y terceros que puedan acceder a la base de datos.	Diseñador (BD).	Cuando sea necesario
Especificar el entorno operacional.	Documentar los supuestos y requisitos de seguridad, de manera que pueda ser validado el impacto operacional.	Especificador de requerimientos.	Cuando sea necesario.
Verificar los atributos de seguridad de los recursos.	Confirmar que los procesos de seguridad establecidos en la política se encuentren en el *software*.	*tester*	Una vez por iteración.

Figura 1.6: Organización SAMM



Fuente: SAMM.

del uso de datos reales.

Proceso de *software* personal. Los desarrolladores individuales están específicamente capacitados en técnicas personales para producir software de alta calidad con previsibilidad mejorada del horario.

Equipos autogestionados. Los miembros del equipo participan en la gestión general del proyecto. Metrics Framework y datos. Cuatro medidas simples se capturan mientras las personas están haciendo el trabajo. Estas medidas permiten que los individuos y los equipos realmente entiendan dónde están y cómo está funcionando el proceso. El TSP incluye un conjunto completo de métricas de calidad y planificación.

Entrenamiento. Un entrenador TSP trabaja hombro a hombro con el equipo al implementar el TSP. Esto asegura que se estén llevando a cabo todas las actividades de TSP según lo planeado y que cada individuo sea consciente, a través de sus métricas personales, de qué tan bien está desempeñando su labor y dónde puede mejorar (Wyrwa, 2012).

1.2.7. Otras metodologías

- Oracle Software Security Assurance: sus productos de servidor de bases de datos han logrado consistentemente altas calificaciones de certificación de seguridad en todos los criterios en los que participa. Las plataformas donde se realizan las evaluaciones incluyen versiones evaluadas de Linux u Oracle Solaris.
- Appropriate and Effective Guidance in Information Security (AEGIS).
- Rational Unified Process-Secure (RUPSec).
- Secure Software Development Model (SSDM).
- Waterfall-Based Software Security Engineering Process Model.

1.3. Características de los *S-SDLC*

En las siguientes tablas se relacionan y establecen las principales características de los *S-SDLC*, incluyendo además de los mencionados a *CLASP*, *TSP-Secure* y *OPENSAMM:*

Tabla 1.2: *S-SDLC* vs. actividades de ingeniería de requisitos y diseño

Item	SDL	CbyC	McGraw's Seven Touchpoints	CLASP	TSP-secure	OPEN-SAMM
Actividades de ingeniería requisitos	La fase de planificación es el espacio apropiado para la definición y el establecimiento de la seguridad de la fiabilidad de una aplicación de software. Al conocer los riesgos, se podrán identificar y corregir los errores de seguridad.	En la fase de requerimientos se especifica el propósito, las funciones y los requerimientos no funcionales. Cada requerimiento debe pasar por un proceso de trazabilidad de requerimiento de seguridad a la amenaza.	La seguridad debe ser ocupada directamente a nivel de requisitos. Se deben incluir tanto las necesidades de seguridad como de seguridad funcional. Brinda un buen conocimiento del *software* alineado en su entorno real. Se identifican activos, entorno y riesgos.	Los requisitos de seguridad constan de los siguientes pasos: i) identificar las funciones y los recursos del sistema y ii) determinar las interacciones de recursos a través del tiempo de vida del sistema.	Actividades de requerimientos. Algunos porcentajes de las vulnerabilidades. Se modelan las amena zas o el desarrollo de los casos de abuso.	Los requisitos de seguridad deben estar integrados con las necesidades del negocio, como la autenticación, la autorización de los usuarios y la validación de las entradas de datos.
Actividades de diseño	Es la etapa del ciclo de vida para determinar requerimientos de diseño, análisis de los espacios de ataques, modelos de amenazas y medidas para mitigar el riesgo utilizando un análisis dinámico.	En la etapa de diseño de alto nivel se puntualiza en la estructura interna del sistema (como la estructura de las bases de datos, elementos para las transacciones y comunicaciones) .	Los diseñadores arquitectos y analistas deben documentar de forma clara las hipótesis e identificar posibles ataques, para la fase de la arquitectura basada en las especificaciones del análisis y diseño de los riesgos.	Compromisos: investigar las tecnologías que satisfacen los requisitos de seguridad. Se documenta la superficie de ataque de una aplicación de *software.*	Se tienen en cuenta ciertos porcentajes de vulnerabilidades que son incluidas durante las actividades de los requerimientos y el diseño de acti vidades.	Se orienta a la evaluación del diseño del *software* y los problemas relacionados con la arquitectura, detectando problemas de manera anticipada.

Tabla 1.3: *Actividades de implementación y verificación*

Item	SDL	CbyC	McGraw's Seven Touchpoints	CLASP	TSP-secure	OPEN-SAMM
Actividades purebas de verificación y validación	Los equipos de desarrollo definen y establecen una lista de las herramientas aprobada por el líder de seguridad del equipo del proyecto; el equipo debe realizar un análisis estático al código fuente.	Los módulos se caracterizarán por tener bajo acoplamiento y alta cohesión. Utiliza un desarrollo iterativo y la revisión del código.	Las actividades constructivas se orientan al diseño, la defensa, y la funcionalidad, se plantean cambios a realizar en la especificación de requisitos, diseño o implementación para mitigar o elimi nar los riesgos.	El equipo de desarrollo por lo general tiene la experiencia en seguridad. Siguen y cumplen los requisitos seguros de codificación, las políticas y normas.	Algunas vulnerabilidades son revisadas durante las actividades de la determinación de requerimientos y diseño de actividades y codificación	Administrar vulnerabilidades implica el manejo de estas y de incidentes de seguridad, así como la recolección de métricas e información.
Actividades de pruebas de verificación y validición.	El *software* ya es totalmente funcional. Se realizan pruebas al *software* en el área de la superficie de ataque.	En las pruebas se tienen en cuenta las especificaciones del *software*, los requerimientos y el diseño de alto nivel. Se reali zan pruebas de valores límites y pruebas de comportamiento a los requerimientos no funcionales.	La prueba de funcionalidad de la seguridad se lleva a cabo con técnicas estándar de pruebas funcionales y pruebas de seguridad, con base en el riesgo a partir de patrones de ataque. Un buen plan de pruebas de seguridad hace ambas cosas.	Las pruebas se realizan para requisitos de seguridad, además de los requisitos de negocio. Se pueden ejecutar las herramientas de evaluación automatizadas.	Elimina las vulnerabilidades inyectadas durante las fases anteriores, a partir de la revisión de código, el análisis dinámico, el análisis estático y las pruebas de seguridad.	Verificar y validar el código comprende los requerimientos del negocio relacionados con la disponibilidad, integridad y confidencialidad, así como una revisión del Top 10 de OWASP.

Tabla 1.4: *S-SDLC vs. recursos y usos a nivel empresarial*

Item	SDL	CbyC	McGraw's Seven Touchpoints	CLASP	TSP-secure	OPEN-SAMM
Recursos	Recurso humano capacitado y actualizado en asistencia de personal de seguridad, tecnologia y economía.	Recurso humano capacitado y actualizado en tecnologia y economía.	Recurso humano capacitado y actualizado en tecnologia y economía. Esta metdología es cíclica y tiene un octavo ítem que corresponde a una revisión externa, otra visión del exterior.	Recurso humano capacitado y actualizado en tecnologia y economía.	Recurso humano capacitado y actualizado en tecnologia y economía.	Recurso humano capacitado y actualizado en tecnologia y economía. Algunas herramientas para la revisión de código, como CodeCrawler, Orion y O2.
Uso por la empresa	Integración de la versión ágil del SDL con metodologías agiles, aplicado a un caso práctico, utiliza lenguajes de la *suite* de Microsoft.	Empresas de diferentes sectores.	Empresas de los sectores aeronáutico, aeroespacial, defensa, seguridad de la información, seguridad y emergencias, sanidad, transporte,telecomunicaciones y TIC.	Sectores energéticos. Empresas del sector público y privado.	Pequeñas y medianas empresas.	SAMM se definió pensando en la flexibilidad, de modo que pueda ser utilizado por las organizaciones pequeñas, medianas y grandes.

1.4. Modelado de amenazas

Es una herramienta formal que puede aplicarse al ciclo de vida del desarrollo de software, la cual le permite establecer y evaluar los riesgos y amenazas a las que puede exponerse una aplicación de software.

Un modelo útil es STRIDE (Microsoft, 2009), que puede agrupar amenazas por categorías, como las presentadas en la **Figura 1.7**.

El modelado de amenazas *STRIDE* es una forma de garantizar que las aplicaciones no sufran a la suplantación de identidad, la manipulación de datos, el repudio, la revelación de información, la denegación de servicios y la elevación de privilegios. En la siguiente tabla se relacionan las amenazas con las propiedades que protegen contra ellas.

Figura 1.7: *STRIDE*