Estado del arte

Ing. Felipe de Jesús Miramontes Romero

Centro de Investigación en Matemáticas A.C.,
Maestría en Ingeniería de Software,
Avenida Universidad 222, La Loma, 98068, Zacatecas, México.
felipemiramontesr@gmail.com
http://www.ingsoft.mx

1 Introducción

Como base fundamental de este estudio se ha realizado una revisión sistemática de la literatura acerca del desarrollo de aplicaciones en red con características enfocadas a la seguridad y las técnicas existentes para su correcta construcción, el resultado del análisis muestra que la literatura existente posee una clara tendencia hacia la generalización de los tipos de Software pues en su mayoría las técnicas estudiadas consideran a las aplicaciones en red como Software genérico, por otra parte y a manera de excepción existen muy pocas que tratan el tema de forma particular.

2 Protocolo de la revisión sistemática

El protocolo de la revisión sistemática es definido como una serie de pasos o etapas que pretenden ayudar al investigador a reunir el suficiente material de estudio sobre alguna temática de importancia, entre los principales beneficios obtenidos se encuentran el enriquecimiento de la base de conocimientos, las actividades a realizar y las decisiones tomadas a lo largo de la investigación, a continuación son mencionadas cada una de las etapas que lo conforman.

1. Planificación.

   En esta etapa es necesario establecer los objetivos y el rumbo que debe tomar la investigación. A continuación son mencionadas las actividades que se deben realizar.

   (a) Realizar una adecuada elección del tema de investigación.
   (b) Seleccionar cadenas y fuentes de búsqueda.
   (c) Establecer criterios para la elección de estudios primarios y secundarios.

2. Revisión o ejecución.

   En esta etapa es necesario ejecutar ciertas actividades que garantizan un correcto desarrollo. Las actividades se mencionan a continuación.

(a) Ejecutar las búsquedas de información.
(b) Evaluar la calidad de la información.
(c) Revisar cada uno de los estudios seleccionados.
(d) Extraer la información relevante y necesaria.
(e) Documentar cada una de las interacciones para llevar un registro histórico que permita controlar el rumbo de la investigación.

3. Publicación.

   En esta etapa es necesario exponer de manera formal los resultados de nuestra investigación por medio de la redacción de un documento formal, en este caso la tesis, en la cual se deben incluir cada uno de los cálculos estadísticos y numéricos realizados.

3 Revisión sistemática para la tesis

El objetivo principal de la Revisión Sistemática de la Literatura (RSL) en la ingeniería de software es proporcionar los medios para suministrar la evidencia de mayor calidad y formalidad de la investigación actual e integrarla con la experiencia practica para lograr la mejor toma de decisiones en relación con el desarrollo y el mantenimiento de Software. La revisión sistemática sera realizada para resolver una problemática concreta, en este caso en particular, la temática que se pretende abordar es la construcción de una técnica para el desarrollo de aplicaciones seguras en red (Aplicaciones Web). La revisión sistemática se apega a las necesidades esta investigación pues sus principales metas son identificar, evaluar y interpretar la mayoría de los recursos literarios disponibles acerca de un tema, cuestión, tópico, área o fenómeno de interés, mismas que sirvieran como base para el desarrollo de la tesis y que además mostraran a los interesados el estado actual de la linea de investigación (Estado del Arte). Debido a la formalidad que la revisión sistemática posee y debido a su reconocimiento como protocolo en comparación con métodos tradicionales se espera que la construcción y redacción del estado del arte sea realizado en tiempo y forma.

Para la recolección del material de estudio utilizando la revisión sistemática se formularon las siguientes preguntas de investigación:

1. ¿Cuáles técnicas son usadas para desarrollar aplicaciones en red seguras?
2. ¿Cuáles son los principales beneficios que las técnicas para el desarrollo de aplicaciones en red seguras aportan a los involucrados en el desarrollo del producto?
3. ¿Cuáles son las principales deficiencias existentes en las técnicas para el desarrollo de aplicaciones en red seguras?

Una vez definidas las preguntas de investigación fueron creadas cadenas de palabras unidas por medio de operadores binarios y fueron usadas para optimizar

la búsqueda e identificación del material de investigación apropiado. Dichas cadenas fueron formadas por medio de palabras claves ligadas a la temática principal y las cuales a continuación son mencionadas:

1. Técnicas AND Desarrollo AND Aplicaciones en Red AND Seguras
2. Deficiencias OR Problemas AND Técnica AND Desarrollo AND Aplicaciones en Red AND Seguras
3. Aseguramiento AND Seguridad AND Aplicaciones en red
4. Desarrollo de Software AND Seguro

Una vez ejecutada la búsqueda de las cadenas anteriormente mencionadas los estudios obtenidos fueron analizados y evaluados para su inclusión en la literatura de estudio por medio de cumplimento de los siguientes criterios de aceptación:

1. ¿La referencia se encuentra publicada en idioma ingles o español?
2. ¿La referencia explícitamente ha sido publicada en años posteriores al 2009 o se ha actualizado en años posteriores al 2009?
3. ¿La referencia proporciona datos fiables y comprobables?
4. ¿La referencia explícitamente discute algún aspecto sobre técnicas para el desarrollo de aplicaciones en red seguras?

4 Literatura incluida en el estado del arte

La literatura que se ha incluido en el estado del arte ha cumplido las paramétricas establecidas a lo largo del desarrollo del proceso definido como revisión sistemática, se han incluido trabajos realizados por entidades académicas y de investigación así como trabajos desarrollados por entidades privadas, empresas, organizaciones gubernamentales y comunidades abiertas, todo esto con el fin de establecer una base confiable para el desarrollo de la nueva técnica para el desarrollo de aplicaciones seguras en red. A continuación son presentadas las técnicas recabadas en el estudio.

4.1 Ciclo de Vida de Desarrollo Seguro

Security Development Lifecicle (SDL, Ciclo de Vida de Desarrollo Seguro) se define como un proceso para el control de la seguridad orientado a Software el cual se ha convertido en un requisito de carácter obligatorio en Microsoft desde el año 2004.

¿Qué es Security Development Lifecicle (SDL)? Es un proceso enfocado en el desarrollo de Software seguro que combina una parte teórica y una parte practica por medio de las cuales inserta técnicas y guías de seguridad en todas las fases del Ciclo de Vida de Desarrollo de Software (CVDS) [1]. SDL tiene como principal meta reducir la gravedad y la cantidad de las vulnerabilidades en

el software desarrollado, Microsoft se enorgullece de haber incluido actividades ligadas a la seguridad y la privacidad en todas las fases del proceso. SDL se encuentra creado con base en 3 conceptos básicos [2]:

1. Educación
2. Mejora continua de los procesos
3. Responsabilidad

SDL ha sido diseñado para ser adoptado de manera independiente a la metodología que las organizaciones utilizan para el desarrollo de software seguro de calidad, la base fundamental del proceso se encuentra construida sobre 5 fases consideradas por Microsoft como el Ciclo Tradicional de Desarrollo de Software (CTDS) (véase figura 1), dichas fases son las siguientes [1]:

1. Requisitos
2. Diseño
3. Implementación
4. Verificación
5. Lanzamiento y respuesta

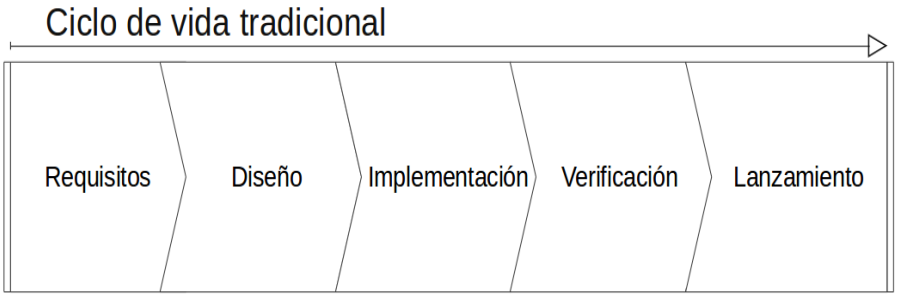


Fig. 1. CTDS según Microsoft.

A continuación son listadas las actividades propuestas por Microsoft para cada una de las etapas de SDL:

1. Actividades de la etapa de Entrenamiento en Seguridad.

   (a) *Entrenamiento sobre seguridad:* practica en la cual es necesario proveer el conocimiento adecuado e información reciente sobre tópicos de seguridad y privacidad a toda la organización, se debe garantizar que todos los miembros cuentan con los conocimientos necesarios. Los roles que se encuentran directamente relacionados con el desarrollo, es decir, desarrolladores, ingenieros de pruebas, administradores, entre otros, deben de terminar de manera exitosa por lo menos 1 curso o certificación cada año

[1].

A continuación se listan las temáticas y tópicos que el entrenamiento en seguridad deben cubrir:

i. Diseño seguro
   A. Reducción de superficie de ataque
   B. Defensa en lo profundo
   C. Principio de privilegios mínimos
   D. Incumplimiento de seguridad

ii. Modelo de amenazas
   A. Descripción de modelo de amenazas
   B. Implicaciones del diseño de un modelo de amenazas
   C. Restricciones de codificación en base a un modelo de amenazas

iii. Codificación segura
   A. Desbordamientos de búfer
   B. Errores aritméticos enteros
   C. Restricciones de codificación en base a un modelo de amenazas
   D. Cross-Site Scripting
   E. Inyección de Structured Query Language (SQL)
   F. Criptografía débil

iv. Pruebas de seguridad
   A. Diferencias entre pruebas de seguridad y pruebas funcionales
   B. Evaluación de riesgos
   C. Métodos de pruebas de seguridad

v. Intimidad
   A. Tipos de datos de privacidad sensible
   B. Mejores practicas de diseño de privacidad
   C. Evaluación de riesgos
   D. Mejores practicas de desarrollo de privacidad
   E. Mejores practicas de pruebas de privacidad

2. Actividades de la etapa de Requisitos.

   (a) *Establecimiento de requisitos de seguridad:* la fase optima para definir requisitos de seguridad es las fase mas temprana del CVDS pues se tiene mayor tiempo para identificar hitos y resultados clave, así si mismo se permite una mayor seguridad y privacidad que ayuda a mantener en control las actividades del calendario, se debe realizar una identificación temprana del proyecto y verificar que cumple con alguna de las siguientes características [1]:

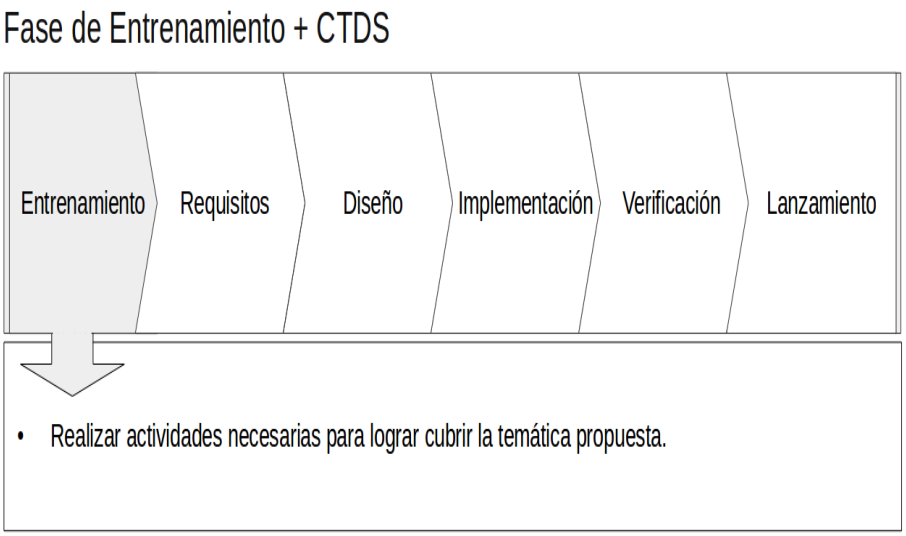


Fig. 2. Fase de entrenamiento añadida al CTDS.

i. ¿El producto que se deberá desarrollar sera sobre algún negocio?
ii. ¿El producto procesara información confidencial?
iii. ¿El producto contendrá componentes atractivos para menores de edad?
iv. ¿El producto realizara accesos a Internet?
v. ¿El producto realizara actualizaciones automáticas?

Una vez que se ha validado la viabilidad del proyecto sera necesario realizar la determinación de requisitos en base al anterior cuestionario, es decir se deben tomar como premisas claves las respuestas obtenidas con el fin de obtener un correcto listado de los requisitos de seguridad. Se deberá identificar al asesor de seguridad quien fungirá como un primer soporte y quien sera el responsable de definir las políticas globales de seguridad, ademas deberá permanecer al pendiente de cualquier situación que pueda afectar la seguridad del producto. Como siguiente paso sera necesario identificar un ente (persona o equipo) responsable del seguimiento y gestión de la seguridad, su principal responsabilidad sera mantener comunicados e informados acerca del estatus de la seguridad del producto a todos los miembros que conforman el equipo de desarrollo. Después, el asesor de seguridad en conjunto con el equipo de seguridad dueño del producto y cada una de las disciplinas involucradas deberán definir los suficientes requisitos de seguridad para garantizar que el producto funcione adecuadamente en el ambiente operacional requerido, una vez realizado lo anterior se deberá crear un plan adecuado para categorizar

y administrar los errores de seguridad relacionados al producto [1].

(b) *Definir niveles de seguridad y barras de errores:* en esta actividad es necesario establecer los niveles de seguridad y privacidad mínimos que el producto deberá satisfacer, los beneficios de realizar esta actividad son, un mejor entendimiento de los riegos de la seguridad y la privacidad y proveer una base solida de referencia a la hora de depurar errores en la fase de desarrollo. Se deberían establecer niveles mínimos en cada etapa del CVDS y dichos niveles deberán ser aceptados por el asesor en seguridad y privacidad [1].

(c) *Realizar la evaluación de riesgos de seguridad:* todas aquellas organizaciones que se dediquen al desarrollo de Software deberán de realizar una evaluación de riesgos que incluya todas las posibles amenazas y vulnerabilidades, a continuación se listan los tipos de información que dichas evaluaciones deben contener [1]:

i. ¿Qué porcentaje del proyecto requiere un modelo de amenazas?
ii. ¿Qué porcentaje del proyecto requiere revisiones del diseño seguro?
iii. ¿Qué porcentaje del proyecto requiere examenes de penetración especializados y si los especialistas son externos?
iv. ¿Exiten requisitos de pruebas o análisis de seguridad adicionales que el asesor considere necesarios para mitigar los riesgos de seguridad?
v. ¿Cual es el alcance especifico de los requisitos de las Pruebas Fuzz?
vi. ¿Cual sera el impacto de la conformidad del producto? (Sera necesario que cada organización emplee su propio Framework para medir el impacto).

A continuación es presentada de manera gráfica la adición de las actividades de la etapa de requisitos al CTDS y a la fase de entrenamiento en seguridad (véase figura 3).

3. Actividades de la etapa de Diseño.

(a) *Establecer los requisitos de diseño seguro:* para establecer los requisitos de diseño seguro es necesario considerar las características de alto riesgo así como establecer técnicas seguras para la codificación, los resultados obtenidos de dichas actividades deben de ser documentados como requisitos de diseño seguro. Es necesario realizar comparativas entre las especificaciones de diseño en contra de las especificaciones funcionales, la especificaciones deben cumplir con la siguiente lista de actividades [1]:

i. Describir de manera precisa el uso previsto de una característica o función.
ii. Describir cómo implementar la característica o función de una manera segura.

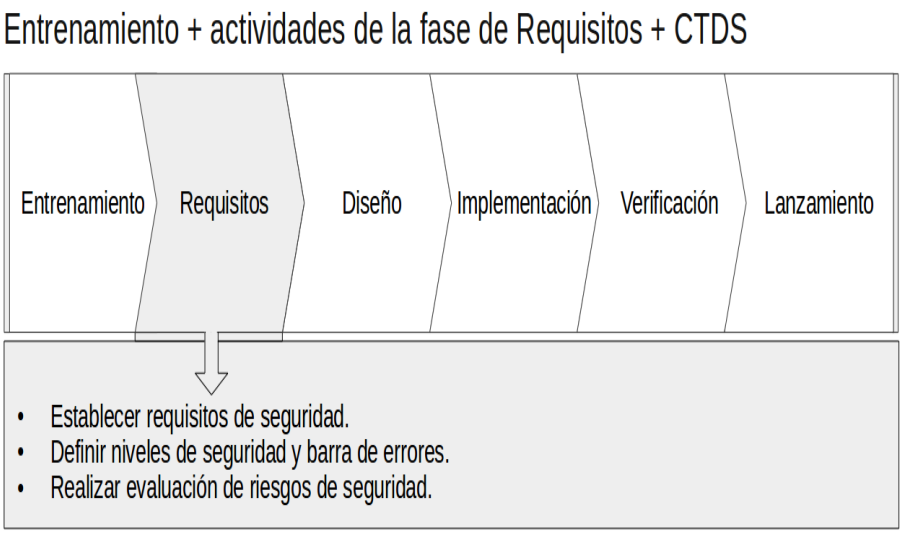


Fig. 3. Actividades de la fase de requisitos añadidas al CTDS y a la fase de entrenamiento.

iii. Describir si la característica o función tocarán datos de tarjetas de pago.

El control de cambios ofrece un registro archivado para los desarrolladores de aplicaciones pues permite revisar los cambios que los datos de impacto controlado como los datos de tarjetas de pago. La Criptografía es una pieza critica en la fase de diseño seguro pues se deben satisfacer de manera obligatoria los mínimos requisitos criptográficos establecidos, entre los cuales se encuentra [1]:

i. Usar Advanced Encryption Standard (AES) para el cifrado y descifrado simétrico.
ii. Usar 128 bits para mejorar las llaves simétricas.
iii. Usar RSA para el cifrado y descifrado asimétrico y firmas.
iv. Usar llaves RSA de 1024 bits o mejores.
v. Usar SHA-256 para Hashing y códigos de autenticación.

(b) *Análisis de la superficie de ataque:* en esta practica se debe realizar una reducción de la superficie de ataque, en otras palabras, realizar los trabajos para lograr la mínima probabilidad de ser vulnerado entre dichos trabajos se debe encontrar como mínimo los siguientes puntos:

i. Usar Code Access Security (CAS).
ii. Administrar las excepciones de Firewall cuidadosamente.

iii. Verificar que el producto funciona correctamente para usuarios sin permisos de administrador.

(c) *Completar el modelo de amenazas:* se debera utilizar el modelo de amenazas con aquellos componentes que fueron identificados como componentes sensibles durante la fase de evaluación de riesgos de seguridad. Poner el practica el modelo de amenazas permite al equipo de desarrollo mantener control sobre las implicaciones de seguridad dentro de un ambiente operacional. Los trabajos deben ser realizados con el equipo completo, dichos trabajos pueden ser listados de la siguiente manera [1]:

i. Completar el modelo de amenazas para los componentes sensibles o componentes que tienen riegos y los cuales fueron identificados en la fase de evaluación de riesgos de seguridad.

Los modelos de amenazas deben considerar las siguientes areas:

A. Todos los proyectos, todo el código que se exponen en la superficie de ataque así como el código que sea escrito por entidades externas al equipo de desarrollo.
B. Proyectos nuevos, todas las características y funcionalidades.
C. Versiones actualizadas, todas las características y funcionalidades agregadas.

ii. Verificar que el modelo de amenazas es el adecuado para satisfacer los requisitos mínimos de calidad.
iii. Verificar que el modelo de amenazas contiene la información necesaria, diagramas de flujo, activos, vulnerabilidades y formas de mitigacion.
iv. Realizar el modelo de amenazas utilizando STRIDE, técnica de descomposición de componentes.
v. Utilizar las herramientas propuestas por Microsoft en su sitio oficial.
vi. Asegurarse que los resultados de cada trabajo sea aprobado por un experto en seguridad.

La documentación generada en esta practica debe encontrarse almacenada y disponible para futuras revisiones en la fase de verificación.

A continuación es presentada de manera gráfica la adición de las actividades de la etapa de diseño al CTDS y a la fase de entrenamiento en seguridad (véase figura 4).

4. Actividades de la etapa de Implementación.

(a) *Utilizar herramientas aprobadas:* se deberá utilizar herramientas aprobadas por el asesor en seguridad, se debe mantener una lista publicada de las

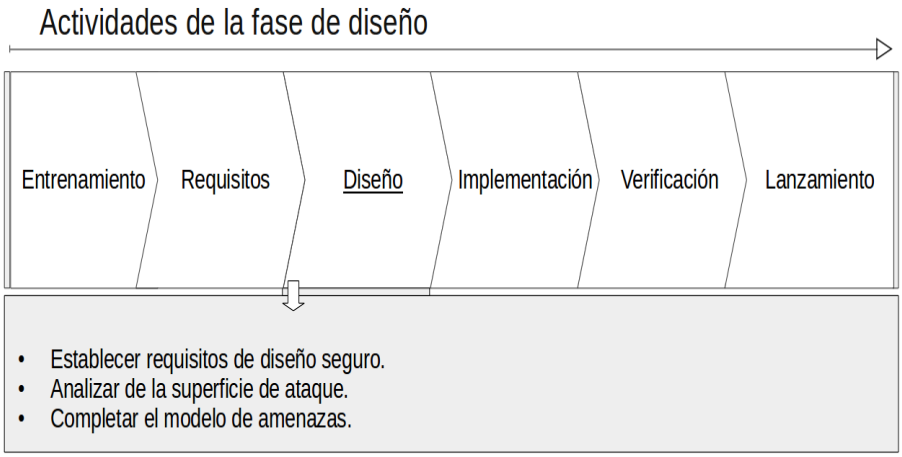


Fig. 4. Actividades de la fase de diseño añadidas al CTDS y a la fase de entrenamiento.

herramientas para que todos los miembros del equipo puedan consultarla en cualquier momento, se deberán utilizar las versiones mas resientes para aprovechar las nuevas funcionalidades y protecciones [1].

(b) *Depreciar funciones no seguras:* se deberá determinar cuales serán las funciones inseguras vetadas, el equipo deberá utilizar cabeceras para detectar cualquier función insegura vetada y compiladores nuevos así como herramientas de análisis de código automáticas que permitan identificar el uso de funciones inseguras [1].

(c) *Realizar análisis estáticos:* se deberá realizar un análisis estático a código fuente, el asesor de seguridad deberá evaluar el uso de herramientas automáticas para la revisión del código o revisiones humana que promuevan la remoción de vulnerabilidades [1].

A continuación es presentada de manera gráfica la adición de las actividades de la etapa de implementacion al CTDS y a la fase de entrenamiento en seguridad (véase figura 5).

5. Actividades de la etapa de Verificación.

(a) *Realizar análisis de código dinámico:* se deberán realizar verificaciones en tiempo de ejecución para verificar que el funcionamiento es el diseñado, se deberá especificar que tipo de herramientas podrán utilizarse para el monitoreo del comportamiento del producto [1].

(b) *Realizar Pruebas Fuzz:* estas son realizadas por medio de la inserción de información mal formada o al azar con el fin de producir errores en el

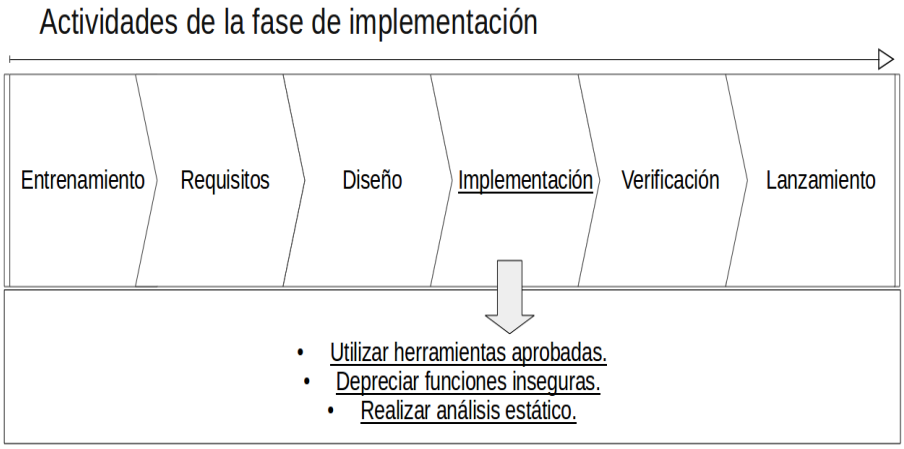


Fig. 5. Actividades de la fase de implementacion añadidas al CTDS y a la fase de entrenamiento.

producto [1].

(c) *Realizar una revisión de la superficie de ataque:* para evitar vulnerabilidades producidas por la desviación funcional del producto y el diseño del mismo se deberá realizar revisiones al modelo de amenazas y a la superficie de ataque [1].

A continuación es presentada de manera gráfica la adición de las actividades de la etapa de implementación al CTDS y a la fase de entrenamiento en seguridad (véase figura 6).

6. Actividades de la etapa de Lanzamiento.

(a) *Crear un plan de respuesta a incidentes:* debido a la incertidumbre sobre futuras vulnerabilidades emergentes el plan de respuesta deberá incluir los siguientes puntos [1]:

i. Una lista de la información de contacto de quien servirá como primer contacto ante una emergencia.
ii. Información de contacto de autoridades para la toma de decisiones 24 horas al día 7 días a la semana.
iii. Planes de servicio de seguridad (procedimientos de escalamiento) de código heredado de otros grupos dentro de la organización.
iv. Planes de servicio de seguridad (procedimientos de escalamiento) de código de terceros con licencia, incluyendo los nombres de archivos, versiones, código fuente, información de contacto con otros provee-

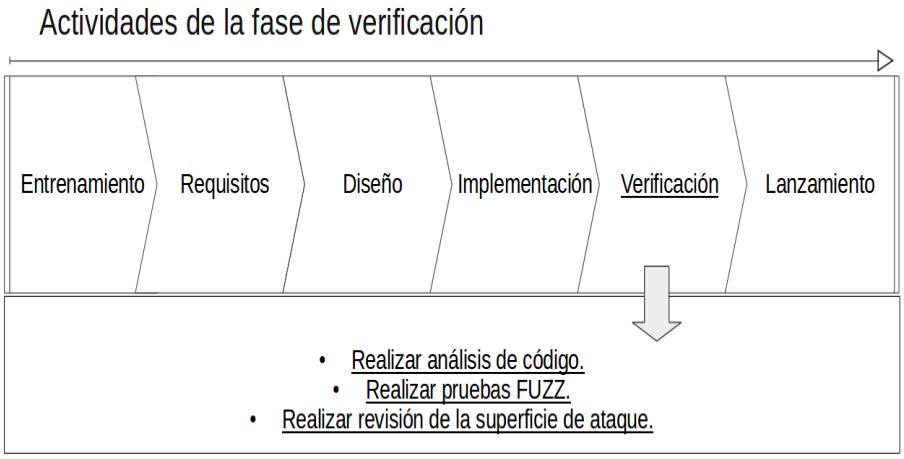


Fig. 6. Actividades de la fase de verificación añadidas al CTDS y a la fase de entrenamiento.

dores, y autorización contractual para realizar cambios (en su caso).

(b) *Realizar un revisión final de la seguridad:* no es un examen de penetración, no es una forma de mitigar problemas de seguridad olvidados, la revisión final incluye examinar el modelo de amenazas, las solicitudes de excepción, las salidas de la herramientas y el desempeño contra los niveles de calidad previamente determinadas [1].

Existen dos tipos de revisiones finales de la seguridad y pueden ser listados de la siguiente manera:

i. Pasada: todos los problemas de seguridad identificados han sido arreglados.
ii. Pasada con excepciones: todos los problemas de seguridad identificados han sido arreglados y las escepciones resueltas, aquellas que no puedan abordarse deran ser arregladas en la siguiente version del producto, si existe alguna escepcion esta tiene que ser revisada por el asesor de seguridad, el asesor tiene que llegar a algún compromiso con el dueño del producto y si no lo logra el producto no deberá ser liberado.

(c) *Archivar la información:* para mejorar la velocidad y la calidad de respuesta durante un incidente se deberá incluir los siguientes puntos:

i. Especificación de características.
ii. Código fuente, binarios y símbolos privados.
iii. modelos de amenazas.

iv. Casos de prueba.
v. Documentación relacionada.
vi. Planes de respuesta.
vii. Licencias y términos de servicios para cualquier producto de terceros.

A continuación es presentada de manera gráfica la adición de las actividades de la etapa de liberación al CTDS y a la fase de entrenamiento en seguridad (véase figura 7).

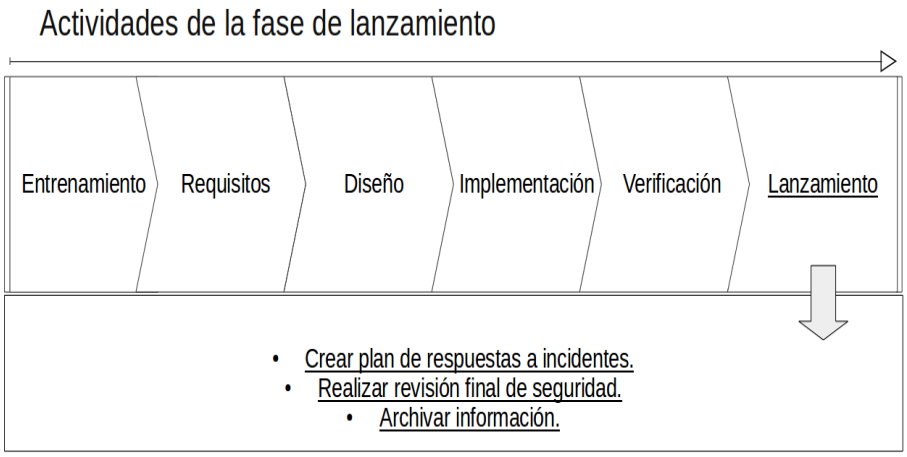


Fig. 7. Actividades de la fase de lanzamiento añadidas al CTDS y a la fase de entrenamiento.

7. Actividades de la etapa de Respuesta.

   (a) *Ejecutar plan de respuesta a incidentes:* se deberán poner en practica las actividades planeadas con anterioridad como respuesta a cualquier ataque, sospecha o amenaza (véase figura 8) [2].

SDL fue creado como un proceso del cual cualquiera puede disponer, su principal meta es mejorar de manera drástica la seguridad y la privacidad de la información contenida en los productos desarrollados bajo su dirección, SDL posee acciones obligatorias sin embargo es posible integrar en su proceso actividades o directivas privadas con el fin de construir una metodología única para cada organización. Las aportaciones que la nueva metodología deberá combinar son, los procesos, la formación y las herramientas especializadas, que en conjunto deberán producir mayor previsibilidad y capacidad técnica así como un producto mas seguro lo cual se vera reflejado en un menor riesgo para la organización y el usuario final del producto.

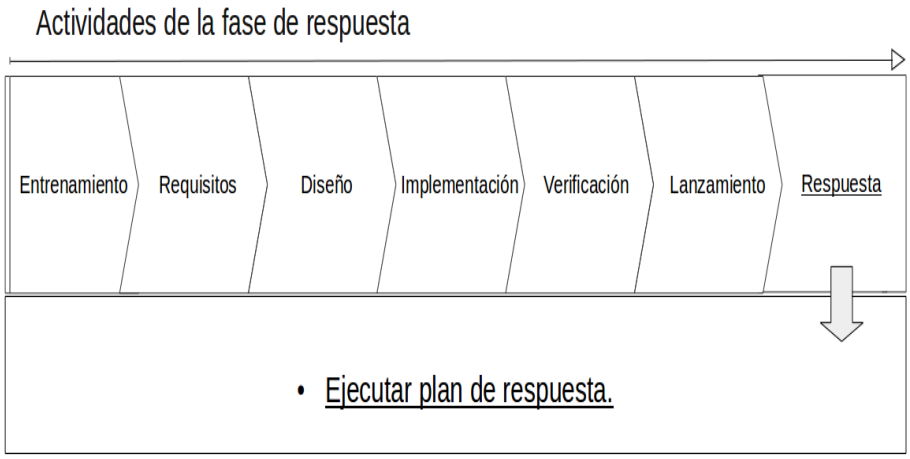


Fig. 8. Actividades de la fase de respuesta añadidas al CTDS y a la fase de entrenamiento.

¿Por qué adoptar SDL? Internet, o como algunos la llamarían *The World Wide Web* se ha convertido en el mayor medio de comunicación, pues ha logrado congregar a centenas de miles de usuarios que conectados entre si comparten e intercambian información al mismo tiempo [3]. En la actualidad existe una demanda continua de nuevos medios seguros de comunicación a través de la red, debido a ello los productores de Software requieren de innovadoras técnicas que garanticen la producción de Software seguro y ademas de ello contengan características adecuadas para una rápida y correcta adopción lo cual se deberá ver reflejado en el aspecto económico de la empresa.

SDL posee características únicas que pueden ser aprovechadas para lograr una correcta y rápida inserción en el trabajo, algunas de importancia son listadas a continuación [3]:

1. SDL es definido como sus creadores como un proceso que puede ser adaptado a cualquier técnica utilizada para desarrollar Software de calidad.
2. SDL esta diseñado para cualquier tamaño de empresa.
3. SDL presenta una documentación Online en varios idiomas.
4. Cada tópico de importancia es abordado a detalle.
5. SDL promueve el cumplimiento de los estándares ISO/IEC 27034-1 y ISO/IEC 27034-1:2011.

¿Cómo adoptar SDL? Para lograr una correcta adopción de SDL es necesario seguir de manera puntual 7 pasos o etapas que han sido establecidos por Microsoft pues se ha logrado recopilar información valiosa a lo largo de varios años de practica dentro de los sectores financieros de alto riesgo. A continuación son presentadas dichas etapas [1]:

1. Educacion
2. Personalización
3. Conocimiento de herramientas e infraestructura
4. Entrenamiento
5. Puesta en marcha
6. Medición
7. Revisión

La fase de educación es sumamente importante, en su mayoría los expertos reconocen que la correcta comprensión de la información acerca de SDL por parte de los equipos de desarrollo es la llave para la correcta adopción pues es necesario comprender de manera detallada como es que cada uno de los puntos de inserción embonan con las actividades y esfuerzos realizados cotidianamente dentro de la organización, es decir, en esta etapa es necesario asegurarse que dentro de la organización existe un conocimiento pleno sobre SDL.

La fase de personalización es utilizada para lograr la correcta estructuración y conexión entre el ciclo de vida de desarrollo de Software cotidiano y SDL, es decir, se debe comprender de manera detalla cuales son las actividades o fases que serán incluidas en el CVDS utilizado en la organización pues SDL esta diseñado para ajustarse y moldearse a conveniencia de los interesados [1].

La fase de conocimiento de herramientas e infraestructura hace énfasis en establecer normativas de uso y disposición de herramientas enfocadas en mejorar el proceso de adopción de SDL, el equipo deberá conocer a fondo las herramientas que serán utilizadas ademas de establecer un correcto entorno de desarrollo. Microsoft ofrece una amplia gama de herramientas libre de cargo que están enfocadas en mejorar la correcta inserción y uso de SDL y las cuales pueden ser encontradas Online en el siguiente Uniform Resource Locator (URL): `http://www.microsoft.com/en-us/sdl/adopt/tools.aspx`.

La fase de entrenamiento considera necesario realizar una introducción informativa general a todos los miembros de la organización con el fin de promover el conocimiento de las actividades que serán puestas en practica, se deberá considerar que sera necesario realizar sesiones de entrenamiento para cada tipo de rol o para aquellos roles que tengan responsabilidades de alta jerarquía. Los expertos consideran que los entrenamientos específicos deberán ser llevados acabo una vez que los miembros de la organización ya estan utilizando SDL [1].

La fase de puesta en marcha debe ser monitoreada constantemente pues es aquí cuando la mayoría de las problemáticas surgen y es cuando sera necesario evaluar las nuevas acciones realizadas con el fin de realizar modificaciones o adaptaciones que resulten en mejoras continuas al proceso [1].

La fase de medición esta dedicada en analizar de manera detallada los resultados obtenidos de dos principales practicas y las cuales son mencionadas a continuación:

1. Revisión final de seguridad
2. Postmortem

El cometido principal de estas practicas es eliminar cualquier tipo de defecto en la seguridad de los productos antes de llegar al cliente, todo esto por medio de comparativas con modelos de riesgos, administración de historiales de defectos entre otros. De esta manera es posible tener una base contable sobre los resultados obtenidos a través del tiempo promoviendo así la mejora continua del proceso.

En la fase de revisión se deberán llevar a cabo análisis formales de los datos recavados en la fase anterior con el principal objetivo de promover mejoras del proceso por medio de modificaciones a las actividades, a dichas revisiones deberán asistir todos los involucrados [1].

¿Cómo evaluar el impacto de SDL? Para lograr la inserción de conceptos o modificaciones a los procesos y lograr productos mas seguros es necesario realizar la acciones correspondientes, las variables que juegan un papel importante para lograr escalar dentro de los niveles de madurez de cualquier modelo son el tamaño de la organización, los recursos como el tiempo, talento y presupuestos además claro del respaldo de los directivos, para obtener un correcto control de los impacto intangibles es necesario lograr una comprensión detallada de los elementos que constituyen los procedimientos de desarrollo de seguridad y establecer normas de implementación según el nivel de madurez que el equipo de desarrollo posea, SDL posee su propio modelo de optimización el cual ayuda en todas las cuestiones anteriormente mencionadas. El modelo esta constituido por 5 áreas de capacidades las cuales corresponden con el CVDS y las cuales pueden ser mencionadas de la siguiente manera [4]:

1. Formación, políticas y capacidades organizativas
2. Requisitos y diseño
3. Implementación
4. Comprobación
5. Lanzamiento y respuesta

A continuación son listados los niveles de madurez utilizados por el modelo SDL:

1. Básico
2. Dinámico
3. Estandarizado
4. Avanzado

El modelo comienza en nivel básico en el cual la organización posee pocos procesos, cursos de formación y herramientas, este nivel tiene como cometido poder llevar a la organización al nivel dinámico en el cual la misma poseerá

procesos eficaces, personal altamente cualificado, herramientas especializas y un alto grado de responsabilidad por parte de los involucrados internos y externos, en este orden se pretende llegar al nivel superior conocido como nivel avanzado (véase figura 9).

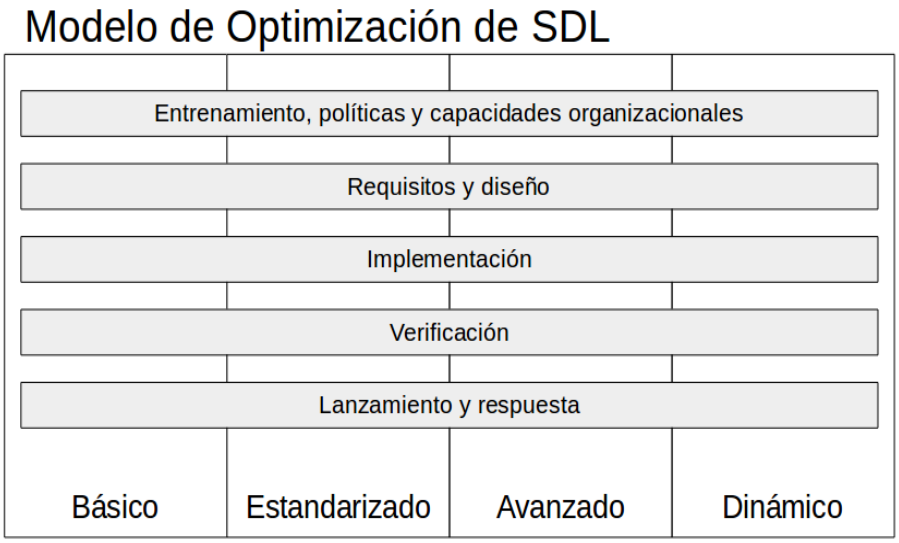


Fig. 9. Modelo de optimización utilizado por SDL.

¿Quiénes deben poner en practica SDL? Microsoft propone una lista de características que los proyectos deben tener para ser candidatos ideoneos seguir un proceso SDL.

1. Aplicaciones que procesan datos de identificación personal o cualquier otro tipo de información confidencial o privilegiada.
2. Aplicaciones que se comunican o realizan envíos de información frecuentemente a través de Internet u otros redes.
3. Aplicaciones implementadas en entornos empresariales.

¿Cuáles son las cualidades de SDL? Las cualidades que SDL posee según el estudio realizado por Ikram El rhaffari y Ounsa Roudies y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la Seguridad Informática (SI) y a las Tecnologías de la Información (TI) y son listadas a continuación [5].

1. Cualidades relacionadas a la Ingeniería de Software.

   (a) El producto final tendrá varios enfoques de calidad.
   (b) El mapeo al CTDS es facil.
   (c) Se alinea fácilmente con otros métodos de Ingeniería de Software.
   (d) La información para la implementación y los recursos necesarios se encuentran Online.
   (e) Posee su propia guía para el desarrollo de actividades.

2. Cualidades relacionadas a la Seguridad Informática.
   (a) Posee un enfoque basado en promover la seguridad para alcanzar la calidad.
   (b) Implica y compromete a un equipo de seguridad y promueve la interacción entre los actores.
   (c) Posee un bajo indice de cumplimiento y privacidad.

3. Cualidades relacionadas con TI.
   (a) Se enfoca en promover la calidad.
   (b) Su documentación posee un alto indice de completitud y se encuentra disponible de manera Online.
   (c) Provee una completa cobertura organizacional.
   (d) Es evolutivo.
   (e) Se adapta fácilmente a diferentes tipos de estructuras organizacionales.
   (f) Posee su propio modelo de madurez.
   (g) Provee soporte para el aseguramiento de la calidad de los servicios.

El proceso promueve la mejora continua en las organizaciones ayudándolas alcanzar la certificación ISO/IEC 27034-1 la cual garantiza que la organización mantiene procesos que involucran la aplicación de controles, medidas y mediciones con el fin de gestionar el riesgo de uso de los productos desarrollados [1].

¿Cuáles son las deficiencias que presenta SDL? Las deficiencias que SDL posee según el estudio realizado por Ikram El rhaffari y Ounsa Roudies y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación [5].

1. Deficiencias relacionadas a la Ingeniería de Software.
   (a) No es Flexible y es riguroso.

2. Deficiencias relacionadas a la Seguridad Informática.
   (a) No posee métricas de seguridad.

3. Deficiencias relacionadas con Information Technologies (IT), no presenta deficiencias de acuerdo al estudio de Ikram El rhaffari y Ounsa Roudies [5].

4.2 Proceso de Seguridad en Aplicación Completo y Ligero

Comprehensive, Lightweight Application Security Process (CLASP, Proceso de Seguridad en Aplicación Completo y Ligero), es un proyecto creado como parte de The Open Web Application Security Project (OWASP) bajo la supervision de Pravir Chandra en colaboración de Jeremy Feragamo, Dan Graham, John Viega, Jeff Williams y Alex Newman. Comprehensive, Lightweight Application Security Process (CLASP) es un recurso que se encuentra bajo licencia Open Source, sus responsables hacen una invitación abierta a todos los profesionales relacionados a la seguridad informática a realizar aportaciones y revisiones del material existente por medio de la inscripción a una membresia de bajo coste [6].

¿Qué es CLASP? CLASP es definido como un conjunto de piezas de proceso que pueden ser adoptadas para trabajar en conjunto con ciclos de vida de desarrollo de Software definidos. La meta principal que persigue CLASP es proveer de una estructura fiable para la depuración de problemáticas relacionadas a la seguridad en fases tempranas del CVDS [6].

CLASP esta formado por una gran cantidad de recursos extraídos de Ciclos de Vida de Desarrollo de Software los cuales metódicamente fueron descompuestos para crear un conjunto de requisitos de seguridad que son la base de las mejores practicas de CLASP y las cuales permiten a las organizaciones a solucionar vulnerabilidades de manera sistemática. Las actividades para la mejora de la seguridad de CLASP fueron diseñadas para ser integradas facilmente con otros procesos de desarrollo existentes, cada una de estas actividades deberá ser asignada a uno o mas roles, CLASP provee una guía para los participantes del proceso. CLASP provee un Léxico de Vulnerabilidades (LV) propio el cual esta diseñado para ayudar a los equipos de desarrollo a abordar y remediar errores específicos de diseño o codificación que puedan ser explotados [7].

La estructura de componentes CLASP y sus dependencias se organizan de la siguiente manera:

1. Vistas CLASP.

   Existen 5 tipos de vistas que son consideradas perspectivas de alto nivel, cada vista es dividida en componentes de proceso por medio de una organización del tipo (Vista - Actividad - Componente de Proceso).

   Las vistas CLASP son listadas a continuación.

   (a) Vista de Conceptos (VC)
   (b) Vista Basada en Roles (VBR)
   (c) Vista Basada en Evaluación de Actividades (VBEA)
   (d) Vista Basada en la Implementación de Actividades (VBIA)

(e) Vista de Vulnerabilidades (VV)

Para lograr obtener una idea clara sobre las interacciones entre componentes es necesario visualizar el diagrama propuesto por OWASP (Véase figura 10).

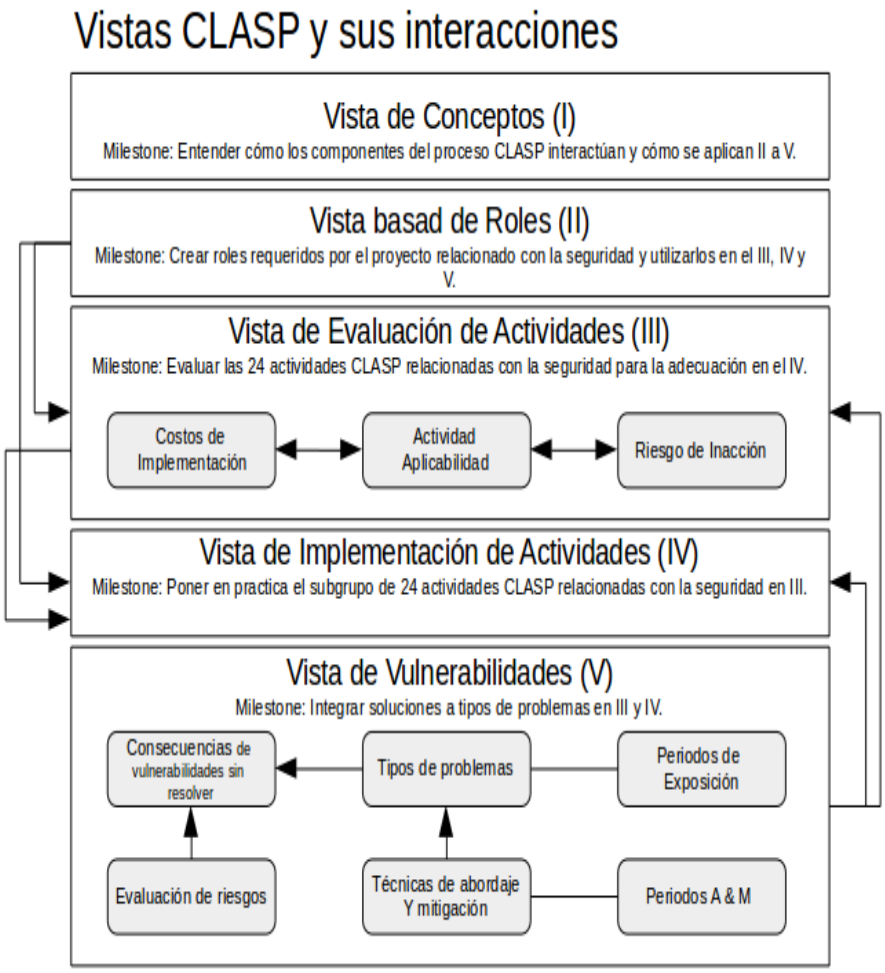


Fig. 10. Vistas CLASP y sus interacciones.

2. Recursos CLASP.

Los recursos propuestos por CLASP proveen artefactos útiles cuando en el proyecto que se debe desarrollar se utilizan herramientas que automatizan

las piezas del proceso CLASP [7].

A continuación se muestra de manera gráfica el nombre y la localización de los recursos CLASP asi como las vistas que cada uno de ellos soporta (véase figura 11).

Tabla de recursos y localización

Recurso CLASP	Localización del recurso
Principios básicos de seguridad de las aplicaciones (todas las vistas)	A
Ejemplo del Principio Básico: validación de entrada (todas las vistas)	B
Ejemplo de violación de principio básico: modelo Penetrate-And-Patch (todas las vistas)	C
Servicios de seguridad de núcleo (todas las vistas, en especial la III)	D
Ejemplo de hojas de trabajo guías de codificación (vistas II, III y IV)	E
Sistema de evaluación hojas de trabajo (vistas III y IV)	F
Ejemplo de hoja de ruta: proyectos heredados (vista III)	G1
Ejemplo de hoja de ruta: proyectos nuevos (vista III)	G2
Creación del plan de Ingeniería de Procesos (vista III)	H
Creación del equipo de Ingeniería de Procesos (vista III)	I
Glosario de términos de seguridad (todas las vistas)	J

Fig. 11. Recursos CLASP y su localización

3. Casos de Uso de Vulnerabilidades (CUV).

   Los CUV describen las condiciones bajo las cuales los servicios de seguridad pueden ser vulnerados, ademas proporcionan a los usuarios ejemplos de las relaciones causa y efecto entre seguridad / diseño y código fuente, así como las vulnerabilidades resultantes en los servicios básicos de seguridad es decir., autenticación, autorización, confidencialidad, disponibilidad, responsabilidad, y no repudio [7].

   Los CUV están basados en la siguiente lista de arquitecturas de componentes.

   (a) Monolithic UNIX
   (b) Monolithic Mainframe
   (c) Arquitectura distribuida

   Los CUV deben ser utilizados como un puente entre la VC y el LV [7].

4. Mejores practicas.

   CLASP posee 7 practicas enfocadas en la mejora de la aplicación de la seguridad, estas son:

   (a) Realizar programas de sencibilización
   (b) Realizar evaluaciones
   (c) Capturar requisitos de seguridad
   (d) Construir procedimientos de remediación a vulnerabilidades
   (e) Definir y monitorear métricas
   (f) Publicar reglas de seguridad operativa

Una vista de alto nivel al orden de ejecución de actividades relacionadas a las políticas de seguridad puede promover el aumento de la conciencia sobre la aplicación de actividades para la mejora de la seguridad. El orden ascendente de ejecución de actividades en la organización puede ser listado de la siguiente manera [7]:

1. Mejores practicas para la aplicación de la seguridad
2. Políticas para la aplicación de la seguridad
3. Políticas de seguridad de IT
4. Operación de políticas de seguridad
5. Políticas de seguridad corporativas

CLASP define Vulnerabilidad de Seguridad (VS) como una debilidad en un entorno de Software y define Consecuencia como un fallo en los siguientes servicios básicos de seguridad:

1. Autorización (control de acceso a los recursos)
2. Confidencialidad (en los datos y otros recursos)
3. Autenticación (establecimiento de identidad e integridad)
4. Disponibilidad (denegación de servicios)
5. Responsabilidad
6. No repudio

CLASP posee su propia Taxonomia la cual se describe como una clasificación de alto nivel que se divide en clases para una mejorar la evaluación y resolución de las Vulnerabilidades de Seguridad del código fuente [7].

Las clases en las que se divide son las siguientes.

1. Tipos de problemas

2. Categorías
3. Periodos de exposición
4. Consecuencias
5. Plataformas
6. Recursos
7. Evaluaciones de riesgos
8. Forma de abordaje y periodos de mitigación

¿Porqué adoptar CLASP? Entre los principales atractivos que CLASP se encuentran sus principios básicos los cuales están enfocados en las determinaciones de Free and Open-Source Software (FOSS) y los cuales pueden ser listados de la siguiente manera [7]:

1. Es libre y abierto.
2. Es gobernado por consenso áspero y código funcional.
3. Obliga a cumplir con un código de ética.
4. No tiene fines de lucro.
5. No es impulsado por intereses comerciales.
6. Posee un enfoque basado en el riesgo.

¿Cómo adoptar CLASP? CLASP ha sido concebido como un conjunto de piezas de proceso que pueden ser adoptadas por cualquier compañía e insertadas en cualquier CVDS definido, CLASP en su VBEA menciona que debido a la gran cantidad de piezas existentes ciertas organizaciones podrían creer que los esfuerzos necesarios para lograr una correcta inserción de las mismas podría acarrear inversiones peligrosas, sin embargo tal y como se menciona en el sitio oficial de OWASP, "No es necesario adoptar todas las piezas de CLASP para obtener mejoras en la seguridad" [8].

Como pilar fundamental para la adopción de CLASP es necesario que los involucrados en el desarrollo de los nuevos productos conozcan a fondo la VC pues en ella es posible encontrar la descripción de la estructura principal de CLASP, dicha vista es descrita como la introducción a los conceptos detrás de la correcta adopción de las piezas del proceso, en ella sera posible encontrar ejemplos sobre posibles caminos de adopción así como información relacionada sobre como CLASP puede ayudar como el aseguramiento de la integridad de los datos [6].

Una vez que se ha logrado el entendimiento de la VC sera necesario continuar con el proceso de adopción por medio de la comprensión de la siguiente información proveída por CLASP [6].

1. Las 7 practicas que definen CLASP.
2. Resumen de los servicios de seguridad de alto nivel.
3. Núcleo de principios de seguridad para el desarrollo de Software.

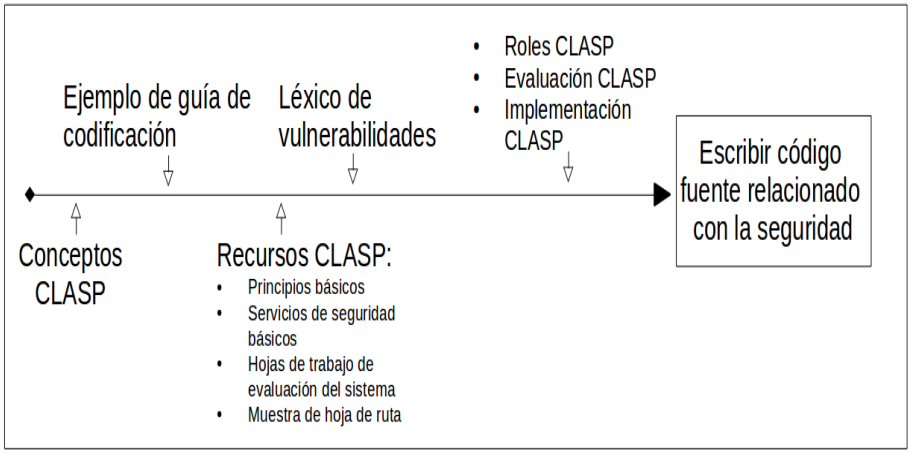


Fig. 12. Posible secuencia al aplicar los componentes CLASP en un CVDS definido.

4. Roles que son incluidos en el desarrollo de Software.
5. Colección de actividades para construir Software mas seguro.
6. Asesoría del proceso de ingeniería CLASP y mapas de ruta.
7. Lista de guías de codificación para ayudar a los desarrolladores y auditores con las revisiones de código.
8. Léxico de vulnerabilidades.
9. Glosario de términos y frases comunes para la aplicación de la seguridad.
10. Lista de vulnerabilidades.

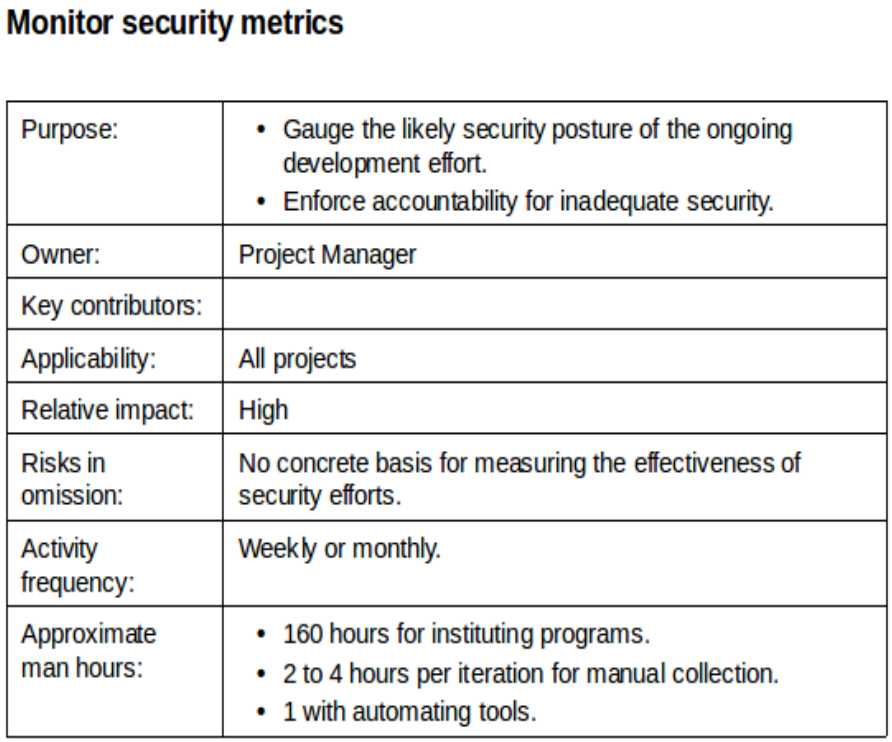
Ademas del correcto entendimiento de la información proveída por CLASP en la lista anterior sera necesario que los miembros del equipo conozcan a fondo el contenido de las 4 vistas restantes con el fin de generar una alto grado de eficacia a la hora de la total adopción de las piezas CLASP.

¿Cómo evaluar CLASP? CLASP en su VBEA provee la información necesaria para realizar evaluaciones sobre el desempeño de las actividades realizadas con el fin de mejorar la seguridad de los productos, en particular esta vista provee la siguiente información acerca de cada nueva actividad relacionada a las evaluaciones [8]:

1. Propósito de la actividad
2. Responsable de la actividad
3. Contribuidores
4. Aplicabilidad
5. Impacto relativo
6. Riesgos por omisión
7. Frecuencia de la actividad

8. Horas hombre aproximadas para la actividad

A continuación se presenta el formulario para el monitoreo de métricas propuesto en la VBEA (véase figura 13).



Monitor security metrics

Purpose:	• Gauge the likely security posture of the ongoing development effort. • Enforce accountability for inadequate security.
Owner:	Project Manager
Key contributors:	
Applicability:	All projects
Relative impact:	High
Risks in omission:	No concrete basis for measuring the effectiveness of security efforts.
Activity frequency:	Weekly or monthly.
Approximate man hours:	• 160 hours for instituting programs. • 2 to 4 hours per iteration for manual collection. • 1 with automating tools.

Fig. 13. Formulario para el monitoreo de métricas contenido propuesto en la VBEA.

Como soporte para al monitoreo de métricas OWASP provee una sección en la cual es posible encontrar información relacionada con los siguientes aspectos.

1. ¿Cómo identificar las métricas que deben de ser recolectadas?
2. ¿Cómo identificar como serán usadas las métricas recolectadas?
3. ¿Cómo recopilar datos?
4. ¿Cómo realizar una estrategia para la presentación de informes?
5. ¿Cómo recopilar y evaluar las métricas?

Por otra parte OWASP propone el uso de Software Assurance Maturity Model (SAMM) como modelo de madurez el cual es definido como un marco abierto

para ayudar a las organizaciones a formular y poner en práctica una estrategia para la seguridad del Software que se adapta a los riesgos específicos que enfrenta la organización [9].

¿Quiénes deben poner en practica CLASP? OWASP provee una lista de entidades que en base a la experiencia obtenida a lo largo de varios años considera que deberían adoptar sus proyectos de seguridad entre ellos CLASP [10].

A continuación son listadas dichas entidades.

1. Desarrolladores de aplicaciones.
2. Arquitectos de Software.
3. Autores e ingenieros de seguridad informática.
4. Aquellos que quieran el apoyo de una comunidad profesional mundial para desarrollar o probar una idea.
5. Cualquier persona que desee hacer uso de la organización profesional de los conocimientos OWASP.

¿Cuáles son las cualidades de CLASP? Las cualidades que CLASP posee según el estudio realizado por Ikram El rhaffari y Ounsa Roudies y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación [5].

1. Cualidades relacionadas a la Ingeniería de Software.
   (a) El producto final tendrá enfoque en la seguridad.
   (b) La información para la implementación y los recursos necesarios se encuentran Online.
   (c) Es flexible.
   (d) Posee su propia guía para el desarrollo de actividades.

2. Cualidades relacionadas a la Seguridad Informática.
   (a) Posee un completo enfoque en la seguridad.
   (b) Implica y compromete a un equipo de seguridad.
   (c) Se implementa a través del CVDS.
   (d) Provee la información necesaria para el monitoreo y creación de métricas para la evaluación.

3. Cualidades relacionadas con IT.
   (a) Su documentación posee un alto indice de completitud y se encuentra disponible de manera Online.
   (b) Provee una completa cobertura organizacional.
   (c) Se adapta fácilmente a pequeñas organizaciones.

¿Cuáles son las deficiencias que presenta CLASP? Las deficiencias que CLASP posee según el estudio realizado por Ikram El rhaffari y Ounsa Roudies y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación [5].

1. Deficiencias relacionadas a la Ingeniería de Software.
   (a) El mapeo al CTDS es complejo.
   (b) No se alinea fácilmente con otros métodos de Ingeniería de Software.

2. Deficiencias relacionadas a la Seguridad Informática.
   (a) Posee un bajo indice de cumplimiento y privacidad.

3. Deficiencias relacionadas con TI.
   (a) No se enfoca en la calidad de otros aspectos del producto.
   (b) Evoluciona lentamente.
   (c) No posee su propio modelo de madurez.
   (d) Provee bajo soporte para el aseguramiento de la calidad de los servicios.

4.3 Metodo de Corrección por Construcción

Correctness by Construction (CbyC, Corrección por Construcción), es un método radical que fusiona características de los Métodos Tradicionales y los Métodos Ágiles. Praxis ha puesto en practica Correctness by Construction (CbyC) a lo largo de 12 años y a logrado obtener excelentes estadísticas de productividad, 0.05 defectos por cada 1,000 líneas de código y un aumento de alrededor de 30 líneas de código promedio por día por persona [11].

¿Qué es CbyC? CbyC es un método que se encuentra consolidado sobre 3 principios básicos que forman su eje principal, CbyC posee características que permiten mejorar la calidad de la seguridad de los productos desarrollado y sus principios básicos son listados a continuación.

1. Crear productos en los cuales sea muy difícil de introducir errores.
2. Asegurar la eliminiación de los errores lo más pronto posible del punto de su introducción.
3. Generar evidencia de aptitud para el propósito durante todo el desarrollo como un subproducto natural del proceso.

En contraste con el método Correctness by Debuging (CbyD) que sigue siendo la forma en que la mayoría del Software se desarrolla en la actualidad, CbyC busca producir un producto correcto desde el inicio de su producción. La fase de pruebas se convierte en la demostración de sus características funcionales y no el punto de partida para la depuración pues se debe asegurar que los productos desarrollados integren las propiedades requeridas en lugar de sólo realizar pruebas y esperar por el defecto. CbyC es un método técnico para el desarrollo de

Software altamente compatible con los principios de Personal Software Process (PSP) y Team Software Process (TSP). Evidencia tentativa de pequeña escala muestra que CbyC combinado con PSP y TSP puede resultar en tasas de defectos aún más bajas. El enfoque técnico de CbyC puede complementar PSP para ofrecer altos indices de seguridad [11].

CbyC posee 8 etapas o fases que poseen características propias de los Métodos Tradicionales y los Métodos Ágiles (véase figura 14) [12].

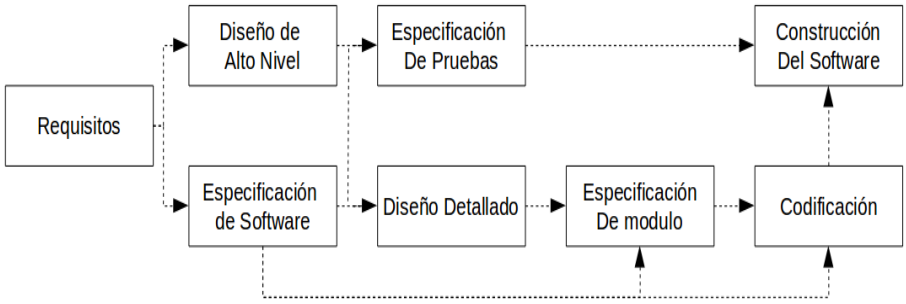


Fig. 14. Fases y secuencia de CbyC.

1. Fase de Requisitos.

   En esta fase se debe especificar el propósito del desarrollo, los requisitos no funcionales y las funciones a desarrollar, se debe tomar en cuenta el diagramado por medio de técnicas y herramientas formales como Visual Paradigm y Unified Modeling Language (UML) [12].

2. Fase de Diseño de Alto Nivel.

   En esta fase es necesario describir la estructura interna del producto, se deben realizar actividades tales como: distribuir de manera balanceada las funcionalidades a desarrollar, definir la estructura de las base de datos, definir los mecanismos para transacciones y comunicaciones así como priorizar los requisitos de protección y seguridad [12].

3. Fase de Especificación del Software.

   En esta fase se debe realizar la especificación de la interfaz del usuario, las especificaciones de los niveles superiores y desarrollar un prototipo para la validación [12].

4. Fase de Diseño Detallado.

   En esta fase se debe definir los módulos, procesos y cada funcionalidad respectivamente, se deben tomar en cuenta el diagramado por medio de técnicas y herramientas formales como Visual Paradigm y UML [12].

5. Fase de Especificación de Módulos.

   En esta fase se deben definir el estado y comportamiento de cada módulo teniendo un enfoque de bajo acoplamiento y alta cohesión [12].

6. Fase de Codificación.

   En esta fase es necesario considerar el uso de un lenguaje que tenga características optimas para la comprobación matemática, en esta fase es necesario realizar pruebas estáticas y revisiones de código [12].

7. Fase de Especificación de Pruebas.

   En esta fase se debe considerar las Especificaciones de Software, los Requisitos y el Diseño de Alto Nivel para realizar pruebas de comportamiento y pruebas que cubran los requisitos no funcionales. [12].

8. Fase de Construcción del Software.

   Considerando que CbyC utiliza técnicas de Métodos Ágiles, la primera entrega debe contener un producto con todas las interfaces y mecanismos de comunicación, la funcionalidad del producto se debe incrementar conforme a cada interacción [12].

CbyC depende del conocimiento exacto sobre lo que el producto debe o no hacer y cuales características son prioritarias debido a esto se debe tener muy en claro que tipo de producto se esta desarrollando, si algún miembro del equipo de desarrollo lo desconoce esto podría ser causa de la inserción de defectos inesperados [13].

CbyC incluye actividades de carácter genérico y entre las cuales se encuentran las siguientes.

1. Planificación de procesos.
2. Capacitación del personal.
3. Trazabilidad de los requisitos a traves de la especificación de código y casos de prueba.
4. Gestión de fallos.
5. Gestión de cambios.
6. Gestión de la configuración.
7. Recolección de métricas.

¿Porqué adoptar CbyC? CbyC combina notaciones matemáticamente rigurosas con enfoques ágiles para el desarrollo gradual; el resultado es que la industria obtendrá bajas tasas de defectos combinados con una alta productividad. La experiencia obtenida por Praxis indica que la mejora de la calidad y la seguridad de los productos es notoria así como la productividad de cada uno de los miembros del equipo de desarrollo [11].

¿Cómo adoptar CbyC? Para lograr la correcta adopción de CbyC es necesario que los miembros del equipo de desarrollo comprendan el flujo del proceso así como las características de cada etapa. Es importante mencionar que el dominio de Métodos Tradicionales de desarrollo de Software como PSP y TSP potenciaran la rápida adopción de CbyC pues este posee características propias de dicho métodos. Ademas de ello los miembros del equipo de desarrollo deberán tener un solido conocimiento sobre los siguientes puntos.

1. Especificaciones precisas.

   Las especificaciones precisas y el dominio de lenguajes de programación comprobables matemáticamente como SPARK son considerados obligatorios pues el cometido principal que se pretende obtener es eliminar cualquier ambigüedad, si se adopta CbyC omitiendo los pre-requisitos para le desarrollo anteriormente mencionados solo quedaran las buenas practicas provistas por los Métodos Ágiles, CbyC captura esta idea con la frase "Write Right" [11].

2. Validación robusta.

   Debido a que CbyC utiliza notaciones no ambiguas es posible utilizar metodos para validar los entregables de cada una de las etapas. Por ejemplo se puede demostrar que la especificación formal tiene ciertas propiedades de seguridad necesarias, que el código fuente está libre de errores de tiempo de ejecución, y que el código fuente implementa correctamente propiedades clave de la especificación considerando hacer valido el principio de la detección temprana de errores. CbyC captura esta idea con la frase "Check Here Before Going There" [11].

3. Desarrollo incremental.

   CbyC considera den los productoos ideas que se complementan, la primera es acerca de la minimizacion de las brechas semánticas entre artefactos por medio de la detección temprana de errores (ejemplo: sera difícil predecir el comportamiento del código generado de un mal diseño),la segunda es acerca de el incremento gradual de características considerando que el primer entregable sera el esqueleto gráfico completo del producto así como los mecanismos de comunicación, la funcionalidad real deberá ser alcanzada gradualmente de esta manera el sistema puede ser probado y demostrado desde el principio, que es una importante medida de fomento de la confianza. Es posible

combinar pequeñas brechas semánticas entre componentes, anotaciones precisas y validaciones robustas y lograr generar pruebas de certificación como un subproducto. CbyC captura esta idea con la frase "Step, Don't Leap" [11].

4. Evitar la repetición.

   La repetición es la segunda causa de la inserción de defectos situada abajo de los diseños con ambigüedades, se debe considerar que si existen descripciones dobles o información repetida provocara confusión en el equipo de desarrollo pues es regular que los datos varíen en cada una, debido a esto es necesario evitar repetir documentos de diseño ademas se debera considerar hacer documentos detallados solo cuando sea inevitable. CbyC captura esta idea con la frase "Say Something Once, Why Say It Again?" [11].

5. Luchar por la simplicidad.

   Sera necesario que los miembros del equipo mantengan simple cada aporte realizado al proyecto. CbyC captura esta idea con el acrónimo KISS [11].

6. Gestión de riesgo.

   Sera necesario desarrollar las partes menos obvias del sistema de esta manera se pretenden atacar los defectos con mayor nivel de complejidad cuando se tiene un mayor rango de recursos (tiempo y opciones de diseño) a disposición. CbyC captura esta idea con la frase "Do The Hard Things First" [11].

7. Pensar duro.

   CbyC subraya que el razonamiento lógico debe demostrar la aptitud para lograr el propósito del sistema. Un sistema desarrollado de esta manera debe ser certificable a cualquier seguridad aplicable o estándar de seguridad. CbyC captura esta idea con la frase "Argue Your Corner and Screws? Use A Screwdriver, Not A Hammer" [11].

¿Cómo evaluar CbyC? Debido a la naturaleza del proceso y su amplia relación con métodos tradicionales como PSP y TSP es posible utilizar las métricas que estos Métodos Tradicionales proponen, entre ellas., la cantidad de defectos inyectados en cada fase, la cantidad de defectos mitigados en cada fase, entre otros. CbyC propone como actividad genérica la recolección de métricas sin embargo no existe información de carácter publico que resuelva este problema.

¿Quiénes deben poner en practica CbyC? Praxis se ha convertido en el principal promotor de CbyC y propone como principales objetivos de mercado a organizaciones con los siguientes rubros [14].

1. Ferroviaria.
2. Aeroespacial.
3. Milicia.
4. Nuclear.
5. Administración de trafico aéreo.

¿Cuáles son los beneficios que ofrece CbyC? Los beneficios que aporta CbyC según Praxis se listan a continuación [15].

1. Alta productividad pues el enfoque ahorra esfuerzo significativamente en el desarrollo de pruebas.
2. Software con pocos defectos pues los resultados refieren la existencia de muy pocos errores después de la entrega.
3. Software Garantizado pues la confianza se refleja en un estándar propio de garantía de software.
4. Bajos costos de apoyo el Software es fácil de mantener, con defectos fijos por garantía.
5. Los clientes satisfechos el enfoque está orientado hacia el cumplimiento de los requisitos de negocio subyacentes del cliente.

Las cualidades que CbyC posee considerando los puntos propuestos por Ikram El rhaffari y Ounsa Roudies en su estudio y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación.

1. Cualidades relacionadas a la Ingeniería de Software.
   (a) El producto final tendrá enfoque en la seguridad.
   (b) Fácil trazabilidad a las fases de CTDS.
   (c) Se alinea con otros métodos de ingeniería de Software.

2. Cualidades relacionadas a la Seguridad Informática.
   (a) Posee un completo enfoque en la seguridad.
   (b) Implica y compromete a un equipo de seguridad y su interacción con otros actores del desarrollo.
   (c) Se implementa a través del CVDS.
   (d) Cumple con las necesidades de privacidad y confianza.

3. Cualidades relacionadas con TI.
   (a) Tiene un enfoqué en la calidad pues utiliza técnicas de PSP y TSP.
   (b) Provee una completa cobertura organizacional a través de las técnicas de PSP, TSP y Capability Maturity Model Integration (CMMI).
   (c) Se adapta fácilmente a pequeñas organizaciones.
   (d) Provee las características necesarias para evolucionar dentro de la organización.
   (e) Es posible la combinación de lineamientos CMMI para la evaluaciond e niveles de madurez.
   (f) Provee amplia cobertura a diferentes estructuras organizaciónales.

¿Cuáles son las deficiencias que presenta CbyC? Las deficiencias que CbyC posee considerando los puntos propuestos por Ikram El rhaffari y Ounsa Roudies en su estudio y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación [5].

1. Deficiencias relacionadas a la Ingeniería de Software.
   (a) Pocos recursos informativos disponibles.
   (b) Es un proceso considerado riguroso.
   (c) No existe una guía detallada para la realización de sus actividades.

2. Deficiencias relacionadas a la Seguridad Informática.
   (a) Es posible realizar en análisis de métricas sin embargo solo es propuesto como actividad genérica y existe detalles sobre como llevarlo acabo.

3. Deficiencias relacionadas con TI.
   (a) Solo existen datos informativos y no descriptivos.
   (b) Posee un un soporte deficiente para el aseguramiento de la calidad.

4.4 Proceso de Ingeniería de Requisitos de Seguridad

Security Requirements Engineering Process (SREP, Proceso de Ingeniería de Requisitos de Seguridad), describe como deben de ser integrados los requisitos de seguridad en el proceso de ingenieria de Software de manera sistemática e intuitiva. Security Requirements Engineering Process (SREP) describe cómo integrar el Common Criteria en el de ciclo de vida de desarrollo del Software [16].

¿Qué es SREP? SREP es un método para el establecimiento de requisitos de seguridad en el desarrollo de Security Information Systems (SIS) el cual esta basado en activos y es impulsado por riesgos. Describe cómo integrar el Common Criteria en el de ciclo de vida de desarrollo del Software por medio de la integración de un repositorio de recursos de seguridad que soporta el re-uso de requisitos, activos, amenazas y contra-medidas [16].

SREP posee 9 actividades así como varias interacciones, cada interacción tendrá como resultado avances internos o externos de varios artefactos que funcionaran como la linea base de proceso [16].

1. Realizar acuerdo sobre definiciones.

   En esta actividad sera necesario establecer un criterio común de entendimiento sobre las definiciones de seguridad que serán empleadas, así como las políticas de seguridad organizacionales y la visión de la seguridad del SIS. El documento de visión de la seguridad deberá ser escrito, en el se deberá hacer hincapié en la indicación del activo mas importante, la información [16].

2. Identificar activos críticos o vulnerables.

   En esta actividad se debe de considerar en primera instancia como activo mas valioso la información contenida en el sistema, ademas de ello se deberán valorar otros activos (dinero, productos tangibles o intangibles) según sea el caso y el enfoque del propio sistema [16].

3. Identificar objetivos de seguridad y sus dependencias.

   En esta actividad es posible utilizar el Security Resource Repository (SRR), si alguno de los activos identificados en la actividad anterior es parte del SRR sera posible re-utilizar los objetivos de seguridad, de lo contrario sera necesario determinar los objetivos para cada nuevo activo así como los requisitos legales y las limitaciones presentadas en las leyes del país donde se desarrollara el sistema [16].

4. Identificar amenazas y desarrollar artefactos.

   Si los activos identificados en la actividad anterior forman parte del SRR sera posible re-utilizar las amenazas asociadas, de lo contrario las nuevas amenazas podrían prevenir el objetivo de seguridad siendo logradas por medio de la instanciación de los casos de uso del negocio en casos de mal uso o instanciando los arboles de amenazas y ataques asociados con el negocio y el patrón de aplicación [16].

5. Evaluar de riegos.

   Una vez identificadas las amenazas se deberá determinar la probabilidad de correncia de cada amenaza así como evaluar su impacto y su riesgo de cada una, para llevar a cabo esta actividad es necesario utilizar la técnica propuesta por la Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información (MAGERIT) la cual se basa en tablas de análisis [17].

6. Obtener requisitos de seguridad.

   En esta actividad cada objetivo de seguridad debe ser analizado para asegurar su relevancia en conjunto con una lista de amenazas que impliquen mayor riesgo, esto con el fin de seleccionar los requisitos de seguridad adecuados para la depuración de amenazas con respecto a la evaluación de riesgos. Después sera necesario transformar los objetivos de seguridad (Confidencialidad, Integridad, Disponibilidad, Autenticidad, Responsabilidad) en limitantes sobre las operaciones que se utilizan en los requisitos funcionales [16].

7. Categorizar y priorizar los requisitos.

De acuerdo con el impacto y la probabilidad de las amenazas y según el riesgo, se debe clasificar los requisitos de seguridad en categorías por medio de la creación de taxonomías.

8. Inspección de requisitos.

   En esta actividad se deberá generar un reporte de validación. Por lo tanto sera necesario evaluar la calidad de los trabajos anteriormente realizados considerando los requisitos de aseguramiento de Common Criteria y los cuales fueron establecidos en conjunto con todos los interesados en la primera actividad, después se deberá realizar la primera version del Security Requirements Rationale Document (SRRD) con la ayuda de las clases de aseguramiento de Common Criteria mostrando que todos los requisitos de seguridad se cumplen y que se logran todos los objetivos de seguridad, que el problema de seguridad definido anteriormente se resuelve, que todas las amenazas han sido contrarrestadas, que las políticas de seguridad de la organización se hacen cumplir y que todos los supuestos se cumplen [16].

9. Mejora del SRR.

   En esta actividad se debera añadir al SRR los nuevos elementos (amenazas genericas y especificas, requisitos desarrollados en la actividad 4 y 6), posteriormente se debera crear el Security Target Document (STD) del Common Criteria [16].

¿Porqué adoptar SREP? SREP tiene como meta principal facilitar la reutilización de recursos, el proposito del desarrollo con el reuso de requisitos es identificar las descripciones de los sistemas que se podrían utilizar (total o parcialmente) con un número mínimo de modificaciones, reduciendo de este modo el esfuerzo total de desarrollo. La reutilización de requisitos de seguridad ayuda a aumentar la calidad: la inconsistencia, los errores, la ambigüedad y otros problemas pueden ser detectados y corregidos para una mejor utilización en proyectos posteriores. De esta manera, se garantiza la puesta en marcha de posibles ciclos de desarrollo rapidos basados en soluciones probadas [18].

¿Cómo adoptar SREP? SREP describe detalladamente como integrar el Common Criteria en el CVDS en conjunto con el SRR para promover el reuso de requisitos de seguridad que deberan ser modeloados por medio de las herramientas propuestas por UMLSec, SREP promueve la utilización de las etapas propuestas por Unified Process (UP), debido a esto es necesario que los equipos de desarrollo conozcan el funcionamiento de las herramientas externas utilizadas por SREP [18].

¿Cómo evaluar el impacto de SREP? Para lograr la correcta evaluación del impacto generado debido a la adopcion de SREP es necesario recurir a las

actividades propuestas por el Common Criteria en su seccion numero 3, en dicha seccion se definen los criterios de evaluación de Protection Profile (PP) y Security Target (ST), ademas se presentan los niveles de evaluación que deben ser utililzados, dicho niveles estan basados en una escala predefinida de calificacion y son llamados Evaluation Assurance Level (EAL) los cuales definen una escala para medir el aseguramiento de los componentes Target of Evaluation (TOE) [19].

Common Criteria establece los roles que deben ser incluidos en las actividades de evaluacioón y los cuales pueden ser mencionados de la siguiente manera [19].

1. Product Owner (PO)
2. Desarolladores
3. Evaluadores de seguridad de productos IT

La evaluación ha sido el medio tradicional para la obtención del aseguramiento y es la base del enfoque de Common Criteria. Las técnicas de evaluación pueden incluir, pero no están limitados a [19]:

1. Análisis y comprobación de procesos y procedimientos
2. Comprobación de la aplicación de los procesos y procedimientos definidos
3. Análisis de la correspondencia entre el diseño de representaciones TOE
4. Análisis del diseño de representación TOE contra los requisitos
5. Verificación de pruebas
6. Anáslis y guia de documentos
7. Análisis de pruebas funcionales desarrolladas y los resultados obtenidos
8. Pruebas funcionales independientes
9. Análisis de vulnerabilidades incluyendo hipotesis por defecto
10. Pruebas de penetración

Debido a que Common Criteria tiene como meta aplicar en minimo esfuerzo requerido para porveer el nivel necesario para el aseguramiento de la evaluación, el incremento del esfuerzo se constituye de la siguiente manera [19]:

1. Alcance, es decir, el esfuerzo es mayor debido a que una parte mayor del producto se incluye.
2. Profundidad, es decir, el esfuerzo es mayor, ya que se implementa en un nivel más fino de diseño y detalle de implementación.
3. Rigor, el esfuerzo es mayor, ya que se aplica de manera más estructurada y formal.

¿Quiénes deben poner en practica SREP? La organizacion internacional de Common Criteria para la certificacion de productos menciona que las organizaciones candidatas a la inserción de actividades de Common Criteria son aquellas enfocadas en desarrollar productos ligados a las siguientes categorias [20].

1. Sistemas y dispositivos para el control de acceso
2. Sistemas y dispositivos biometricos
3. Sistemas y dispositivos de protección de fronteras
4. Proteccion de datos
5. Bases de datos
6. Sistemas y dispositivos de detección
7. Integrated Circuit (IC), targetas inteligentes, sistemas y dispositivos relacionados con targetas inteligentes
8. Sistemas de administración de llaves
9. Sistemas con funciones multiples
10. Sistemas y dispositivos realacionados con las redes
11. Sistemas operativos
12. Sistemas y dispositivos relacionados a firmas digitales
13. Computación confiable

¿Cuáles son los beneficios que ofrece SREP? Las cualidades que SREP posee considerando los puntos propuestos por Ikram El rhaffari y Ounsa Roudies en su estudio y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación.

1. Cualidades relacionadas a la Ingeniería de Software.
   (a) El producto final tendrá enfoque en la seguridad.
   (b) Se alinea con otros métodos de ingeniería de Software.
   (c) Las actividades que lo componene se encuentran descritas en las guías formales propuestas por Common Criteria y UP.

2. Cualidades relacionadas a la Seguridad Informática.
   (a) Posee un completo enfoque en la seguridad.
   (b) Implica y compromete a un equipo de seguridad y su interacción con otros actores del desarrollo debido al enofque interactivo adoptado de UP.
   (c) Cumple con las necesidades de privacidad y confianza.
   (d) Sus metricas de seguridad se encuentran definidas por Common Criteria.

3. Cualidades relacionadas con TI.
   (a) Tiene un enfoqué en la calidad pues utiliza técnicas de Common Criteria y UP.
   (b) Provee evolucion y versionamiento constante bajo la supervision de International Organization for Standardization (ISO).
   (c) Es posible el uso de Common Criteria Evaluation Assurance Levels (CCEAL) para la evaluacion de niveles de madurez.

¿Cuáles son las deficiencias que presenta SREP? Las deficiencias que SREP posee considerando los puntos propuestos por Ikram El rhaffari y Ounsa Roudies en su estudio y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación [5].

1. Deficiencias relacionadas a la Ingeniería de Software.
   (a) No utiliza las fases estandar de desarrollo de Software o CTDS.
   (b) Existen uy pocos recursos informativos disponibles.
   (c) Es de caracter riguroso.

2. Deficiencias relacionadas a la Seguridad Informática.
   (a) Debe exitir una conciencia preliminar sobre la seguridad en la organización.

3. Deficiencias relacionadas con TI.
   (a) La documenación disponible se encuentra bajo cargos economicos.
   (b) No provee impacto ornanizacional general pues solo se enfoca en el proceso de desarrollo de Software.
   (c) Es necesario acceder a soporte externo para asegurar los niveles de calidad.
   (d) Se enfoca en ornizaciones con ciertos grados de madurez y potencial economico.

4.5 Modelado Conceptual de Seguridad

Conceptual Security Modeling (CoSMo, Modelado Conceptual de Seguridad), propuesto por Christine Artelsmair, Wolfgang Essmayr, Peter Lang, Roland Wagner y Edgar Weippl, Conceptual Security Modeling (CoSMo) fue diseñado para reducir la brecha resultante del trato pobre de la seguridad durante la realización de los requisitos de seguridad de desarrollo de software, segun [21] la importancia de la seguridad es subvaluada y considerada como una especie de complemento que será aplicado al sistema después del desarrollo.

¿Qué es CoSMo? Es un técnica de modelado conceptual, segun [21] esta constituida por requisitos y mecanismos de seguridad, cada requisito puede ser alcanzado por medio de uno o mas mecanismos de seguridad, esto resulta en una matriz requisitos/mecanismos, dichos requisitos y mecanismos deberan ser formulados como una abstraccion de alto nivel.

A continuación se presentan los requisitos y mecanismos considerados por [21] como los mas importantes.

1. Autenticidad, Autenticación e Identificacion.

   Segun [21] la Autenticación aborda el requisito de aseguramiento de Autenticidad, la Autenticación puede tomar varias formas.

(a) Autenticación general de la identidad / autenticación
(b) Autenticación del contenido del mensaje / autenticación
(c) Autenticacion del origen del mensaje / autenticación

En [21] se menciona que la Autenticación general de la identidad es comunmente consevida por medio de la insercion de un nombre de usuario (Identificador) y una clave (Password). La Autenticación del contenido del mensaje se refiere al proceso de verificacion de la información contenida en el mensaje que fue enviado. La Autenticación del origen del mensaje se refiere al proceso de verificación del emisor y el origen de un mensaje, esto es logrado por medio de el mecanismo de Autenticación del origen del mensaje incluyendo los conceptos de: firmas digitales, certificados digitales y confianza con terceros.

2. Integridad, Discreción y Privacidad.

   De acuerdo con [21] existen dos tipos de restricciones que corresponden directamente a la Integridad y a la Discresción. En general, las restriciones de Integridad son reglas que administran la actulizacion de la informacion y la validación de los datos. Las restriciones de Discreción son reglas que administran la clasificacion de los datos y su acceso. Los dos tipos de restricciones se subdividen en subtipos: restricciones semanticas y restriciones de Control de Acceso.

   Segun [21] las restricciones de la Integridad semantica permiten definir y mantener el correcto estado de la información durante la operación. Las restricciones de la Integridad del Control de Acceso son declaraciones explicitas que pretenden notificar la Autorización para la modificación de ciertos datos con el fin de proteger las modificaciones maliciosas o accidentales. Las restriciones de la discrecion semantica especifican los niveles en los cuales los datos y su asociacion son clasificados. Las restricciones de la Discresion del Control de Acceso especifican cuales usuarios estan autorizados para accesar a ciertos datos.

3. Atorización, Control de Acceso y Disponibilidad.

   En [21] se mensiona que la Autorización es la especificación de un conjunto de reglas acerca de "¿Quien tiene?, ¿Cual tipo de acceso?, ¿A que información?". El Control de Acceso es logrado por medio de procesdimiento que controlan la Autorización limitando el acceso a los datos solo a usuarios autorizados, el Control de Acceso usualmente requiere de la Autenticación como prerequisito. La Autorización y el Control de Acceso se vinculan con los requisitos de Discreción y Privavidad de los Information Systems (IS).

   El control de acceso puede tomar control por medio de las siguentes formas:

(a) Control de Acceso discrecional.
(b) Control de Acceso obligatorio.
(c) Control de Acceso basado en roles.

En [21] se dice que la Disponibilidad es el requisito de servir a los actores autorizados con la información adecuada cuando esta sea requerida. Para lograr el aseguramiento de la Disponibilidad es necesario proteger la información de los actores no autorizados. Para mantener la información disponible es necesario controlar el acceso a los datos.

4. Rendicion de Cuentas, Auditoria y No Repudio.

   De acuerdo a [21] la Rendición de Cuentas captura el requisito realacionado a la responsabilidad que los individuos tinen sobre las actividades relavantes de reguridad y el cual puede ser alcanzado por medio del mecanismo de Auditoria. El No Repudio es un caso particular de Rendicion de Cuentas y es un requisito que sera logrado por medio de metodos de Criptografía. Al mecanismo encargado de llevar registro de todas las actividades relacionadas a la seguridad por un usuario es llamado Auditoria. La Auditoria esta formada por dos componentes: la recolección y organización de los datos asi como el analisis de los datos para descubrir vulnerabilidades.

5. Anonimato y Originalidad.

   En [21] se define como Anonimato a la carencia de identidad y se menciona que existen dos enfoques para proveer de Aninimato en la red:

   (a) Servicios anonimos: basados en un sistema de trasporte no anonimo (capa 1 a 4 en Open Systems Interconnection Model (OSI)).
   (b) Redes anonimas: implementando la anonimidad en las capas bajas de OSI y construyendo diferentes servicios sobre ella con Anonimato hasta cierto punto (por ejemplo, anonimato real, pseudo-anonimato, auto-identificación opcional).

   En este punto es necesario considerar a la infraestructura como un mecanismo mas, el cual jugara un papel muy importante para el desarrollo de nuevos sistemas seguros.

6. Validez.

   En [21] se menciona que los contratos y firmas digitales poseen validez legal y que debe de ser considerarda cada una de las vulnerabilidades existentes de los metodos implementados. Para lograr alcanzar una correcta Validez en los IS sera necesario considerar la integración de nuevas herramientas dentro de la infraestructura.

7. Criptografía y Mecanismos Criptográficos.

   Segun [21] los Mecanismos Criptograficos mas frecuentes son:

   (a) Algoritmos de cifrado que protegen la Confidencialidad de los datos.
   (b) Firmas digitales que provee de iIntegridad y Autenticidad.
   (c) Funciones para la revision de la Integridad (Hash Functions) usadas tipicamente en conección con firmas digitales.

8. Certificados Digitales y Terceros Confiables.

   En [21] se menciona que los Certificados Digitales son usados para proveer Autenticidad a los involucradosen la comunicación, por ejemplo: la certificación de una llave publica provee la confianza de un individuo u organización. Una Certification Authority (CA) actua como un Tercero Confiable que garantiza el enlace seguro entre los involucrados.

¿Porqué adoptar el CoSMo? El cometido principal de CoSMo es solucionar la falta de una técnica de modelado conceptual apropiado para la seguridad, [21] identifica la necesidad de integrar ciertas consideraciones de seguridad dentro del proceso de modelado de Software y realiza una contribucion doble, [21] menciona en primer lugar que el modelado conceptual abarca los requisitos y mecanismos de seguridad de alto nivel y como es que las consideraciones de seguridad se pueden integrar en el proceso de modelado conceptual mejorando el proceso de manera radical, en segundo lugar menciona que los requisitos frecuentes de seguridad que claramente indican cuales mecanismos son utilizados para lograr su propio cumplimiento deben ser sistemáticamente archivados, mejorando asi el consumo de recursos como tiempo de investigación y desarrollo.

¿Cómo adoptar el CoSMo? En [21] se menciona que al adoptar el CoSMo es necesario el uso de herramietas tales como UML ya que permiten el modelado de requisitos de seguridad a nivel conceptual por medio de diagramas de casos de uso, [21] menciona que UML es fácil, comprensible y ayuda a construir un entendimiento común de un nuevo sistema, debido a esto se debera poseer un conocimiento adecuado para la construcion de diagramas de calidad. Es importante tener en cuenta que sólo exigen requisitos de seguridad y que no existe mecanismo alguno que especifique cómo lograr dichos requisitos. Segun [21] los diagramas de casos de uso pueden ser enriquecidos a nivel conceptual por medio de los requisitos descritos con anterioridad en la sección ¿Qué es CoSMo?.

¿Cómo evaluar el impacto del CoSMo? CoSMo no posee un metodo definido para su evaluación, sin embargo es posible inferir la posibilidad de evaluar la completitud de requisitos de seguridad por medio de la caparativa entre los requisitos necesarios para el sistema y aquellos logrados.

¿Quiénes deben poner en practica el CoSMo? [21] menciona que las organizacines que deben utilizar el CoSMo son aquellas enfocadas en desarrollar sistemas ligados a las siguientes categorias entre otros:

1. Sistemas y dispositivos de control de acceso.
2. Sistemas y dispositivos confiables.
3. Sistemas y dispositivos biometricos.
4. Sistemas y dispositivos de comercio electronico.
5. Sistemas y dispositivos bancarios.

¿Cuáles son los beneficios que ofrece el CoSMo? Las cualidades que CoSMo posee considerando los puntos propuestos por Ikram El rhaffari y Ounsa Roudies en su estudio y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación.

1. Cualidades relacionadas a la Ingeniería de Software.
   (a) Esta digirido al desarrollo de varios tipos de Software.
   (b) Se alinea facilmente con las fases del CVDS.
   (c) Se alinea con otros metodos de ingeniería de Software que consideran una etapa especifica para la administración de requisitos.

2. Cualidades relacionadas a la Seguridad Informática.
   (a) Posee un completo enfoque en la seguridad.
   (b) Considera la conciencia en la seguridad como requisito preliminar.

3. Cualidades relacionadas con TI.
   (a) Provee la capacidad de evolucionar pues es posible insertar otros requisitos y mecanismos en su estructura.
   (b) Esta dirigido a pequeñas organizaciones.

¿Cuáles son las deficiencias que presenta el CoSMo? Las deficiencias que CoSMo posee considerando los puntos propuestos por Ikram El rhaffari y Ounsa Roudies en su estudio y el cual esta enfocado en el análisis de elementos pertenecientes a la Ingeniería de Software, a la SI y a las TI y son listadas a continuación [5].

1. Deficiencias relacionadas a la Ingeniería de Software.
   (a) Muy pocos recursos disponibles.
   (b) No existe una guía para su resalización, la información existente solo acopla los requisitos propuestos y los mecanismos usados para alcanzarlos asi como algunos casos practicos.

2. Deficiencias relacionadas a la Seguridad Informática.

   (a) No existe especificación alguna sobre la implicación de equipos de seguridad, aunque ha sido considerado para trabajos futuros.
   (b) No existe información sobre técnicas para el aseguramiento del cumplimiento y privacidad.
   (c) No posee métricas de seguridad.

3. Deficiencias relacionadas con TI.
   (a) No existe información sobre su enfoque en la calidad, aunque puede ser combinado con algun metodo para el aseguramiento de la calidad
   (b) La documentación existente es muy limitada.
   (c) Está digido solo a equipos de desarrollo.
   (d) No posee niveles de madurez.
   (e) No posee técnias para el aseguramiento de la calidad.

4.6 Lenguaje Unificado de Modelado Afín a la seguridad

¿Qué es el Lenguaje Unificado de Modelado Afín a la seguridad?

¿Porqué adoptar el Lenguaje Unificado de Modelado Afín a la seguridad?

¿Cómo adoptar el Lenguaje Unificado de Modelado Afín a la seguridad?

¿Cómo evaluar el impacto del Lenguaje Unificado de Modelado Afín a la seguridad?

¿Quiénes deben poner en practica el Lenguaje Unificado de Modelado Afín a la seguridad?

¿Cuáles son los beneficios que ofrece el Lenguaje Unificado de Modelado Afín a la seguridad?

¿Cuáles son las deficiencias que presenta el Lenguaje Unificado de Modelado Afín a la seguridad?

4.7 Casos de Uso Incorrecto

¿Qué son los Casos de Uso Incorrecto?

¿Porqué adoptar los Casos de Uso Incorrecto?

¿Cómo adoptar los Casos de Uso Incorrecto?

¿Cómo evaluar el impacto de los Casos de Uso Incorrecto?

¿Quiénes deben poner en practica los Casos de Uso Incorrecto?

¿Cuáles son los beneficios que ofrecen los Casos de Uso Incorrecto?

¿Cuáles son las deficiencias que presentan los Casos de Uso Incorrecto?

4.8 Metodología de Pruebas de Seguridad de Código Abierto

¿Qué es la Metodología de Pruebas de Seguridad de Código Abierto?

¿Porqué adoptar la Metodología de Pruebas de Seguridad de Código Abierto?

¿Cómo adoptar la Metodología de Pruebas de Seguridad de Código Abierto?

¿Cómo evaluar la Metodología de Pruebas de Seguridad de Código Abierto?

¿Quiénes deben poner en practica la Metodología de Pruebas de Seguridad de Código Abierto?

¿Cuáles son los beneficios que ofrece la Metodología de Pruebas de Seguridad de Código Abierto?

¿Cuáles son las deficiencias que presenta la Metodología de Pruebas de Seguridad de Código Abierto?

4.9 Construcción de la Seguridad por Modelos de Madurez

¿Qué es la Construcción de la Seguridad por Modelos de Madurez?

¿Porqué adoptar la Construcción de la Seguridad por Modelos de Madurez?

¿Cómo adoptar la Construcción de la Seguridad por Modelos de Madurez?

¿Cómo evaluar la Metodología de Pruebas de Seguridad de Código Abierto?

¿Quiénes deben poner en practica la Construcción de la Seguridad por Modelos de Madurez?

¿Cuáles son los beneficios que ofrece la Construcción de la Seguridad por Modelos de Madurez?

¿Cuáles son las deficiencias que presenta la Construcción de la Seguridad por Modelos de Madurez?

4.10 Sistema de Calidad de Requisitos de Ingeniería

¿Qué es el Sistema de Calidad de Requisitos de Ingeniería?

¿Porqué adoptar el Sistema de Calidad de Requisitos de Ingeniería?

¿Cómo adoptar el Sistema de Calidad de Requisitos de Ingeniería?

¿Cómo evaluar el Sistema de Calidad de Requisitos de Ingeniería?

¿Quiénes deben poner en practica el Sistema de Calidad de Requisitos de Ingeniería?

¿Cuáles son los beneficios que ofrece el Sistema de Calidad de Requisitos de Ingeniería?

¿Cuáles son las deficiencias que presenta el Sistema de Calidad de Requisitos de Ingeniería?

Glossary

Common Criteria Common Criteria para Information Technology Security Evaluation (abreviado como: Common Criteria o CC) es un estandar internacional (ISO/IEC 15408) para la certificación de la seguridad. Actualmente se encuentra en la version 3.1 revision 4. 33, 35, 36, 37

Criptografía es una disciplina científica enfocada a crear fórmulas matemáticas que proporcionan altos niveles de seguridad. Dichas fórmulas son aplicadas a mensajes para hacer representaciones lingüísticas ininteligibles a simple vista y que se puedan descifrar únicamente conociendo la fórmula o las llaves que se usaron para cifrarlo. 5, 8, 40

Cross-Site Scripting es un tipo de inseguridad informática o agujero de seguridad típico de las aplicaciones en red también conocido como XSS, que permite a una tercera persona inyectar en páginas web visitadas por el usuario código JavaScript o en otro lenguaje similar (ej: VBScript), evitando medidas de control como la Política del mismo origen. 5

Firewall es software o hardware que ayuda a evitar que los hackers y algunos tipos de malware de llegar a su PC a través de una red o de Internet. Para ello, el control de la información que viene de Internet o una red y, a continuación, ya sea bloqueando o permitiendo que pase a través de su PC.. 8

Flexible es la capacidad de hacer cambios en el producto que se está desarrollando, o en la forma en que se desarrolla, hasta relativamente tarde en el desarrollo, sin ser demasiado perturbador. En consecuencia, la posterior puede hacer cambios, entre más flexible es el proceso menos perjudicial es.. 18

Framework es una abstracción en la que el software que proporciona una funcionalidad genérica se puede cambiar de forma selectiva por el código escrito por el usuario adicional, proporcionando así el software de aplicación específica. 7

Hash Functions Una función de hash es una función que se puede utilizar para mapear los datos de tamaño arbitrario a los datos de tamaño fijo. Los valores devueltos por una función hash se llaman valores hash, códigos hash, sumas de hash, o simplemente hashes. Un uso es una estructura de datos llamada una tabla hash, ampliamente utilizado en los programas informáticos para la rápida búsqueda de datos. 41

Hashing es la producción de valores hash para acceder a datos o para la seguridad. Un valor hash (o simplemente hash), también llamado un resumen del mensaje, es un número generado a partir de una cadena de texto. El hash es sustancialmente menor que el propio texto, y se genera por una fórmula de tal manera que es muy poco probable que algún otro texto producirá el mismo valor hash.. 8

Identificador Los identificadores son símbolos léxicos que nombran entidades. 39

KISS es un acrónimo de "Keep It Simple, Stupid" como un principio de diseño señalado por la Marina de Estados Unidos en 1960. El principio KISS afirma que la mayoría de los sistemas funcionan mejor si se mantienen simples en lugar de hacerse complicados; Por lo tanto, la simplicidad debe ser un objetivo clave en el diseño y la complejidad innecesaria debe evitarse. La frase se ha asociado con el ingeniero aviones Kelly Johnson (1910 a 1990). 31

Métodos Ágiles es un proceso incremental, pequeño y frecuente con entregas con ciclos rápidos, también Cooperativo (clientes y desarrolladores trabajan constantemente con una comunicación muy fina y constante), sencillo (El método es fácil de aprender y modificar para el equipo, es documentado y adaptativo (capaz de permitir cambios de último momento). Las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que puede que no curen todos los males pero harán la entrega del proyecto menos complicada y más satisfactoria tanto para los clientes como para los equipos de entrega. 27, 28, 29, 30

Métodos Tradicionales es una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. Las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar. 27, 28, 30, 31

Online es un estado de conectividad, frente al término fuera de línea (Offline) que indica un estado de desconexión. 14, 15, 18, 26

Open Source se refiere a un programa o software en el que el código fuente (la forma del programa cuando un programador escribe un programa en un lenguaje de programación en particular) está a disposición del público en general para su uso y / o modificación de su diseño original sin cargo . Código fuente abierto se crea normalmente como un esfuerzo de colaboración en el que los programadores mejorar el código y comparten los cambios dentro de la comunidad. 18

Password es una cadena de caracteres utilizados para la autenticación del usuario para demostrar la aprobación de identidad o de acceso para obtener acceso a un recurso (ejemplo: un código de acceso es un tipo de contraseña que debe ser mantenida en secreto). 39

Praxis Integrity Partnership Creativity Excellence, The Foremost International Specialist in Critical Systems Engineering. `http://www.praxis-his.com`. 27, 29, 31, 32

Pruebas Fuzz es una técnica sencilla para la alimentación de entrada al azar a las aplicaciones. 7, 10

Software es cualquier conjunto de instrucciones que dirige un sistema o equipo electrónico para llevar a cabo operaciones específicas. Las aplicaciones informáticas se compone de programas de ordenador, bibliotecas y datos no ejecutables relacionados (como la documentación en línea o los medios digitales). 1, 2, 3, 17, 18, 19, 22, 23, 25, 26, 27, 28, 29, 30, 32, 33, 37, 38, 41, 42

SPARK es un lenguaje de programación especialmente diseñado para sistemas de alta integridad. Es un subconjunto anotado de Ada desarrollado por la empresa británica Praxis High Integrity Systems, que elimina ciertas características del lenguaje consideradas peligrosas en este tipo de sistemas (como las excepciones o la sobrecarga de operadores), y que añade anotaciones formales para realizar automáticamente análhampton Program Analysis Development Environment), un conjunto de herramientas destinadas al análisis de flujo de datos y de información. El nombre SPARK deriva de SPADE Ada Kernel. 30

STRIDE es un sistema desarrollado por Microsoft para pensar acerca de las amenazas de seguridad informática. Proporciona una regla mnemotécnica para amenazas de seguridad en seis categorías: Spoofing of user identity, Tampering, Repudiation, Information disclosure (privacy breach or data leak), Denial of service (D.o.S), Elevation of privilege. 9

Taxonomia es la práctica y ciencia de la clasificación. La palabra también se utiliza como sustantivo, una taxonomía o esquema taxonómico, es una clasificación particular. 22

UMLSec es una extensión para el Lenguaje Unificado de Modelado (UML) para la integración de la información relacionada con la seguridad en las especificaciones UML. 35

UNIX es un sistema operativo portable, multitarea y multiusuario desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Dennis Ritchie, Ken Thompson y Douglas McIlroy.. 21

Visual Paradigm es una herramienta CASE UML en apoyo a UML 2, SysML y Business Process Modeling Notation del Object Management Group. 28

Acronyms

AES Advanced Encryption Standard. 8

CA Certification Authority. 41
CAS Code Access Security. 8
CbyC Correctness by Construction. 27, 28, 29, 30, 31, 32
CbyD Correctness by Debuging. 27
CCEAL Common Criteria Evaluation Assurance Levels. 37
CLASP Comprehensive, Lightweight Application Security Process. 18, 19, 20, 21, 22, 23, 24, 26
CMMI Capability Maturity Model Integration. 32
CoSMo Conceptual Security Modeling. 38, 41, 42
CTDS Ciclo Tradicional de Desarrollo de Software. 4, 5, 7, 9, 10, 11, 13, 17, 26, 32, 37
CUV Casos de Uso de Vulnerabilidades. 21
CVDS Ciclo de Vida de Desarrollo de Software. 3, 5, 7, 15, 16, 19, 23, 26, 32, 35, 42

EAL Evaluation Assurance Level. 35

FOSS Free and Open-Source Software. 23

IC Integrated Circuit. 37
IS Information Systems. 39, 40
ISO International Organization for Standardization. 37
IT Information Technologies. 18, 22, 26, 36

LV Léxico de Vulnerabilidades. 19, 21

MAGERIT Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información. 34

OSI Open Systems Interconnection Model. 40
OWASP The Open Web Application Security Project. 18, 20, 23, 25, 26

PO Product Owner. 36
PP Protection Profile. 35
PSP Personal Software Process. 27, 30, 31, 32

RSL Revisión Sistemática de la Literatura. 2

SAMM Software Assurance Maturity Model. 25
SDL Security Development Lifecicle. 3, 4, 13, 14, 15, 16, 17, 18
SI Seguridad Informática. 17, 18, 26, 32, 37, 42
SIS Security Information Systems. 33

SQL Structured Query Language. 5
SREP Security Requirements Engineering Process. 33, 35, 36, 37
SRR Security Resource Repository. 34, 35
SRRD Security Requirements Rationale Document. 35
ST Security Target. 35
STD Security Target Document. 35

TI Tecnologías de la Información. 17, 18, 26, 27, 32, 33, 37, 38, 42, 43
TOE Target of Evaluation. 35, 36
TSP Team Software Process. 27, 30, 31, 32

UML Unified Modeling Language. 28, 41
UP Unified Process. 35, 37
URL Uniform Resource Locator. 15

VBEA Vista Basada en Evaluación de Actividades. 19, 23, 24
VBIA Vista Basada en la Implementación de Actividades. 19
VBR Vista Basada en Roles. 19
VC Vista de Conceptos. 19, 21, 23
VS Vulnerabilidad de Seguridad. 22
VV Vista de Vulnerabilidades. 19

References

1. Microsoft, "Microsoft security development adoption: Why and how," Septiembre 2013.
2. Microsoft, "Implementación simplificada del proceso sdl de microsoft," Febrero 2010.
3. C. S. Nuno Teodoro, "Web application security," Enero 2011.
4. S. L. Michael Howard, "Microsoft security development lifecicle," Junio 2015.
5. O. R. Ikram El rhaffari, "Benchmarking sdl and clasp lifecycle," Mayo 2014.
6. OWASP, "CLASP introduction." `https://www.owasp.org/index.php/CLASP`. Fecha de acceso: 25-11-2015.
7. OWASP, "CLASP concepts." `https://www.owasp.org/index.php/CLASP_Concepts`. Fecha de acceso: 25-11-2015.
8. OWASP, "Clasp activity-assessment view," Marzo 2006.
9. OWASP, "SAMM introduction." `https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model`. Fecha de acceso: 30-11-2015.
10. OWASP, "OWASP introduction." `https://www.owasp.org/index.php/Category:OWASP_Project`. Fecha de acceso: 30-11-2015.
11. P. Amey, "CbyC cbyc." `https://buildsecurityin.us-cert.gov/articles/knowledge/sdlc-process/correctness-by-construction`. Fecha de acceso: 2-12-2015.
12. C. J. B. Abundis, "Metodologías para desarrollar software seguro," Diciembre 2013.
13. R. C. Anthony Hall, "Correctness by construction: Bettercan also be cheaper," Febrero 2002.
14. Praxis, "Praxis the foremost international specialist in critical systems engineering." `http://www.praxis-his.com/`. Fecha de acceso: 6-12-2015.

15. Praxis, "Praxis high integrity systems." http://web.archive.org/web/20081114100215/http://www.praxis-his.com/services/software/approach.asp. Fecha de acceso: 6-12-2015.
16. M. P. Daniel Mellado, Eduardo Fernandez-Medina, "Applying a security requirements engineering process," 2006.
17. MAP, "Metodología de análisis y gestión de riesgos de los sistemas de información (magerit - v 2)," 2005.
18. M. P. Daniel Mellado, Eduardo Fernandez-Medina, "A common criteria based security requirements engineering process for the development of secure information systems," 2007.
19. C. C. Org, "Introduction and general model," 2012.
20. C. C. Org, "Common Criteria certified products list - statistics." http://www.commoncriteriaportal.org/products/stats/. Fecha de acceso: 20-12-2015.
21. P. L. R. W. E. W. Christine Artelsmair, Wolfgang Essmayr, "Cosmo: An approach towards conceptual security modeling," 2002.