Hacia la Ingeniería de Software Seguro

Marta Castellaro, Susana Romaniz, Juan Ramos, Pablo Pessolani

Facultad Regional Santa Fe - Universidad Tecnológica Nacional
Lavaise 610 (S3004EWB) Santa Fe Argentina - (54-342 4601579/2390)
mcastell@frsf.utn.edu.ar, sromaniz@frsf.utn.edu.ar,
jramos@frsf.utn.edu.ar, ppessolani@frsf.utn.edu.ar

Resumen. La criticidad de los actuales sistemas de información en diferentes dominios de la sociedad determina que, si bien es importante asegurar que el software se desarrolla de acuerdo a las necesidades del usuario (requerimientos funcionales), también lo es garantizar que el mismo sea seguro. Hasta hace poco tiempo la Ingeniería de Software y la Ingeniería de Seguridad venían desarrollándose de forma independiente, limitando la posibilidad de que la seguridad sea considerada como parte del proceso de desarrollo de sistemas de software seguros. Si no se dispone de una definición precisa de lo que significa seguridad y cómo debe comportarse un software, no tiene sentido alguno preguntarse si el mismo es seguro o no. Es necesario revisar la terminología y definir un conjunto de conceptos de manera más rigurosa, considerar y comparar diferentes metodologías de tratamiento de la seguridad pasibles de ser articuladas con las metodologías para el proceso del ciclo de vida del desarrollo de software ya existentes, y considerar los modelos de madurez y aseguramiento de calidad. Este artículo presenta una revisión de la taxonomía existente en ese dominio, y un análisis sobre metodologías actuales conducentes hacia una disciplina denominada Ingeniería de Software Seguro.

Palabras claves. Software assurance. Quality assurance. Software Secure.

1. Introducción

La creciente dependencia de tareas críticas respecto del software conlleva a que el valor del mismo ya no resida solamente en la aptitud que posee de mejorar o sostener la productividad y la eficiencia de las organizaciones, sino que además se requiere que posea la capacidad de continuar operando de manera confiable aún cuando deba enfrentar eventos que amenazan su utilización. Esto ha conducido a un conjunto de cambios, derivados de la incorporación de los aspectos de seguridad, en la visión de las propiedades de los sistemas de información que hacen uso intensivo de software, por lo que resulta necesario revisar algunos conceptos de manera más precisa. Surgen de esta revisión dos conceptos fundamentales como son "*software assurance"* (SA) y "*software secure"*

(SS), y las implicancias que traen aparejados los avances de las áreas de la Ingeniería de Software y de la Ingeniería de Seguridad en los últimos años. Actualmente se encuentran definidas y publicadas varias metodologías nuevas para el ciclo de vida de desarrollo del software (del inglés *Software Development Life Cycle* -SDLC-), pasibles de ser articuladas con las tradicionales, que permiten incorporar de manera controlada el conjunto de aspectos que satisfacen los requerimientos de SA y SS.

2. Definiciones y Taxonomía

Hasta hace pocos años, el concepto SA hacía referencia a dos propiedades del software: calidad y confiabilidad (*reliability*); pero en la actualidad, ha sido extendido para expresar la idea de 'seguridad garantizada' del software. Esto demanda una especial atención debido al incremento de las amenazas atribuibles a la posibilidad de 'explotar vulnerabilidades' presentes en el software, poniendo en riesgo a las organizaciones y al cumplimiento de su misión [1]. El nivel de exposición al riesgo es cada vez mayor y, en general, se comprende de manera limitada debido a que: a) el software es el nexo más débil en la correcta ejecución de sistemas interdependientes; b) el tamaño y la complejidad del software dificultan su comprensión e imposibilitan la realización de pruebas exhaustivas; c) existen limitaciones para realizar un examen exhaustivo de software adquirido o de componentes de software que se integran; d) los programas de ataque presentan una naturaleza más furtiva y sofisticada; e) las consecuencias que resultan de la reutilización de software legado en nuevas aplicaciones son imprevisibles; f) los líderes de las organizaciones tienen dificultades para respaldar decisiones respecto a inversiones en seguridad del software acordes a los riesgos que enfrentan.

Se pueden encontrar diferentes definiciones para el concepto de SA, entre las que se destacan por ser las más difundidas las propuestas en [2], [3], [4], [5] y [6]. Las mismas se complementan entre sí, y difieren ligeramente en el énfasis y el abordaje del problema de garantizar la seguridad del software. Para el presente artículo se adopta una definición que sintetiza aquéllas obtenidas de distintas fuentes [7]: El concepto de SA implica contar con la confianza justificada en que el software exhibirá todas las propiedades requeridas de manera consistente y garantizará confiabilidad durante su operación a pesar de la presencia de fallas intencionales; en términos prácticos, el software debe: 1) ser capaz de resistir la mayoría de los ataques, 2) tolerar tanto como resulte posible aquellos ataques que no puede resistir, 3) contener el daño, 4) recuperarse a un nivel normal de operación tan pronto como sea posible luego de los ataques que sea incapaz de resistir o de tolerar.

Los siguientes conceptos respecto al software están incluidos en SA [8]:

- *Confiabilidad (Reliability)*. La probabilidad que el software opere libre de fallas durante un período/intervalo de tiempo especificado/esperado, o por un número especificado/esperado de operaciones, en un ambiente especificado/esperado bajo condiciones operativas especificadas/esperadas;
- *Protección (Safety)*. La persistencia de la confiabilidad de cara a accidentes o percances, es decir, eventos no planificados que derivan en muerte, lesión, enfermedad, daño a la propiedad o su pérdida, o perjuicios al ambiente;
- *Seguridad (Security)*. La habilidad del software de resistir, tolerar y recuperarse de acontecimientos que intencionalmente amenazan su confiabilidad.

La existencia de vulnerabilidades en el software y de métodos de ataque conocidos que se

aprovechan de ellas, facilita que los atacantes tengan al software por objetivo para violar una o más propiedades de seguridad, o para forzarlo a ejecutar en estado inseguro. Un software que posee el atributo de seguridad permanece confiable (es decir, correcto y predecible) a pesar de los esfuerzos intencionales de comprometer su confiabilidad. El objetivo principal es la construcción de software de calidad superior y que continúe funcionando correctamente bajo un ataque intencional [9].
El software que ha sido desarrollado teniendo en cuenta la seguridad, debería presentar presenta las siguientes propiedades a lo largo de su ciclo de vida:

- *Ejecución predecible*: Existe la confianza justificada respecto a que, cuando se ejecuta, funciona de la manera esperada; lo que implica que está significativamente reducida o eliminada la habilidad de ingresos maliciosos destinados a alterar la ejecución o los resultados producidos.
- *Trustworthiness*: Existe un número mínimo de vulnerabilidades explotables intencionalmente (si bien la meta es que no exista ninguna vulnerabilidad explotable).
- *Conformidad*: Las actividades planificadas, sistemáticas y multidisciplinarias con las que ha sido desarrollado aseguran que los componentes, productos y sistemas de software cumplen los requerimientos, los estándares aplicables, y los procedimientos especificados para su uso

Estas propiedades se deben interpretar y limitar en base a los escenarios reales, tales como: a) cuál es el nivel requerido de seguridad; b) cuáles son los aspectos más críticos a lo que se deben atender; c) qué acciones resultan aptas para el costo y la programación del proyecto. Estas son cuestiones que se deben resolver mediante una correcta "gestión del riesgo de seguridad".
Los resultados en el campo de la investigación y las experiencias de la industria indican que reducir las vulnerabilidades potenciales tan temprano como sea posible dentro del SDLC, particularmente mediante la adopción de procesos y prácticas mejoradas para la seguridad, resulta más adecuado que la modalidad tradicional difundida de desarrollar el software, y luego liberar parches para el software cuando está operativo [8, 10].

3. Propiedades del software seguro

Antes de poder determinar las características de seguridad del software y analizar las maneras para medirlas y mejorarlas de manera efectiva, se deben definir las propiedades que las describen, que comprenden: 1) un conjunto de propiedades fundamentales cuya presencia (o ausencia) son el terreno firme que hacen seguro al software (o no); 2) un conjunto de propiedades conducentes que no hacen seguro al software en forma directa, pero que permiten caracterizar cuán seguro es.

3.1. Propiedades fundamentales

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

- *Confidencialidad*. El software debe asegurar que cualquiera de sus características (incluidas sus relaciones con su ambiente de ejecución y sus usuarios), los activos que administra y/o su contenido son accesibles sólo para las entidades autorizadas e inaccesibles para el resto.
- *Integridad*. El software y los activos que administra son resistentes y flexibles a la subversión (modificaciones no autorizadas del código, los activos administrados, la confi-

guración o el comportamiento del software por parte de entidades autorizadas). Esta propiedad se debe preservar durante el desarrollo del software y su ejecución.

- *Disponibilidad*. El software debe estar operativo y accesible a sus usuarios autorizados (humanos o procesos) siempre que se lo requiera; y desempeñarse con una performance adecuada para que los usuarios puedan realizar sus tareas en forma correcta y dar cumplimiento a los objetivos de la organización que lo utiliza.

Para las entidades que actúan como usuarios (generalmente asociadas con los usuarios finales) se requieren dos propiedades adicionales:

- *Accountability*. Todas las acciones relevantes relacionadas con la seguridad de una entidad que actúa como usuario se deben registrar y trazar a fin de poder establecer responsabilidades; la trazabilidad debe ser posible tanto durante la ocurrencia de las acciones registradas como a posteriori.
- *No Repudio*. La habilidad de prevenir que una entidad que actúa como usuario desmienta o niegue la responsabilidad de acciones que han sido ejecutadas. Asegura que no se pueda subvertir o eludir la propiedad Accountability.

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

3.2. Propiedades conducentes

Las siguientes son propiedades que, si bien no hacen directamente a la naturaleza segura del mismo, permiten caracterizarlo de alguna manera con respecto a la seguridad:

- *Dependability*. Propiedad del software que asegura que éste siempre ha de operar de la manera esperada (sin defectos y debilidades). La intencionalidad y el impacto resultante en caso de un ataque determinan si un defecto o una debilidad realmente constituyen una vulnerabilidad que incrementa el riesgo de seguridad.
- *Correcto.* Es una propiedad tan importante como las que caracterizan a la seguridad y, en ningún caso, se la debe satisfacer a expensas de otra. Generalmente su alcance está restringido a las condiciones previstas de operación; pero la seguridad requiere que se la preserve incluso bajo condiciones no previstas, que son las que se presentan cuando el software se encuentra sometido a un ataque. Si no se especifican requerimientos explícitos de un comportamiento seguro, entonces es imposible garantizar que el software correcto también sea seguro. Así, ningún requerimiento relacionado con una función, una interfase o un atributo de desempeño puede ser considerado como "correcto" si sólo se puede satisfacer a costa de permitir que el software se comporte de manera insegura, o que impida determinar o predecir si se ha de comportar de manera segura.
- *Predecible*. Las funcionalidades, propiedades y comportamientos del software siempre serán los que se espera que sean, tal que en todo momento las condiciones bajo las que opera (es decir, el ambiente, las entradas que recibe) también sean predecibles; así, el software nunca se desviará de su operación correcta bajo condiciones previstas. Esta propiedad extiende la seguridad del software a su operación bajo condiciones no previstas, en las que los atacantes intentan explotar fallas en el software o en su ambiente.
- *Confiable.* Orientada a preservar la ejecución predecible y correcta del software a pesar de la existencia de defectos no intencionales y otras debilidades, y de cambios de estado no predecibles en el ambiente.
- *Protección.* Depende de la propiedad anterior y, por lo general, el hecho de no satisfacerla plantea implicancias muy significativas. Cuando el sistema ha sufrido una fa-

lla o un ataque, debe detenerse o quedar parcialmente operativo en un estado seguro de tal forma de impedir catástrofes (pérdida de vidas humanas, pérdidas de activos valiosos, compromiso de la sustentabilidad del ambiente, etc.).

En el contexto de las dos últimas propiedades, las fallas se consideran aleatorias e impredecibles, mientras que en un contexto de seguridad, son el resultado de un esfuerzo humano (directo, o a través de código malicioso). Es la falta de malicia lo que hace, de alguna manera, diferentes los requerimientos asociados a los términos en inglés *safety* y *reliability* respecto de lo que se conoce como la seguridad del software**.**

4. Modelos y métodos del 'Ciclo de Vida del Desarrollo de Software' (SDLC)

Una metodología de desarrollo de software que incorpore mejoras en el sentido de incluir la seguridad, debe proporcionar un marco integrado y fase por fase para promover el desarrollo de la seguridad del software a lo largo de todo el ciclo de vida.

Como se mencionó en la introducción ha habido avances en este sentido, si bien no se identifica una metodología particular. En general se han adicionado principios, prácticas y actividades vinculadas a la seguridad a métodos existentes. La definición de requerimientos resulta crítica, y en un proceso de desarrollo de software mejorado para la seguridad se compensan las insuficiencias respecto a este atributo existentes en los requerimientos del software mediante el agregado de prácticas y la verificación de su adecuación en todas las fases del ciclo. La idea que sustentan las diferentes metodologías propuestas por diversos autores es que se apliquen las mejores de prácticas de seguridad al conjunto de artefactos de software que se crean a lo largo de todo el SDLC. En la mayoría de ellas se propone que de la aplicación de estas prácticas resulte un proceso neutral, que se pueda utilizar con una amplia variedad de modelos de desarrollo. Además, considerando el SDLC, los controles de seguridad no se deben limitar a las fases de requerimientos, diseño, codificación y testeo; es necesario continuar realizando revisiones de código, pruebas de seguridad, controles de configuración, y aseguramiento de la calidad durante el despliegue y la operación, para garantizar que las actualizaciones y los parches no agreguen debilidades de seguridad o de lógica maliciosa.

En los últimos años, se han realizado diferentes intentos para definir el modelo más efectivo para organizar las diferentes actividades técnicas, de administración y de organización del SDLC. Entre estas propuestas se destacan: a) *Modelo en cascada* [11], [12]; b) *Modelo iterativo e incremental* [13,14]; c) *Modelo en espiral* [15]; d) *Modelo unificado* [16, 17, 18].

Por otra parte, una metodología para el desarrollo de software seguro debe proporcionar un framework. En algunos casos, se han introducido modificaciones en las actividades tradicionales, mientras que en otros se han insertado actividades con el objetivo de reducir el número de defectos y errores que deriven en vulnerabilidades del software y en los servicios que lo utilizan. Se observa la existencia de metodologías que están siendo empleadas con distintos grados de éxito en proyectos de desarrollo de productos o de pilotos.

- *Microsoft Trustworthy Computing SDL* [19]. Esta metodología está vigente y en evolución, y está disponible información detallada y extensiva para quienes deseen adoptar el enfoque. Propone como objetivos principales: 1) reducir el número defectos y/o errores de diseño y de código relacionados con la seguridad de los productos Microsoft; 2) reducir la severidad del impacto de cualquier defecto residual.

- *Oracle Software Security Assurance Process* [20]. Es un proceso interno, no difundido externamente, existe muy poca documentación al respecto.
- *Comprehesive, Lightweight Application Security Process (CLASP)* [21]. Diseñado para aplicar en cada fase del ciclo de vida metodologías de seguridad. Comprende un conjunto de actividades enfocadas en la seguridad que pueden integrarse en cualquier proceso de desarrollo de software. Ofrece documentación completa sobre dichas actividades, los riesgos de no incluirlas, y un catálogo de vulnerabilidades. Está disponible como un plug-in para *Rational Unified Process* y liberado bajo licencia 'open source'
- *Siete 'touchpoints' para Seguridad de Software* [8]. Propuesta de un conjunto de mejores prácticas a ser aplicadas en varios artefactos del desarrollo de software.
- *TSP-Secure* [22]. Es una extensión a '*Team Software Process*' (TSP), del que no existe información más que una presentación realizada en 2006 por el *Software Engineering Institute* (SEI).

También caben mencionar algunos proyectos del ámbito académico que trabajan sobre metodologías, cuya efectividad aún no se ha podido verificar mediante su aplicación en proyectos de desarrollo: *Appropriate an Effective Guidance In Information Security* (AEGIS) [23], *Rational Unified Process Secure* (RUPSec) [24], *Secure Software Development Model* (SSDM) [25], *Waterfall-Based Software Security Engineering Process Model* [11], [26], *Extensiones de Seguridad propuestas a MBASE* [27], y *Secure Software Engineering* (S2e) [28].

En este conjunto de metodologías, en las que se observa continuidad son *Microsoft Trustworthy Computing SDL*, *CLASP, AEGIS*, *SSDM* y *S2e*.

Analizando estas metodologías y comparándolas, tomando como referencia el conjunto de fases comunes a la mayoría de los SDCL a las que se asocian las nuevas actividades introducidas, surgen las siguientes tendencias:

1. Se incorporan actividades en todas las fases; no obstante se observan ciertas diferencias en la naturaleza de las mismas dependiendo de la metodología en particular.
2. Se incluyen otras actividades de carácter transversal durante: a) fase de planificación: ejecutar un programa de concientización (de la organización, del personal y de los equipos de desarrollo) sobre la criticidad de la seguridad para la continuidad del negocio; b) conjunto de fases: ejecutar monitoreo y seguimiento centrado en el riesgo y su análisis; c) fase de mantenimiento y soporte: prestar servicios de respuesta a incidentes de seguridad, y gestionar un proceso de descubrimiento de problemas de seguridad.
3. Existe un conjunto común de artefactos propuestos para la ejecución de muchas de las actividades propuestas, entre los que se destacan: a) resultados de un proceso de análisis de riesgo; b) casos de mal uso y de abuso; c) árboles de ataque para el modelado de amenazas; d) *UML Secure* y Patrones de ataque; e) codificación segura, f) pruebas de penetración y pruebas creativas.

En la Figura 1 se muestra un diagrama en el que se resume las principales actividades y artefactos que se incorporan en las metodologías comparadas.

5. Aseguramiento del Software y Aseguramiento de la Calidad

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

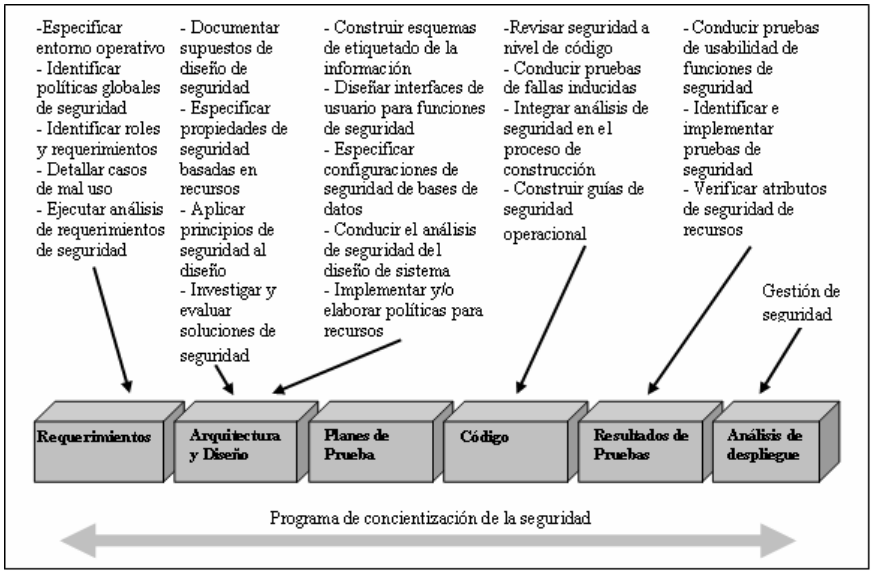

Figura 1: Actividades y Artefactos en las distintas metodologías

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

Como ejemplos de abordajes en la primera dirección se pueden destacar:

- *Standard of Good Practice* [29]: QA para prácticas de desarrollo de sistemas de información y de software seguro; establece la necesidad de respetar durante el SDLC las pautas del aseguramiento de la calidad en las principales actividades relacionadas con la seguridad, a fin de garantizar que los requerimientos de seguridad se definan adecuadamente y se desarrollen los controles para satisfacerlos.
- *In Best Practices on Incorporating QA into Your Software Development Life Cycle* [30]: Se identifica un conjunto de métricas para la gestión del riesgo de la seguridad, las que se incluyen en el esquema de trabajo de QA en las diferentes fases del SDLC. Como ejemplo: 1) fase de requerimientos: a) identificación de niveles aceptables del tiempo total de fuera de servicio y de pérdida de datos, b) identificación de los requerimientos de seguridad; 2) fase de diseño: a) identificación y provisión de contramedidas para las vulnerabilidades, b) diseño que refleje las necesidades de las actividades de gestión de incidentes y recuperación ante desastres, y la capacidad de realizar pruebas.

En cuanto a abordajes en el segundo sentido, recientemente se ha publicado un interesante aporte proveniente del campo de la industria:

- *Building Security In Maturity Model* (BSIMM) [31], *Modelo de madurez* diseñado para ayudar a comprender y planificar una iniciativa de seguridad de software. Se trata de un modelo creado a partir del relevamiento y análisis de datos de nueve iniciativas destacadas de seguridad de software; dicho proceso permitió identificar que,

aunque las metodologías particulares aplicadas se diferencien (sea *Touchpoints, Microsoft SDL* o *CLASP*), existen muchos puntos en común, los que son capturados y descritos en el modelo. Éste no es una guía completa de "cómo manejar" la seguridad de software, sino una colección de buenas prácticas y actividades utilizadas hoy.
Como estructura de organización, introduce el *Software Security Framework* (SSF), que proporciona un andamio conceptual para el modelo. Este esquema de trabajo reconoce 12 Prácticas en 4 Dominios:

- *Gobernanza*: Prácticas que ayudan a organizar, manejar y medir una iniciativa de seguridad de software, entre las que el desarrollo de personal se considera central;
- *Inteligencia*: Prácticas que recolectan el conocimiento que posee la organización en su conjunto para realizar actividades de seguridad del software, expresado como la guía proactiva de seguridad y como el modelo de amenazas de la organización;
- *SSDL Touchpoints*: Prácticas asociadas con el análisis y el aseguramiento de los artefactos y los procesos de desarrollo de software;
- *Despliegue*: Prácticas relativas al mantenimiento y aseguramiento de la infraestructura.

BSIMM identifica para cada Práctica, 3 Niveles de Madurez, proporcionando una progresión natural a lo largo de las actividades asociadas con cada práctica. Así, mediante la identificación de los Objetivos y las Actividades correspondientes a cada Práctica que se aplique, y garantizando un balance apropiado con respecto a los Dominios, es posible crear un plan estratégico para cada iniciativa de software seguro.

6. Resultados de la aplicación de las Metodologías Vigentes

Los resultados que se observan en la creación y/o actualización de las metodologías para SDLC que soportan el desarrollo de productos con garantías de seguridad resultan alentadores, dado los pocos años que median desde que las comunidades de la Ingeniería de Calidad y de la Ingeniería de la Seguridad comenzaran a afrontar el problema común.
En la actualidad, el mayor de los retos lo constituye la consolidación de taxonomías concernientes a seguridad y su relación con las ya existentes en el dominio de la calidad del software. A esto se agregan los desafíos que plantea la necesidad de trasladar estas taxonomías al ámbito de los diferentes idiomas, proceso que requiere del esfuerzo de los especialistas de seguridad orientado a facilitar no sólo la comunicación entre pares sino también con las comunidades de desarrolladores.
Con relación a la evolución de modelos y de metodologías, también son muy importantes los logros que comienzan a percibirse, en especial acompañados de herramientas que les permitan a los equipos avanzar en su explotación sin necesidad de atravesar períodos prolongados de capacitación. Esto favorece las posibilidades de establecer mecanismos fluidos de retroalimentación y también derivar esfuerzos hacia programas de concientización respecto al impacto que tendrá en las organizaciones y en la industria del software la posibilidad de producir software de calidad, extendida a la seguridad.
Dado el estado del arte, quienes deben tomar decisiones relativas a la adopción de métodos y metodologías de desarrollo de software en las organizaciones aún no cuentan con estándares para respaldar decisiones de compromiso. No obstante, dentro de un programa de mediano plazo, resulta ineludible su consideración.

7. Conclusiones

Las experiencias están demostrando que resulta necesario una disciplina de la ingeniería que conforme el fundamento para comprender en profundidad las cuestiones de la seguridad en el desarrollo de sistemas de información, proporcione el conocimiento apropiado para asistir a los ingenieros en sistema de información y en seguridad, y eduque a los usuarios de sistemas en aspectos relaciones con la seguridad de los sistemas de información.
Conforme las tendencias reseñadas, se considera estar ante el surgimiento de lo que algunos investigadores proponen denominar como Ingeniería del Software Seguro [32], cuyo alcance comprende, entre otras, a la ingeniería de requerimiento de seguridad, el modelo de seguridad y el desarrollo de software seguro. Su principal objetivo como campo de investigación es la producción de técnicas, métodos, procesos y herramientas que integren los principios de la ingeniería de seguridad y de calidad, y que le permitan los desarrolladores de software analizar, diseñar, implementar, testear y desplegar sistemas de software seguro. El desarrollo de dichas técnicas se debe basar en los resultados obtenidos por la comunidad de investigadores en ingeniería de seguridad, complementados con los resultados provistos por la comunidad de ingeniería de software (técnicas de ingeniería de requerimientos, metodologías de desarrollo de software y lenguajes de modelado, y testeo).
Esta nueva área de la Ingeniería ofrece varias ventajas, entre las que se destacan:

1. Permitir el desarrollo de mejores técnicas relacionadas con la seguridad y de mejores definiciones de los esquemas de trabajo metodológicos;
2. Ofrecer el basamento para una ontología de la seguridad completa y reconocida, que le permita a los desarrolladores considerar no sólo los desafíos tecnológicos relativos a la seguridad, sino también las implicancias sociales derivadas.

Referencias

1. Department of Homeland Security. National Strategy to Secure Cyberspace. Action-Recommendation 2-14. 2003.
2. Committee on National Security Systems. “National Information Assurance (IA) Glossary”. CNSS instruction No. 4009. Revised 2006.
3. M. Komaroff and K. Baldwin. “DoD Software Assurance Initiative”. 2005.
4. National Aeronautics and Space Administration (NASA). “Software Assurance Standard”. Standard No. NASA-STD-2201-93. USA. 1992.
5. US Computer Emergency Response Team, “Build Security In”. USA. [https://buildsecurityin.us-cert.gov]
6. National Institute of Standards and Technology, “SAMATE—Software Assurance Metrics and Tool Evaluation”. USA. [http://samate.nist.gov]
7. K. Goertzel, T. Winograd, H. McKinley, L. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau. “Software Security Assurance - State-of-the-Art Report”. Information Assurance Technology Analysis Center & Data and Analysis Center for Software. USA. 2007.
8. G. McGraw. “Software Security: Building Security In”. Addison-Wedsley. USA. 2006.
9. K. Goertzel, T. Winograd, H. McKinley, P. H. and B. Hamilton. “Security in the software lifecycle. Making Software Development Processes -and Software Produced by Them- More Secure”. Draft V1.2. Department of Homeland Security. USA. 2006.
10. S. Romaniz. “Buenas prácticas de elicitación de los requerimientos de seguridad”. IV Congreso Iberoamericano de Seguridad Informática CIBSI 2007. Argentina. 2007.
11. W. Royce, “Managing the Development of Large Software Systems, Concepts and Tech-

niques". Proceedings of IEEE. USA. 1970.
12. "Defense System Software Development: MIL-STD-2167A". [http://www2.umassd.edu/SWPI/DOD/MIL-STD-2167A/DOD2167A.html]
13. M. Yatoco. "Joint Application Design/Development". 1999. [http://www.umsl.edu/~sauterv/analysis/JAD.html]
14. J. Radatz, M. Olson and S. Campbell. "CrossTalk: The Journal of Defense Software Engineering, MIL-STD-498. USA. 1995.
15. B. Boehm, et al., "A Software Development Environment for Improving Productivity". IEEE Computer 17, no. 6. 1984.
16. IBM/Rational, "Rational Unified Process Best Practices for Software Development Teams". Rational Software White Paper, no. TP026B Rev 11/012001. USA. 2001.
17. S. Ambler. "The Agile Unified Process". Canada. 2006. [http://www.ambysoft.com/unifiedprocess/agileUP.html]
18. S. Ambler. "Enterprise Unified Process (EUP)". Canada. 2006. [http://www.enterpriseunifiedprocess.com/]
19. S. Lipner and M. Howard. "The Trustworthy Computing Security Development Lifecycle". USA. [http://msdn.microsoft.com/en-us/library/ms995349.aspx]
20. "Oracle Software Security Assurance". Redwood Shores, CA: Oracle Corporation. [http://www.oracle.com/security/software-security-assurance.html]
21. Fortify Software Inc. "CLASP: Comprehensive, Light Application Security Process". ISA. [http://www.fortifysoftware.com/security-resources/clasp.jsp]
22. J. Over (CMU SEI). "TSP for Secure Systems Development". [http://www.sei.cmu.edu/tsp/publications/tsp-secure.pdf]
23. I. Flechais, C. Mascolo and M. A. Sasse. "Integrating Security and Usability into the Requirements and Design Process". Proceedings of the Second International Conference on Global E-Security, London, UK. 2006.
24. P. Jaferian et al. "RUPSec: Extending Business Modeling and Requirements Disciplines of RUP for Developing Secure Systems". Presentation at the 31st IEEE EuroMicro Conference on Software Engineering and Advanced Applications. 2005.
25. A. Sodiya, S. Onashoga and O. Ajayi. "Towards building secure software systems". Proceedings of Issues in Informing Science and Information Technology. England. 2006.
26. M. Zulkernine and S. Ahamed. "Software Security Engineering: Toward Unifying Software Engineering and Security Engineering". Chap. XIV in Enterprise Information Systems Assurance and System Security: Managerial and Technical Issues, Merrill Warkentin and Rayford B. Vaughn, eds. Hershey, PA: Idea Group Publishing. 2006.
27. D. Wu, I. Naeymi-Rad and E. Colbert. "Extending Mbase to Support the Development of Secure Systems," in Proceedings of the Software Process Workshop. China. 2006.
28. Secure Software Engineering. [http://www.secure-software-engineering.com/]
29. Information Security Forum. "The Standard of Good Practice for Information Security". vers. 4.1. Information Security Forum. UK. 2005.
30. K. Sadovsky, C. Roode and M. Arseniev. "Best Practices on Incorporating Quality Assurance into Your Software Development Life Cycle". EDUCAUSE. USA. 2006.
31. G. McGraw, B. Chess and S. Migues. "Building Security In Maturity Model" vers. 1.0. Cigital, Inc. and Fortify Software. 2009. [http://www.bsi-mm.com/]
32. H. Mouratidis and P. Giorgini. "Integrating Security and Software Engineering". Idea Group Publishing. USA. 2007.