Diseño de software seguro

Manuel Dávila Sguerra

El software es un elemento catalogado de seguridad nacional en los países que tienen un plan de desarrollo orientado hacia la sociedad del conocimiento. La preocupación en esos países se basa en el hecho de que la Ingeniería de software no ha logrado que los desarrollos de software cumplan con la característica denominada "trusworthy", es decir, digno de confianza. Este artículo pretende recoger una serie de especificaciones que contribuyan a que se cumpla el desarrollo de software seguro. No es la seguridad perimetral la que preocupa en esta investigación, es el software en sí.

Para ello se ha consultado la estrategia de desarrollo de software de los Estados unidos para el 2015, se resumen los planteamientos de Gary McGraw, uno de los líderes en la propuesta de metodologías de software seguro, y se hace una propuesta para incluir en el ciclo de desarrollo las etapas necesarias para lograr estos desarrollos.

Finalmente, haciendo uso de los referentes de ACM e IEEE sobre los diseños curriculares para crear programas de Ingeniería de sistemas o afines, se proponen algunos puntos importantes para incluir en los nuevos programas la formación en este tipo de competencias.

Palabras clave: Software seguro, trustworthy, diseño curricular, Ingeniería de sistemas.

1. Introducción

En los países orientados a conformar sociedad del conocimiento y que tienen en el desarrollo de software un compromiso como industria, se le considera a éste un producto de seguridad nacional. Es el caso de los Estados Unidos que en el reporte de la segunda cumbre del Centro Nacional de Software así lo presenta, en un documento llamado "Software 2015: A national software strategy to ensure U.S. Security and competitiveness" [1]

Consideran que su nación es altamente dependiente de las tecnologías de la información siendo el punto central el software, el cual se encuentra en todos los elementos de la vida cotidiana de las personas y las organizaciones. Determinan que su país es una nación bajo riesgo de consecuencias inaceptables por las fallas del software. Estas fallas pueden crear serios problemas en varios riesgos, enumerando específicamente la infraestructura nacional, la economía, la vida de las personas, la pérdida de confianza pública, de la identidad y liderazgo.

Si bien nuestros países no se comparan con la capacidad productiva de software

que tiene Estados Unidos, es de mucha utilidad conocer esos planes, pues de todas maneras las tecnologías informáticas tienen carácter global. Siendo innegable la importancia del software, así como su penetración en la vida cotidiana se ve la necesidad de revisar el estado de la Ingeniería de software, la cual en sus 50 años de existencia aún no ha podido resolver el riesgo de que un avión se caiga por culpa de un *overflow* en una variable.

Vale la pena anotar que el nivel de seguridad que se necesita en el diseño de los programas va más allá de la tradicional seguridad perimetral, para entrar en el campo del diseño de las aplicaciones.

Este estudio ha contemplado las premisas de la estrategia de los Estados Unidos como referencia para entrar en una discusión de índole más local.

2. Software digno de confianza o trustworthiness

Se introduce el término *trusworthiness* o la cultura sobre las características que debe cumplir el software para ser "trustworthy" o sea, digno de confianza. Estas características se resumen en: seguridad, confianza, fiabilidad y supervivencia. El significado de cada una de estas características se detalla a continuación.

Seguridad: se refiere a la exposición de los programas a peligros no intencionados en sus especificaciones; es decir, que no se dan por mala Ingeniería de software o malas prácticas en la programación, sino por aspectos de índole externo. Cabe hablar aquí de código malicioso y malintencionado como son los virus, troyanos, puertas traseras, spam lo que no sólo crea problemas reales de disminución de confianza, sino que bajan la productividad porque el tiempo de control y revisiones de los daños es muy grande, pudiendo usarse en otras labores más productivas. Una persona que gaste una hora diaria en revisar estos problemas equivale a un mes laboral del año.

Confianza: la vida normal de las personas transcurre con el uso de elementos o artefactos que dependen del software y que su uso debe generar confianza para una tranquilidad de índole social. El software inseguro puede llegar a hacer daño a las personas y a la propiedad en ambientes de uso cotidiano como la aviación, la medicina, la exploración espacial, la banca y en general el transporte. Obsérvese cómo en estas actividades está en peligro la vida de las personas, no sólo una funcionalidad con repercusiones económicas.

Fiabilidad: en algunos casos el software participa de soluciones de gran escala que pueden afectar de manera masiva a la sociedad, como la defensa nacional, las telecomunicaciones, la energía, el espacio y los sistemas financieros. El hecho de utilizar satélites para las telecomunicaciones y el transporte de los datos nos puede afectar.

Supervivencia: ese término se definió como una característica que asegure que el software se debe mantener en continuo funcionamiento, aún en situaciones adversas y no sólo en ambientes benignos. El reciente drama del Japón con el terremoto y el tsunami dan que pensar en cuanto a lo que se puede catalogar una situación adversa, pues los centros de datos deben contemplar estas situaciones y el software mismo también.

Pasemos ahora a determinar una definición formal para la palabra "Trustworthi-

ness". Lo más cercano que he encontrado en español para traducir este término es "Digno de confianza"; es decir, que debe cumplir con todos los requerimientos, inclusive con los de seguridad, en cualquiera de los componentes de software, aplicación, sistema o red.

Un desarrollo de software *digno de confianza,* tiene atributos de "seguridad, garantía, confianza, fiabilidad, rendimiento, supervivencia, con un amplio espectro de adversidades y compromisos. Se requiere en el hardware, en el software, en las comunicaciones, en componentes de potencia, así como en los desarrolladores y los que hacen el mantenimiento". La inclusión del factor humano en estas taxonomías dan que pensar en cuanto a la formación integral de los profesionales en las universidades en donde los aspectos de tipo ético cobran aún más valor.

La estrategia contempla en primer lugar la necesidad de que a la fuerza de trabajo, es decir a los desarrolladores, se les eduque permanentemente y de les defina una estructura salarial adecuada y competitiva.

Le da a la investigación y al desarrollo la verdadera dimensión para fortalecer esta industria invitando a que las Universidades y las empresas trabajen de manera mancomunada, así como el Estado, que debe contribuir desde las instituciones encargadas de diseñar leyes para fomentarlas. Se da una especial atención a los procesos creativos que según García Córdoba en su libro sobre la Investigación aplicada y tecnológica, los define como "la capacidad de generar soluciones finales desde ángulos insospechados" [2]

El estudio determina que no son las buenas intenciones las que lograrán crear software *digno de confianza*, sino una formación profesional y una intención permanente que debe ser involucrada en el ejercicio profesional de los profesionales. Pero no como una opción, sino como una necesidad. Lo anterior requiere entonces de mejores diseños en las métricas de calidad, en el proceso evaluativo, en las metodologías a utilizar en el futuro, en la relación con las empresas y el Estado, desde las universidades.

3. Metodologías de desarrollo tradicionales

La creación de software se orienta por metodologías que se han venido construyendo con el tiempo; y, pensando que esta concesión aporta académicamente al artículo y a la contextualización del problema central, mencionaremos algo al respecto. Sin importar cuál metodología se use, el modelo que presentaremos al final para el desarrollo del software seguro debe ser tenido en cuenta.

Las metodologías más conocidas son:

Modelo Construir y mejorar (1950 – 1960):

Utilizado por la Volskwagen en la producción y venta de sus vehículos cuyo esquema promueve que el artefacto terminado se use, y se recopilen las fallas detectadas por los usuarios para mejorarlo. Hoy en día, a pesar de su popularidad, recibe justificadas críticas .

Método en cascada (1970):

Esta fue la época en que Edsger Dijkstra creó la programación estructurada y funciona si cada fase está perfectamente desarrollada, lo cual casi nunca se cumple. Propone un desarrollo secuencial.

Modelo prototipo rápido:

Se basa en el modelo de las plantas piloto de los Ingenieros Químicos y va produciendo el programa con las funciones esenciales para ir mejorándolo, en la medida en que el usuario los acepta. Es de anotar la presencia del usuario un poco más involucrado en el proceso.

Modelo incremental:

Es una mezcla del modelo en cascada y del prototipo rápido y ya reconoce que los pasos en el desarrollo no son discretos, y va creando construcciones paulatinas, con el peligro de que el proceso de aprendizaje exceda al de la productividad y se de el "síndrome de la investigación".

Modelo Extreme Programming:

Se basa en el desarrollo del sistema de nómina de Chrysler y se involucra al usuario o cliente cuyo "discurso" es recibido en su lenguaje original y se toma como parte del desarrollo, el cual se hace de manera incremental. La Chrysler dice "se monta al usuario en el desarrollo" como una metáfora relacionada con montarlo en el vehículo, pero que significa tenerlo en cuenta de manera preferencial.

Modelo Round Tripping:

Se soporta en los generadores de código basado en diseño de patrones. Está muy orientado a la Programación orientada a objetos.

Modelo Iterativo RUP:

Se considera uno de los más realistas, pues hace seguimiento entre cada estado y el anterior.

El modelo tradicional, que envuelve a la gran mayoría de las metodologías, contempla los siguientes ciclos: especificaciones o modelo funcional, diseño o arquitectura, programación, pruebas, documentación, entrenamiento y mantenimiento. Se acostumbra a expresar la Complejidad del software como una función del tipo de programa (*N*), el número de entradas (*I*), el número de salidas (*O*), y una potencia *p* de tal manera que:

Complejidad = N*I*(O elevado a la potencia p) [3]

4. Seguridad de las aplicaciones y seguridad perimetral

Hasta el momento la seguridad perimetral ha seguido evolucionando cada vez con mayor fuerza. Las gerencias de infraestructura informática tienen más en cuenta la necesidad de incluir proxys y firewall para filtrar el tráfico, tanto al nivel de las URL como de los puertos y servicios de la red. Estos profesionales tienen conocimientos sobre redes y algunos sobre seguridad, ciframiento, certificados de seguridad, detección de intrusos, virus, gusanos, troyanos y de más software intrusivo que llega a la red.

Sobre esto sólo se puede decir que seguirá evolucionando como lo ha hecho, centrando su atención en la capa baja del modelo OSI. En la capa alta están las aplicaciones, a las cuales llegan los usuarios y a veces los intrusos que han logrado pasar los filtros de la seguridad perimetral.

Dice el libro "Computer Crime" de la editorial O'reilly que, en la gran mayoría de los casos, los intrusos entran no por complejos conocimientos de redes, sino por el simple hecho de haber podido atrapar las claves de seguridad de los usuarios. [4]

Cualquiera que sea el caso, surge la pregunta sobre la forma como se protegerá la aplicación a sí misma, bien sea por adecuados trabajos en la etapa de diseño, por fuertes procedimientos de escritura del código fuente o por adecuados análisis de riesgo en la interacción con la plataforma tecnológica. Lo anterior lleva a concluir que es necesario incluir dentro del ciclo de desarrollo del software, cualquiera que sea la metodología utilizada, tareas relacionadas con la seguridad, no ya perimetral, sino de las aplicaciones mismas.

Un inconveniente es el hecho de que los profesionales de las redes y la seguridad de hoy en día, no son los mismos de la Ingeniería de software. Es claro que el diálogo se llevará a nivel de la capa de desarrollo, lo que necesita profesionales aceptados por quienes hacen las aplicaciones; es decir, analistas, arquitectos, programadores y demás especialistas.

La formación de este tipo de profesional es nueva y puede surgir de una mezcla de las alternativas de formación que plantea la ACM y la IEEE en el computing curricula 2005. Este referente plantea cinco procesos formativos que son: Ingeniería de computadores, Ciencia de computadores, Ingeniería de software, Sistemas de información y Tecnología de la información.

No está dentro de este estudio el análisis de este referente, pero basta con decir que el perfil del profesional que se vislumbra para intervenir en el ciclo de desarrollo de software seguro puede estar en una mezcla bien diseñada curricularmente entre Ingeniería de software y Tecnología de la información.

Los diagramas de Computing Curricula de estas dos orientaciones se muestran a continuación, aclarando que el eje horizontal indica nivel de conocimiento desde lo teórico (izquierda) a lo práctico (derecha), y el eje vertical va desde abajo a arriba indicando áreas de profundización, es decir, desde el chip, la electrónica, la plataforma, el software, los lenguajes, las metodologías, hasta la implementación. Bajo esas consideraciones la ACM y la IEEE consideran cinco énfasis sobre los cuales se pueden formar los profesionales. Sin entrar en grandes detalles y dejando al lector la indagación de la lógica de estas gráficas, cada una de las cuales tiene un currículo sugerido, nos referiremos a: ***CE***: Computer Enginering. Forma profesionales que van a diseñar computadores. ***CS***: Computer Science. ***SE***: Software Engineering. ***IS***: Information System. ***IT***: Information Technology.

El gráfico abajo a la derecha sólo insinúa que una Universidad puede diseñar el perfil de los profesionales dependiendo de sus objetivos, como una mezcla de estos énfasis.

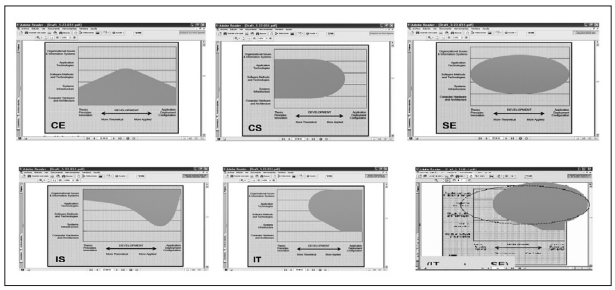

Figura 1. Los énfasis de la Ingeniería de sistemas según ACM e IEEE [6]

En el tema que estamos trabajando, es decir el software seguro, requiere profesionales expertos en SE - Ingeniería de software y con conocimientos sólidos de redes y seguridad IT: Tecnología de la Información.

5. Metodología para diseñar software seguro

Gary McGraw, autor del libro Software Security [5], propone una metodología para incluir dentro del ciclo de desarrollo, la cual puede ser introducida en cualquiera de los modelos de desarrollo que hemos mencionado, pero cabe anotar la importancia de reconsiderar el mecanismo cíclico de cada etapa, para evitar aplicar modelos netamente secuenciales.

En la figura 2 se muestra el ciclo de desarrollo incluido el análisis de la seguridad.

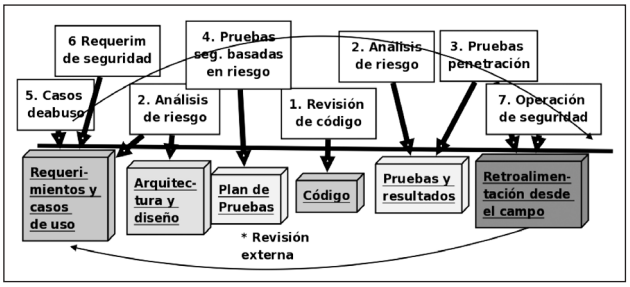

Figura 2. Análisis de la seguridad en el ciclo de desarrollo [7]

Revisión de código con herramientas: el artefacto es el código al cual se le aplica un análisis estático del código fuente, para encontrar riesgos probables. Por ejemplo, buffer overflow encontrado en la línea 8. Este análisis estático busca instrucciones no seguras, pero como todos, no es suficiente por sí solo. ***Análisis de riesgo de la arquitectura:*** el artefacto es el Diseño y especificaciones al cual se le aplica un análisis de los posibles ataques como pueden ser problemas de autenticación o fallas en consideraciones de acceso a la web. ***Pruebas de penetración:*** el artefacto es el ambiente del sistema al cual se debe practicar pruebas de "cajas negras" producidas por otros sistemas, no solamente las pruebas bien conocidas. Un ejemplo puede ser un manejo pobre de la interfaz. ***Pruebas de seguridad basado en riesgos:*** los artefactos son unidades y sistema. Se refiere a los posibles escapes de datos, en donde la mejor actitud es pensar como un atacante. ***Casos de abuso:*** los artefactos son los casos de uso y los requerimientos. Se debe hacer una descripción de las susceptibilidades del sistema bajo ataques bien conocidos. ***Requerimientos de seguridad:*** los artefactos son los requerimientos. Se deben tener en cuenta aspectos como cifrado, certificados de seguridad y en general seguridad funcional. Un riesgo es adolecer de esa descripción. ***Operaciones seguras:*** el artefacto es el sistema en funcionamiento. Se asume que los ataques son inevitables, por lo que se debe tener información para conocer y reconstruir los eventos; por ejemplo, con bitácoras adecuadas.

6. Pérdida de calidad

La calidad de los resultados es el objetivo final del desarrollo de software. Ésta tiene muchas definiciones y mecanismos para lograrla, sin embargo se expondrán los análisis de los autores Bijay K. Jakaswal y Peter C. Patton, autores del libro Design for Trusworthy software [7] basados en las teorías del Dr Genichi Taguchi, japonés nacido en 1924, quien expuso su teoría sobre la pérdida de calidad (QualityLoss), la que define como "La pérdida llevada por el producto a la sociedad desde el momento en que es empaquetado".

El concepto fundamental se basa en la idea de acercarse lo más posible al objetivo propuesto, de tal manera que la variación del resultado final con respecto a él sea la mínima posible y que los niveles de variabilidad conllevan costos de reproceso, mantenimiento, fallas, reclamos, rendimiento y fiabilidad. Se trata de medir este costo como una función cuadrática de la desviación con respecto a la finalidad. Y aplicar una metodología que incremente el control de variables externas a través de lluvia de ideas, investigación y metodologías formales. Insiste, el Doctor Taguchi, en la parametrización que permita ejercer control sobre variables sensibles y ser tolerantes en el sentido de permitir fijar valores a algunos parámetros sobre realidades comprobadas.

Para comprender el tema, el Dr Taguchi incluye una serie de definiciones en su lenguaje que ayudan a comprender sus postulados sobre la calidad como son:

Robustez*:** "un estado en donde el rendimiento de la tecnología, producto o proceso es mínimamente sensitivo a factores que causen variabilidad al menor costo unitario de manufactura". ***Señal: "es lo que un producto debe mostrar por las características de sus funcionalidades". Sonido en un radio, imagen en un televisor, funcionalidades comprobadas en un paquete de software. ***Ruido:*** "factores que causan variaciones externamente, internamente o entre productos". Interferencias en la imagen de un televisor o uso errático por parte del usuario de un paquete de software, ataques, virus, gusanos y huecos de seguridad, poca documentación, entrenamiento inadecuado, malos procedimientos o malos usos del sistema, accesos no autorizados, sometimiento del sistema a usuarios para los cuales no fue diseñado.

Teniendo en cuenta estas definiciones Taguchi resume en algunas premisas el método: "La pérdida de calidad se debe más a fallas después de ventas"; "La robustez de un producto depende más de la etapa de diseño, que del control durante su funcionamiento"; "no liberar nada que no cumpla los estándares"; "no usar medidas de calidad basadas en el usuario"; "los productos robustos producen una 'señal' fuerte sin importar el 'ruido' externo y con un mínimo de 'ruido' interno".

Se sabe que el Señor Deming, llamado el padre del movimiento de la calidad moderna, fue seguido por Taguchi en algunas de sus teorías y propone también una serie de premisas aplicables al desarrollo de software como estas: "atender la voz del usuario o consumidor, reducción de la variación, medidas estadísticas, ganancia de confianza, respeto por los cotrabajadores, mejora continua en el proceso".

Y la esencia de los puntos de Deming que también son aplicables al caso del software son: "constancia en el propósito"; "evitar dependencia de las inspecciones construyendo calidad desde el principio"; "entrenamiento permanente"; " liderazgo más que supervisión ";

"eliminar incentivos por cuotas y cambiarlas por liderazgo"; "eliminar administración por objetivos cuantificados y cambiarlos por liderazgo"; "eliminar los premios por méritos anuales cambiarlo por administración por objetivos"; "educación y auto mejoramiento".

7. Conclusiones

La experiencia vivida en esta investigación he permitido ir creando una metodología formal para que al hacer un proyecto de desarrollo de software sea tenida en cuenta la seguridad en el software. Académicamente, nos permite transmitir a través de conferencias a los estudiantes estas tendencias. Es de esperarse que las asignaturas de Ingeniería de software le den más importancia, en las universidades, a estas innovaciones.

Se corrobora nuevamente que la Ingeniería de software requiere mejoras permanentes y que aún depende demasiado de las destrezas de los diferentes profesionales que actúan en un desarrollo de software. Nos deja las expectativas de ver cómo en los Estados Unidos se resolverá el problema de la Ingeniería de software y nos invita, a profundizar en el desarrollo "digno de confianza".

Dos experiencias se han podido concretar en resultados. Una, la de proponer una metodología basada en lo propuesto por Gary McGraw, otra el desarrollo de un algoritmo que permite detectar intrusos, no en la red sino en la aplicación misma, el cual ha sido implementado en la plataforma *e-Genesis – El generador de sistemas* de mi autoría y con el desarrollo de un método para hacer autodocumentación de software, temas que podrán expandirse en otra publicación.

8. REFERENCIAS

[1] Report of the 2nd National Software Summit, software 2015: A national software strategy to ensure u.s. security and competitiveness Report of the 2nd National Software Summit Center for National Software Studies. Center for National software studies, Reston, 2005, pp.1-32

[2] Córdoba G, La investigación tecnológica, Limusa Noriega Editores, pp 216,, 2005

[3] Patton P, Jayaswal B., Design for trustworthy Software, Prentice Hall, pp 7, 2006

[4] Icove D. VonStorch W., Computer Crime, O'Reilly and Associated Inc, pp51, 1995

[5] G. McGraw, "Software security Building security in", 1er. ed., Addinson Wesley", pp. 3-176, 2006

[6] ACM-IEEE, Computing Curricula, Snapshots: Graphical Views of the Computing Disciplines, ACM-IEEE, pp16,2005

[7] G. McGraw, "Software security Building security in", 1er. ed., Addinson Wesley", pp. 84, 2006

Manuel Dávila Sguerra. *Ingeniero de Sistemas Uniandes, Director Departamento de Informática y Electrónica Uniminuto, Coordinador Académico ACIS, Columnista de Computer World, eltiempo.com: 110 artículos publicados, Autor de e-Genesis- El Generador de sistemas, Mención especial en el Premio Colombiano de Informática 2006, Escogido entre los 25 IT Manager del año 2008 por la revista IT-Manager, Miembro Fundador de ACIS, de Indusoft hoy Fedesoft, de la Red de Decanos y Directores de Ingeniería de sistemas, REDIS, Autor de los libros "GNU/Linux y el software libre" y "Software libre una visión".*