Universidad de Buenos Aires

Facultades de Ciencias Económicas, Ciencias. Exactas y Naturales e Ingeniería

Carrera de Especialización en Seguridad Informática

Trabajo Final

Título:

Seguridad Informática en Proyectos Ágiles

Autor:

Facundo Mauricio

Tutor:

Diego Carralbal

Año de Presentación: 2018

Cohorte: 2017

Declaración Jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

FIRMADA

FACUNDO NAHUEL MAURICIO

Resumen

Este trabajo de investigación descriptivo-explicativo a partir una fase exploratoria de la bibliografía y la experiencia profesional, se divide en tres partes para facilitar su comprensión. Avanzando desde una visión general, académica y teórica hacia el detalle de implementaciones y mejoras concretas aplicables en la gestión de proyectos con el fin de mejorar la seguridad de los sistemas desarrollados.

En la primera parte, el lector encontrará una descripción de los problemas típicos de seguridad que se asocian al desarrollo de software y se presentan como la puerta para comprender la necesidad de cambios en las metodologías de gestión ágil de proyectos.

En la segunda parte, encontrará información respecto de las metodologías ágiles, particularmente Scrum –definida por la Scrum Alliance–. Estos capítulos explicarán cómo funcionan estas metodologías, principalmente en lo relacionado a cuestiones asociadas con la seguridad informática.

Finalmente, la tercera parte de este documento hace referencia a la experiencia práctica del autor, quien busca proponer, partiendo de su recorrido profesional, cambios en el *framework* de Scrum para incorporar la seguridad informática al proceso de desarrollo en sí, en lugar de entenderla como una funcionalidad o requerimiento adicional a tomar en cuenta para nuevos sistemas.

El objetivo del trabajo es, entonces, brindar herramientas a los futuros managers para incorporar la seguridad informática a los procesos que desarrollen de manera más integrada y, por lo tanto, más eficiente.

Palabras Clave

Seguridad Informática | Gestión de Proyectos | Metodologías Ágiles | Scrum | Riesgos

Índice

Prólogo

El presente documento no hubiera sido posible sin el aporte y el apoyo de los varios amigos, colegas, familiares y principalmente profesores de esta maravillosa casa de estudios que es la Universidad de Buenos Aires. Con su conocimiento y experiencia, permitieron que podamos aprender y crecer como profesionales no solo en el ámbito académico, sino también en el plano ético y moral.

El cuerpo docente de esta Especialización requiere sin dudas un amplio reconocimiento por su calidad y prestigio, destacando particularmente la labor del Dr. Raúl H. Saroka y el Dr. Hugo D. Scolnik como aquellos que han dejado una marca en mi formación académica y profesional, a quienes estoy profundamente agradecido. Adicionalmente una mención especial al Dr. Pedro Hetch quien me apoyó para continuar con este Trabajo Final de Especialización cuando su progreso parecía detenido en el tiempo.

PARTE 1 La seguridad informática

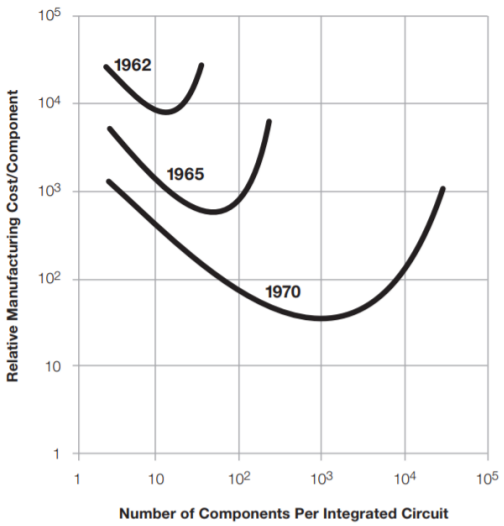
Desarrollo de Software

En el presente capítulo se explorará brevemente la historia de las metodologías de desarrollo de software más comunes desde sus orígenes hasta la actualidad y se considerará cómo evolucionó la complejidad de los proyectos de software hasta convertirse en el enfoque multidisciplinario que conocemos hoy.

Comienzos metodológicos

Diversos artículos ubican los orígenes de la ingeniería de software a principios los años 60; sin embargo, lo que denominamos ingeniería de sistemas fue nombrado por primera vez en el año 1968 en una conferencia organizada por NATO [1]. Podemos afirmar entonces que, si bien se desarrollaba software con anterioridad a esta conferencia, recién aquí se comienza a visualizar la complejidad intrínseca en el diseño y mantenimiento de los sistemas, y se requieren por primera vez discusiones sobre los procesos, las herramientas y principalmente las metodologías que se usarían para llevar adelante los desarrollos en el futuro.

Según N. Wirth, es en este punto donde la programación comienza a enfrentase a la rigidez y el formalismo de las distintas metodologías clásicas que existían en ese entonces. Con el aumento del poder computacional expresado en la famosa ley de Moore, la complejidad de las tareas solicitadas a los desarrolladores y las demandas de los procesamientos que los sistemas tenían que manejar hicieron difícil que exista un solo desarrollador capaz de diseñar, controlar o incluso comprender la totalidad de un sistema por sí mismo. Este proceso se da en congruencia con la disminución de los costos del hardware, lo que acelera esta tendencia y marca el cambio de época, donde la programación pasa de ser una tarea individual para convertirse en el resultado de un proceso grupal y bien organizado. Nótese que este es el momento en que las limitaciones de las creaciones de software dejan de ser principalmente físicas o de hardware para deberse al intelecto humano y de su capacidad para organizarse eficiente y eficazmente. [2]



El diagrama original expresado por el paper de Moore de donde surge la interpretación de su famosa ley, expresando que la densidad de procesadores en los circuitos integrados se duplica cada dos años. Infiriendo que su poder también se duplica. [2]

Los primeros problemas de seguridad

El mercado de la computación a nivel global comienza a cambiar con el crecimiento de Unix, un sistema desarrollado por Ken Thompson en Bell Labs, mucho más liviano y apto para su uso en computadoras más pequeñas que por primera vez alejándose del mundo corporativo y enfocado a empresas y hogares. Estos cambios afectan la realidad del sector tecnológico, modificando el hasta entonces dominante enfoque de los sistemas diseñados para supercomputadoras corporativas, reemplazando estos por nuevos equipos más pequeños capaces de menor poder computacional, pero más accesibles para las tareas comunes que requieren los usuarios personales y las pequeñas y medianas empresas. En este marco surge C –desarrollado por Dennis Richie, también en Bell Labs–, el lenguaje por excelencia para ser usado en sistemas Unix y cuyo crecimiento exponencial se extiende hasta nuestros días, siendo aún la base de muchos de los sistemas que utilizamos, principalmente preferido para aquellos sistemas que requieren un uso eficiente de la memoria y una estabilidad probada en años de implementaciones en infraestructura crítica, a la vez que en diferentes dispositivos electrónicos que buscan velocidad y bajo consumo. Este nuevo

lenguaje (C) representa un gran alivio para desarrolladores gracias a su abstracción y eficiente asignación de recursos. [1]

Con el aumento de la complejidad en los sistemas y con el crecimiento de los lenguajes que comienzan a utilizar abstracciones en lugar de lenguaje de bajo nivel como Assembler, dichos cambios permiten que los proyectos comiencen a crecer en complejidad y que se incorporen nuevos programadores a los equipos lo que fuerza a pensar la modularizacion de los sistemas y la necesidad de incorporar nuevos procesos para maximizar las posibilidades de éxito del proyecto. [1]

Junto con la abstracción, C trae menos estructura y las primeras posibilidades de poner códigos mucho menos sólidos al no validar correctamente los tipos de datos, o no forzar índices en vectores, por citar algunos ejemplos. Esta idea de usar lenguajes de alto nivel para mejorar el código en general terminó afectando la calidad y seguridad de los sistemas creados, exponiéndolos a problemas de longitudes de datos, *overflows*, y punteros que direccionan a posiciones de memoria que, al menos, ofrecen vectores de ataques para futuros hackers e investigadores. Los académicos se enamoraron de C, algunos dicen que debido a su sintaxis más amigable y, según otros, a que C permite romper varias reglas de los lenguajes más estructurados como *Assembler*, dándoles mayor libertad a los desarrolladores para programar sin tantas restricciones o validaciones.

Con el movimiento hacia lenguajes de cada vez más alto nivel –como C++, donde se buscaba atender a los problemas mencionados anteriormente–, se produce un efecto de bola de nieve: son más difíciles de mantener y comprender, con compiladores cada vez más complejos y con mayores capas de abstracción que dan lugar a nuevos errores y vulnerabilidades. Cabe destacar que, según algunos autores, los lenguajes de alto nivel están más cerca de ser parte fundamental del problema que de aportar a las soluciones. [3]

Hacia nuevos horizontes

La siguiente ruptura viene de la mano de la modularización de la programación, donde brillaban lenguajes como Pascal, que permitían a varios equipos trabajar en

simultáneo en distintos módulos para luego integrarlos y crear sistemas complejos en menor tiempo.

Esto nos acerca a los modelos actuales, en los que múltiples equipos trabajan codo a codo para generar complejos sistemas que luego son integrados, y nos alinea con el objeto de análisis de este trabajo, que se centra en la investigación sobre las metodologías ágiles y su necesaria evolución para incorporar la seguridad informática como otro eje tan fundamental como la calidad misma en sus procesos y métricas.

Actualidad de la seguridad informática

A la vista de lo descrito anteriormente, es claro que la seguridad debe incorporarse como prioridad en cualquier desarrollo de software que se precie de valorar a sus usuarios y clientes. Con esto en mente es que el autor de este documento propone fortalecer la incorporación de la seguridad en las metodologías ágiles, buscando que sea integrada como parte de la metodología.

Uno de los desafíos que enfrentamos en la actualidad es la falta de desarrolladores formados en materia de seguridad informática que puedan incorporarse a los equipos de desarrollo que estén utilizando metodologías ágiles. La experiencia de diversos profesionales y la de este autor en particular demuestra que estos dos mundos en principio no parecen convivir de forma natural. Lamentablemente la formación actual en seguridad informática no prepara a los profesionales para formar parte activa de un equipo de desarrollo, haciendo que sea muy costoso para un equipo ágil incorporar un desarrollador no ágil al proyecto.

Con esta visión, este documento explora las diferentes formas en que las metodologías ágiles, en particular Scrum, pueden alinearse con la seguridad informática.

A continuación, una lista de las principales consideraciones:

Backlog de Seguridad

Es necesario que los *Product Owners* (definición en la segunda parte de este documento) acepten la necesidad de incorporar la seguridad como parte fundamental

de los requerimientos de un sistema. Debe comprenderse que en lugar de pensar en la seguridad como una funcionalidad más, debe considerársela como se considera la calidad: algo que debe estar siempre presente y que afecta a todo el sistema en simultáneo.

Revisemos por un momento cual es el proceso tradicional por el cual se manifiesta la creación de un *backlog* y como el mismo acompaña el ciclo de vida de desarrollo del producto, definiendo que será necesario construir para considerar al proyecto completo.

El *backlog,* lista de forma exhaustiva todo aquello que será incorporado en el producto, esencialmente detallando el *roadmap* del mismo y listando todas aquellas funcionalidades que tendrá. Nótese el inconveniente de este enfoque, donde a menos que los interesados en el sistema expliquen de forma particular los requisitos de seguridad, los mismos no serán incluidos en el *backlog*, y por tanto no serán estimados, ni serán priorizados, ni serán discutidos a menos que algún especialista en la materia se detenga a informar los potenciales riesgos.

Este inconveniente es de amplio conocimiento por aquellos profesionales con varios años de experiencia en el rubro, sin embargo, a fin de acercar esta información al lector menos familiarizado con la definición y gestión de proyectos, podemos generar una similitud con los requerimientos asociados a la *performance* que generalmente no son definidos en todos los sistemas. Cuando un sistema es probado por el cliente, el mismo será analizado desde diferentes ángulos, funcionalidad, estética, experiencia de usuario, velocidad de respuesta, etc. Uno pensaría que la mayoría de los parámetros fueron definidos de antemano, pero no siempre es el caso, particularmente *performance* es un ejemplo de aquello que el usuario espera, pero que no siempre define en los requerimientos. Afortunadamente es muy fácil identificar los problemas de performance, puesto que se manifiestan con facilidad al utilizar el sistema. Los problemas de seguridad generalmente se encuentran menos definidos que los requerimientos de *performance* y son ordenes de magnitud más difíciles de encontrar, puesto que no serán percibidos por ningún usuario que carezca de conocimientos

específicos en seguridad informática, particularmente en la disciplina que se conoce como *penetration testing.*

Falta de desarrolladores

Con la escasez actual de recursos en seguridad informática, es difícil pensar que se trata de un problema que pueda resolverse en el corto plazo. Existe en el mercado una falta de personal entrenado en materia de seguridad y aplicaciones seguras. A su vez, las universidades y centros de formación no están enfocándose en este tipo de habilidades y tampoco están formando a la próxima generación de programadores con una mejor capacitación en seguridad.

Esta es una situación que ha sido advertida hace varios años, por lo que se bifurcó la formación en seguridad informática como una corriente laboral en particular, distinta de la del camino del desarrollador tradicional. Esto fomentó la aparición de profesionales orientados específicamente a la seguridad, pero con poca experiencia en desarrollos tradicionales (mucho menos en desarrollos ágiles) siendo luego difícil incorporarlos a los equipos y procesos de desarrollo que se dan en las organizaciones.

Vale la pena considerar este punto para comprender la necesidad de incorporar mejoras en la metodología. Siendo los especialistas en seguridad informática un recurso tan escaso, es necesario considerar alternativas para optimizar su utilización en los proyectos en lugar de asumir su libre disponibilidad. Esta limitante de orden primario, obliga a los responsables de la gestión a maximizar los beneficios entregados por cada profesional de SI a la vez que se deben a la búsqueda de la minimización de riesgos a los que está expuesto el producto que desarrollan. En esta disyuntiva impuesta por un mercado que tarda años en formar expertos en SI y cuya demanda actual excede por lejos la capacidad de las organizaciones de cubrir todas las posiciones, es que el profesional en la gestión de proyectos debe ser creativo y seleccionar alternativas que generen el mayor valor para los clientes y sus *stakeholders.*

Equipos reactivos

Lo explicado convirtió al área de seguridad en un problema para los desarrollos actuales, siendo habitualmente presentada como un área que solo opone resistencia y que aparece sobre el final de los procesos de desarrollo obstaculizando su puesta en producción.

Esto, naturalmente, tiene que ver con un gran número de factores, pero principalmente con la falta de habilidades de los miembros de los departamentos de seguridad para ser parte del proceso de desarrollo en sí, en lugar de aparecer para revisar el sistema una vez terminado y presentar un informe con un sinnúmero de vulnerabilidades y defectos.

Esta interacción entre los programadores y los expertos en seguridad, desde luego poco feliz, no hace más que complicar las posibilidades de crear aplicaciones seguras. Por lo tanto, es importante poner énfasis en que, a menos que la seguridad sea "parte" del proceso de desarrollo, no vamos a tener nunca aplicaciones realmente seguras.

Velocidad al mercado

Otro problema que enfrenta la seguridad en los procesos de desarrollo es el tiempo que insume crear aplicaciones seguras o, aún más, asegurar aplicaciones una vez desarrolladas.

Consideremos el tiempo adicional de desarrollo que requiere incluir medidas de seguridad o bien el tiempo adicional de diseño que requiere diseñar una aplicación segura. El costo no está solo asociado a los recursos adicionales para programar la seguridad en el sistema, sino a los altos costos asociados con su validación y testeo.

Los mencionados costos no solo incrementan las dificultades financieras, sino que además aumentan la presión sobre los tiempos de entrega y salida a producción, pues toda funcionalidad o requerimiento adicional tiene un impacto directo en el lapso requerido para implementarlo.

En las condiciones actuales del mercado, estas demoras son más “costosas” en términos de oportunidad que por los gastos financieros reales que originan. Generalmente es el tiempo y no el costo el factor crítico del éxito de muchos sistemas comerciales, especialmente aquellos orientados a un público masivo.

Preferencias de los usuarios

Entender las preferencias de los usuarios es un desafío para cualquier profesional, por lo tanto, generalizar en este aspecto es bastante complejo. Sin embargo, a raíz de la experiencia del autor y de varios colegas con los que se ha contactado, podemos afirmar que los usuarios suelen pensar poco en la seguridad y se encuentran lógicamente más cerca de los requerimientos desde el punto de vista del negocio.

Desde luego, pues existen muchos y muy variados grupos de usuarios y no todos tienen los mismos requerimientos, pero creemos que es posible sostener que la seguridad no es todavía una prioridad en la mente de la mayoría de los usuarios al momento de la compra de un software.

Este inconveniente se ve trasladado directamente a la ecuación de costos de la empresa que gestiona el desarrollo, pues si los usuarios no valoran la seguridad, es difícil justificar el gasto de desarrollar aplicaciones seguras.

Incorporando la seguridad en el desarrollo ágil

Como se ha expresado con anterioridad, el autor de este documento considera que la solución al problema de las aplicaciones seguras no pasa por la concepción de la seguridad en sí misma o su intrínseca complejidad de implementación, sino principalmente por encontrar procesos en las metodologías de desarrollo que permitan diseñar y programar aplicaciones seguras con la misma rigurosidad procedural con que se miden e implementan controles de calidad.

Como ya se ha dicho, la tercera parte de este trabajo se propone analizar diversas modificaciones al proceso de desarrollo ágil conocido como Scrum, buscando atacar el problema a través de la incorporación de la seguridad, no como funcionalidad adicional,

sino como deuda técnica necesaria para el desarrollo del sistema en cuestión. Antes de avanzar sobre este propósito, en la segunda parte propondremos un análisis de los procesos más relevantes tal como se utilizan actualmente.

Desarrollo seguro

El desarrollo de sistemas seguros no es fin sino un ideal que es constantemente reinventado a medida que diversos actores globales (bien o mal intencionados), encuentran nuevas vulnerabilidad y nuevas formas de proteger la información de los usuarios y organizaciones. No hay metodología ni proceso que garantice la seguridad, como no hay metodología ni proceso que garantice la calidad, sin embargo, hace años que como industria decidimos que era momento de implementar diversas metodologías que permitan reducir la posibilidad de entregar software con defectos. El autor presenta este documento como un intento de consolidar recomendaciones para mejorar *Scrum* a los fines de generar software más seguro. Desde luego, la gestión es solo una parte de este proceso, pero debe ser la que al menos proponga las herramientas para visibilizar esta situación y actuar en consecuencia.

PARTE 2 Metodologías y procesos

Seguridad en el desarrollo de Software

Podemos observar que la seguridad informática no comienza con la invención de las computadoras, sino con la información misma. De hecho, tan pronto como se empezó a transmitir o almacenar información, se comenzaron a identificar formas de protegerla, probablemente remontándose a los orígenes mismos de la escritura. Pueden citarse como ejemplos la encriptación usada por el César para comunicar información confidencial o el desarrollo de un código morse alternativo para encriptar comunicaciones. [4]

También la ley ha acompañado dentro de sus facultades –aunque con claros retrasos– estas tendencias a garantizar la seguridad de la información. Este acompañamiento estuvo siempre subordinado a la agenda política y económica de turno. Así, por ejemplo, la prohibición de las escuchas de conversaciones privadas se produjo recién un año después de la invención del teléfono.

Con la llegada de la Primera y especialmente la Segunda Guerra Mundial, el manejo de la seguridad en las comunicaciones se vuelve un tema fundamental en cualquier discusión sobre el esfuerzo bélico. Esto se alinea perfectamente con la explosión de la tecnología, particularmente el desarrollo de las computadoras, que se convierten en el gestor principal de la información sensible de los gobiernos. De este modo, queda planteada en un primer plano la discusión sobre la seguridad del software, herramienta que administra uno de los activos más valiosos de los Estados, la información.

Cabe destacar que los problemas en la seguridad informática no son solo una cuestión técnica sino el lógico resultado de una consideración económica, que busca maximizar la inversión en aquellas funcionalidades que proponen más valor o son más solicitadas por los usuarios. Las organizaciones no tienen actualmente incentivos económicos para desarrollar aplicaciones particularmente seguras, ya que los usuarios todavía no ponderan a la seguridad como una característica buscada en los sistemas o aplicaciones que consumen.

Hasta tanto los creadores de software no sean capaces de incorporar seguridad sin afectar la experiencia de usuario, generar aplicaciones seguras seguirá siendo muy complejo [5].

Finalmente, hasta que la necesidad de seguridad en los sistemas no se expanda entre los diferentes oferentes del mercado y se genere un efecto colectivo tal que exista un valor percibido adicional por ser considerado una aplicación segura, es difícil pensar que vaya a ser una prioridad para las empresas desarrolladoras de software incorporar la seguridad informática en sus productos. [6]

Gestión y cumplimiento de objetivos

Cuando los profesionales en las diversas áreas asociadas a la gestión nos detenemos a evaluar la necesidad de procesos y metodologías, buscamos un soporte que nos permita hacer y gestionar aquello que parece obvio, pero que en el dinamismo del proyecto tiende a pasar como secundario. Es en este marco conceptual, que los profesionales debemos incorporar en las metodologías que utilizamos los fundamentos teóricos y prácticos que nos permitan monitorear el cumplimiento de distintos objetivos, más allá de aquellos planteados por los clientes.

Un ejemplo de esta situación es la seguridad informática. Pocos clientes proveerán a los equipos de desarrollo con claros lineamientos respecto a la seguridad esperada de un producto final. La mayoría de los clientes asumen que sus aplicaciones serán seguras, pero la definición de seguridad es amplia y compleja, puesto que distintos niveles de seguridad son alcanzados con distintos niveles de inversión, tanto sea en tiempo como en valor monetario.

Los sistemas relacionados a procesos críticos de las organizaciones tienden a tener metas claras respecto a la seguridad esperada de las aplicaciones, pero ¿qué decir entonces de otra miles, sino millones, de aplicaciones? Varias aplicaciones para celulares (conocidas comúnmente como *apps)*, para PyMEs y para usos secundarios en diversas organizaciones no cuentan con claros estándares de seguridad, ni parecen requerirlos. He aquí presentada la falacia, que con su inocencia pone en riesgo a millones de usuarios

y empresas todos los días y cuyos datos e información son sensible a ser obtenidos y manipulados por un actor mal intencionado.

Actualmente, es difícil hacer responsables a las organizaciones por su desconocimiento, pero como expertos y profesionales en la materia, es nuestra labor y deber el de crear mecanismos que ayuden a garantizar la seguridad de los sistemas de nuestros clientes y de los datos de sus usuarios, más allá de su entendimiento técnico en las complejidades inherentes en la seguridad de la información.

Estas observaciones de años de trabajo del autor en la gestión de proyectos grandes y pequeños, para empresas y para organismos del Estado, en Argentina y en el exterior, ponen de manifiesto la necesidad de incorporar la seguridad informática como una serie de mejoras concretas en la implementación de *Scrum* buscando, en el peor de los casos visibilizar la necesidad de seguridad y gestionar los riesgos inherentes al desarrollo, y en el mejor de los casos la implementación de mejoras durante el proceso de creación del software que prevengan problemas asociados a la seguridad informática.

Metodologías de Desarrollo

Con el devenir de los años, se volvió claro que las metodologías clásicas no eran suficientes para gestionar los cada vez más complejos proyectos de software, especialmente dada la volatilidad de la tecnología y los constantes cambios que se dan también en los mercados, que luego afectan los requerimientos de los desarrollos.

En este punto, es crucial aclarar que el presente documento propondrá un análisis desde la perspectiva de la gestión ágil de proyectos principalmente representada por Scrum, el *framework* más popular en la actualidad.

El siguiente análisis corresponde a las principales características de Scrum ponderando principalmente aquellos detalles que, para el autor, tienen más relevancia en el campo de la seguridad informática, como ser riesgos, cambios y valor generado.

Scrum

Esta metodología surge por la necesidad de gestionar proyectos de forma más dinámica y cambiando el enfoque en varias áreas a los fines de mejorar las posibilidades de éxito del proyecto.

Scrum es un *framework* donde el equipo puede atacar problemas complejos de forma adaptativa, al mismo tiempo que entrega productos que proveen el máximo valor posible en forma creativa y eficiente.

Scrum es:

- Liviano
- Simple de entender
- Difícil de dominar [7]

Cambios

En Scrum, la idea del cambio es aceptada como parte natural del proceso de desarrollo de software y no vista como algo a evitar, de hecho, se la considera beneficiosa para el proceso, pues se asume como imposible comprender y diseñar a la perfección un sistema antes de comenzar a construirlo. Además, en el caso de que fuera posible diseñarlo a la perfección previamente, Scrum asume esto como un costo innecesariamente alto, pues es mejor ir experimentando y analizando durante desarrollo en lugar de esperar a tener todo documentado y comprendido para comenzar a programar.

Nótese aquí cómo este enfoque no aplica a un proyecto como la construcción de un puente, donde todo el análisis debe ser hecho al principio, pues es un factor crítico de éxito conocer la longitud, peso, estructura y otros detalles que mantendrán al puente en su lugar. Por otro lado, invito al lector a considerar cómo mucho de esto no se da con respecto al software en la actualidad, donde con frecuencia los sistemas cambian con respecto a la idea original para alinearse con los cambios de mercado, con las acciones

de los competidores, con las nuevas regulaciones, con los cambios en las tecnologías, etc.

Riesgos

Respecto de los riesgos, la metodología Scrum los concibe como un factor mitigable a través de la reducción de los tiempos de desarrollo. Esto no implica la reducción de los tiempos totales, sino la división del desarrollo en múltiples iteraciones de pocas semanas cada una, donde el riesgo se reduce mediante ciclos cortos de desarrollo que permiten a los diferentes interesados proveer *feedback* con rapidez, lo que permitirá evitar grandes desviaciones, pues los cambios y errores serán identificados tempranamente, permitiendo al equipo de desarrollo su rápida corrección, y reduciendo así el riesgo de terminar el proyecto con desviaciones de gran magnitud, al menos en términos de requerimientos.

Valor generado

En el enfoque de Scrum, la idea del valor generado es clave para comprender la gestión de riesgos y especialmente para entender el cambio fundamental de paradigma con respecto a otras metodologías. Vale considerar que un puente construido al 90% de su longitud tiene valor cero, pues no puede ser usado en absoluto hasta tanto esté completo.

En oposición, un proyecto de software debería poder ser cancelado en cualquier parte del proceso y la organización se quedaría con un sistema que entrega una funcionalidad concreta y por lo tanto un valor a los usuarios, aunque no esté completo. Esto se logra al enfrentar cada iteración como si fuera una entrega final a producción, en un sistema que funciona materializando avances cada, pocas semanas. Por lo general, esto permite acelerar la salida a producción, ya que las funcionalidades finalmente requeridas para sacar el producto al público suelen ser menos que las solicitadas inicialmente. Es lo contrario de lo que sucedía en el ejemplo del puente, donde no era posible habilitar ninguna parte de él si no estaba completa la totalidad de la construcción.

La situación descrita nos permite comprender entonces que los proyectos de software deben ser preparados para ser entregados antes de la fecha de finalización prevista, y esto requiere una metodología capaz de manejar un enfoque flexible y siempre lista para salir a producción.

Información adicional

Como el lector puede percibir, Scrum es un *framework* que resulta más útil en la dinámica actual del mercado tecnológico y no es casualidad que su popularidad haya aumentado significativamente en los últimos años, volviéndose la opción más elegida entre los responsables de gestionar proyectos, que deben liderar proyectos desafiantes y entre las organizaciones cuyos escenarios de negocios cambian con gran regularidad.

Cabe destacar que, si bien Scrum es una opción muy acertada para proyectos de software de tamaño medio, no está garantizado que sea la mejor opción para desarrollos gigantes, donde se integra el trabajo de quizá cientos de desarrolladores. En estos casos existen otras alternativas que ofrecen mejores herramientas para lidiar con la complejidad inherente a la gestión de gran cantidad de recursos, tanto humanos como tecnológicos.

Otro punto para tomar en cuenta es la madurez de la organización y su capacidad de seguir procesos y metodologías. No cualquier organización es buena candidata para implementar metodologías ágiles, así como no cualquier organización está preparada para llevar adelante complejos procesos de desarrollo.

Generalmente, es preferible el uso de metodologías ágiles, mas no debe nunca considerarse esto como una verdad absoluta. Cada organización debe elegir la metodología que mejores resultados brinde y que se alinee con su cultura e identidad.

A continuación, y dado el enfoque del documento, se presenta un análisis más detallado de la metodología Scrum, a fin de facilitar al lector la comprensión de las recomendaciones y consejos incluidos en la parte final.

Scrum – Conceptos básicos

El presente documento no pretende interiorizarse en todos los detalles respecto a las metodologías ágiles ni a Scrum, sin embargo, este capítulo propone repasar algunos conceptos básicos a los fines de sentar las bases conceptuales que permitirán comprender las recomendaciones que se mencionarán en capítulos subsiguientes.

Visión general

Luego del informe Chaos del Standish Group queda claro que otras metodologías estaban fallando en entregar los resultados prometidos en los proyectos de IT. Este informe propone que el desarrollo de software es más parecido a la investigación que a la construcción o manufactura. Esto significa que las técnicas deben acomodarse a las distintas necesidades y enfocarse en una mayor flexibilidad, indiscutible factor de éxito dadas las condiciones del mercado actual. [8]

Cómo funciona Scrum

Esencialmente, Scrum es un *framework* que permite el desarrollo de aplicaciones en forma iterativa valorando individuos e interacciones en lugar de procesos y herramientas, a través de sistemas que funcionan mejor que una documentación detallada, favoreciendo la colaboración con los clientes en oposición a las negociaciones de contratos y, finalmente, con flexibilidad para adaptarse al cambio por sobre la planificación y el control.

En la práctica, Scrum se manifiesta de la siguiente manera:

1. El *Product Owner* crea una lista priorizada de funcionalidades que el sistema debe tener para cumplir con el objetivo que se espera del software. Dicha lista es conocida como *Backlog* y representa todo lo que es deseable que un sistema posea en términos de funcionalidad.
2. Durante el *Sprint Planning*, el equipo de desarrollo selecciona una porción del *Backlog,* preferiblemente de la parte de arriba (pues representa las funcionalidades más importantes) y lo incorpora en el *Sprint Backlog* para luego discutir cómo será desarrollada la funcionalidad.

3. El equipo cuenta con una cantidad de tiempo predeterminada (usualmente de dos a cuatro semanas) para completar el trabajo definido en el *Sprint Backlog.* En ese tiempo, el equipo se reúne cada día en el *Daily Scrum* para revisar el progreso de cada tarea y discutir los problemas que surjan durante el desarrollo.
4. El *Scrum Master* es el responsable de mantener al equipo enfocado en los objetivos propuestos y a la vez de asegurarse de que los procesos y las ceremonias se cumplan según lo acordado. Es el principal responsable de coordinar al equipo para que tome las mejores decisiones con el fin de alcanzar el objetivo del proyecto.
5. Al final del *Sprint,* el trabajo realizado debería estar listo para salir a producción, con todos los estándares de calidad y de acuerdo con las especificaciones del cliente.
6. Un *Sprint* termina con un *Sprint Review,* donde el producto se muestra a los clientes y *stakeholders* para recibir un *feedback* y comprobar si el trabajo realizado por el equipo está alineado con la visión y objetivos del cliente. Adicionalmente, el equipo tendrá una reunión interna llamada *Retrospective* para discutir y revisar los procesos y sucesos ocurridos durante el *Sprint* a modo de buscar mejoras en la metodología y darse un espacio de reflexión que fomente el perfeccionamiento del grupo.
7. En este punto, el proceso se repite desde el inicio, seleccionado otro grupo de funcionalidades del *backlog* para incorporar en el siguiente *Sprint.* [7] [9]
8. Visualmente podemos representarlo como:

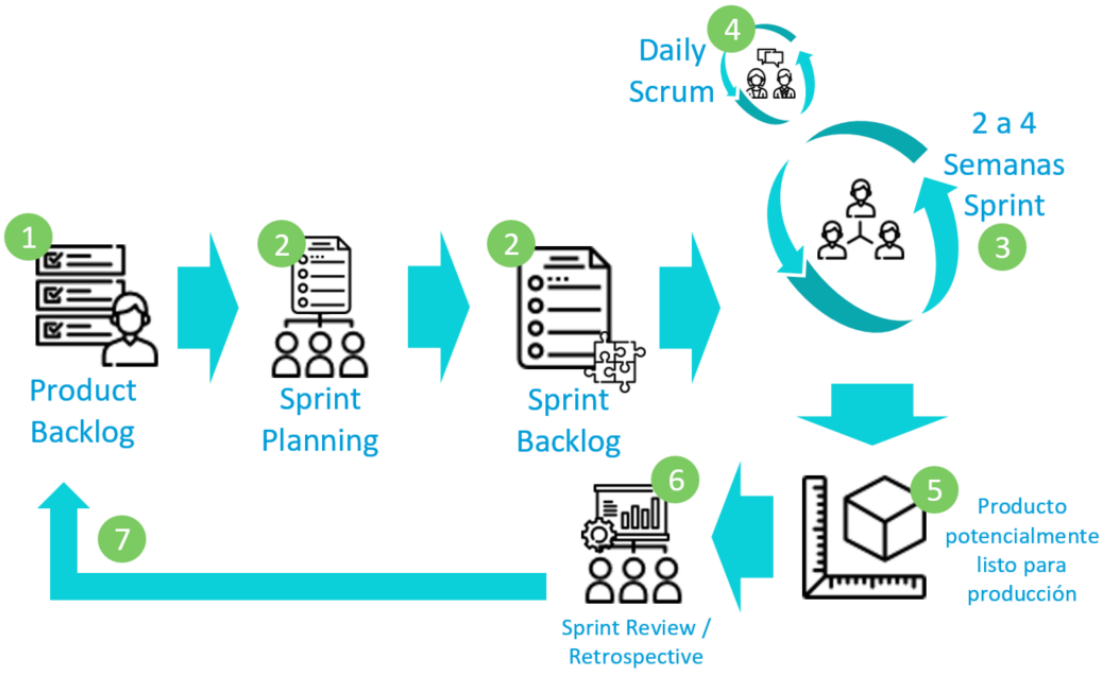


Diagrama expresando el proceso de Scrum (elaboración propia basada en diagramas de Kenneth S. Rubin [10])

El equipo

En Scrum, es fundamental el rol del equipo, que es por definición la pieza clave del éxito de un proyecto, a punto tal que se espera que los equipos de Scrum no tengan un jefe o líder, sino más bien que se autoorganicen a los fines de tomar decisiones colectivas y por consenso en lugar de acatar órdenes “de arriba” como en otras metodologías.

El equipo tiene algunas características particulares, como el hecho de cada miembro debería poder hacer cualquier tarea que se requiera para el proyecto, alejándose de la idea del especialista. También, los compromisos del proyecto son para el equipo y no para ningún miembro en particular, todo el equipo debe salir adelante de las dificultades como una entidad y no dividido en miembros.

Los roles

La siguiente tabla describe los roles comúnmente aceptados en los proyectos de Scrum:

Posición	Descripción
Product Owner	Responsable de maximizar el valor del producto a ser desarrollado.
Scrum Master	Responsable de promover y apoyar el Scrum como *framework* de trabajo, empoderando al equipo y desarrollando la comprensión de las reglas, los métodos, los valores, etc.
Team Member	Responsables de desarrollar el sistema.

La agenda

Durante los desarrollos que utilizan Scrum, se lleva a cabo una serie de ceremonias que funcionan para sostener al equipo durante el proyecto y darle un marco de trabajo. Estas ceremonias se listan a continuación:

Ceremonia	Descripción
Sprint	Es un espacio temporal fijo y definido donde el equipo debe completar partes del proyecto idealmente listas para salir a producción.
Sprint Planning	Es una reunión donde el equipo planifica qué se va a realizar durante el *Sprint.*
Daily Scrum	Es una *mini* reunión diaria, también de tiempo definido, donde el equipo discute lo sucedido el día anterior y lo que se va a hacer durante la jornada.
Sprint Review	Es una reunión al final de cada *Sprint* para revisar lo producido. Generalmente se incluye una *demo* para que los clientes puedan ver el progreso y proveer *feedback*.
Retrospective	Es una reunión posterior a *Sprint Review* donde el equipo puede discutir lo que salió bien y lo que debería mejorarse para la próxima iteración.

Otras consideraciones

A continuación, se listan aquellos ítems que forman parte del *framework* de Scrum, comúnmente conocidos como *artifacts*.

Ítem	Descripción
Product Backlog	Es la lista ordenada de todos los requerimientos conocidos del proyecto.
Sprint Backlog	Es la lista ordenada de todo lo que el equipo se comprometió a producir durante el *Sprint*.
Definition of Done	Es una serie de criterios definidos por el equipo para considerar una tarea como terminada.
User Story	Herramienta para la captura de requerimientos.

Captura de requerimientos con User Stories

Scrum no exige la utilización de ningún método en particular para la captura de requerimientos de un sistema a desarrollar. Sin embargo, entre los practicantes de esta metodología, se da con mucha frecuencia el uso de las llamadas *User Stories*. Esta herramienta permite capturar funcionalidades desde la perspectiva de la persona que desea la nueva funcionalidad, generalmente un usuario del sistema.

Normalmente las *User Stories* tienen el siguiente formato:

- As a < type of user >, I want < some goal > so that < some reason >

Analizando esta estructura en español obtenemos que:

- Como <**tipo de usuario**>, quisiera <**objetivo**> para que <**justificación**>

Siendo:

- **Tipo de usuario:** un usuario del sistema que normalmente representa el beneficiario de la funcionalidad requerida.
- **Objetivo:** la descripción de la funcionalidad en sí misma, desde el punto de vista del usuario que la solicita y en términos genéricos. La idea es describir la funcionalidad con el mínimo detalle, buscando enfatizar la necesidad, pero dejando la solución (el cómo) al equipo de desarrollo.
- **Justificación:** es la razón por la cual el usuario solicita dicha funcionalidad. Existe principalmente como medio para darle al equipo de desarrollo la posibilidad de buscar alternativas creativas a la necesidad expresada por el usuario. El usuario expresa el qué y el porqué, pero deja el cómo para ser definido por el equipo durante el desarrollo.

Por ejemplo, para apoyar la comprensión del lector podemos considerar los siguientes casos de *User Stories*:

1. Como **usuario administrador** quisiera **poder crear una copia de mi disco** en caso de que **falle el disco principal.**
2. Como **cliente del banco** quisiera poder **transferir dinero a otra cuenta con mi teléfono móvil** para **evitar tener que ir personalmente a la sucursal.**
3. Como **recepcionista** quisiera **redirigir mails entrantes a grupos de usuarios** para **reducir la carga operativa** de hacerlo manualmente.
4. Como **arquitecto** quisiera poder **monitorear el progreso de la obra desde internet** para **no tener que visitar la construcción** tan seguido.

Como podemos apreciar, el objeto final de este enfoque es darle libertad al equipo de desarrollo para generar la solución óptima, evitando definir el cómo y enfocándose en cuál es la necesidad real del usuario.

Si bien este método ha probado ser muy efectivo en la captura de requerimientos, también muestra algunas debilidades respecto de las cuestiones técnicas, pues pone el foco en el negocio y deja el análisis técnico en un segundo plano. En este enfoque, el cliente pierde visibilidad con respecto a cómo se hace o funciona el sistema y solo se enfoca en sus necesidades. Si bien no hay duda de que esto representa

una ventaja para los usuarios, es de destacar que hay riesgos en alejar al negocio del entendimiento y la comprensión técnica de lo que se está construyendo.

PARTE 3 Incorporando seguridad en Scrum

Introducción

Para bajar a la realidad toda la teoría presentada en las primeras dos partes de este documento, se pretende a continuación transmitir parte de la experiencia adquirida por el autor en un proyecto realizado durante su estadía en el exterior.

La información aquí dispuesta ha sido limitada para no aburrir al lector con detalles demasiado técnicos. Sin embargo, se pretende dar a conocer a aquellos que no han tenido la experiencia de conducir un proyecto de software algunas de las consideraciones que debe tener en cuenta un Líder de Proyecto a la hora de coordinar un desarrollo y algunos de los problemas que enfrentamos quienes tenemos a cargo cotidianamente desarrollos que ponen a la seguridad como eje central.

Caso Práctico

El caso consta del desarrollo de un sistema para la gestión de impuestos relacionados con el manejo de residuos (el cliente solicitó no ser mencionado). Quien escribe este documento, participa en dicho proyecto como *Scrum Master* y como *Senior Business Analyst,* siendo su rol clave en la incorporación de procesos relacionados con la seguridad durante el proceso de desarrollo.

Los siguientes párrafos describen, según la experiencia del autor y simplificadamente, las diferentes etapas, obstáculos e ideas aplicadas al *framework* Scrum para reducir los vectores de ataque a los que el producto final está expuesto.

Requerimientos

El cliente solicita un sistema que permita completar uno o varios formularios a las empresas, de modo tal que puedan hacer la declaración de los residuos recolectados, transportados y procesados. Dichas presentaciones, en carácter de declaración jurada, sirven como medio para el cálculo del impuesto a pagar por parte de estas empresas.

Las diferentes declaraciones, además, sirven para gestionar retenciones, excepciones impositivas, gestión de desastres naturales y también como medio para controlar lo reportado por otras empresas, pues la idea es cruzar información en varios

reportes a fin de identificar conductas de evasión, delictivas o dañinas para el medio ambiente.

Desafíos de Seguridad

Dada la criticidad de la información almacenada, y puesto que la integridad de los formularios es asimilable a la requerida en una declaración jurada, era condición necesaria asegurar que la información completada por las empresas permanezca inalterable a través del tiempo. Al mismo tiempo, para incrementar la complejidad de este punto, era requisito esencial del sistema la posibilidad de enmendar declaraciones anteriores, por lo cual fue creado un sistema de versionado de datos a fin de acomodar la posibilidad de almacenar la información en sus múltiples versiones.

Por otro lado, el sistema requería dar cuenta de que el usuario que completase el formulario tuviera la autoridad legal para responder en nombre de la empresa a los fines de que el reporte tenga validez jurídica, lo que además implicaba generar un proceso de revisión interna donde al menos dos empleados debían aprobar el reporte para que fuera considerado válido.

Finalmente, puesto que el sistema era responsable por varios cientos de millones de dólares, numerosos procesos internos de aprobación debían ser verificados antes de emitir multas o aprobar exenciones impositivas de gran magnitud de dinero. Para ello, los usuarios y sus roles debían tener rangos de aprobación de acuerdo con sus permisos y la habilidad de escalar aprobaciones en varios casos para ciertos procedimientos.

Obstáculos

1. La empresa proveedora del servicio trabajaba utilizando metodologías ágiles, mientras que el cliente, por ser un organismo de gobierno, estaba acostumbrado a una gestión de proyectos más tradicional.
2. El equipo de desarrollo, si bien estaba acostumbrado a trabajar en proyectos ágiles, no poseía una amplia experiencia en el desarrollo de aplicaciones seguras.
3. Se propone la migración de un proceso manual a un sistema informático.

4. Las empresas que reportan el manejo de residuos no cuentan con personal capacitado en la utilización de aplicaciones complejas, por lo que el sistema debe ser sencillo de usar y altamente seguro, dos componentes que no se dan en la práctica sin altos costos.

Lecciones aprendidas

Durante el desarrollo del mencionado sistema se trabajó principalmente en las distintas funcionalidades que conformaban el corazón del software y se dejó de lado la seguridad, como algo para ser evaluado y mejorado una vez que el sistema estuviera más maduro y avanzado.

Esta es una tendencia natural, pues aplicar medidas de seguridad temprano en el proceso de diseño suele implicar un incremento notable en la complejidad del desarrollo, ya que los programadores deben lidiar con todas las medidas de seguridad mientras están desarrollando nuevas funcionalidades o probándolas en sus ambientes locales, y quieren compartirlas con clientes o compañeros de trabajo a fin de obtener *feedback* rápidamente para mejorarlas.

Es claro entonces que la seguridad incorporada desde el día uno se vuelve una carga para el desarrollador. Sin embargo, la experiencia también nos muestra que incorporar la seguridad sobre el final del desarrollo significa que muchas cuestiones han quedado sin revisión, pues esta incorporación tardía es generalmente de tipo perimetral y no implica que un diseño sea seguro en su totalidad.

Con frecuencia, nos encontramos con que los desarrollos que realizan una serie de ejercicios de *pentesting* sobre el final del proceso generan una cultura de seguridad que refiere a la protección exterior de un sistema, pero no a su interior. Es la teoría del M&M, que menciona Kevin Mitnick en *El arte de la Decepción,* según la cual en estos casos la protección parece sólida, pero una vez que se rompe la cobertura exterior, internamente todo es blando y fácil de romper. [11]

Revisando la experiencia

Tal como fue descrito en las lecciones aprendidas, la seguridad de nuestra aplicación se basaba en una serie de resultados de *pentesting* que permitía que fuera sólida desde el exterior, pero que internamente tuviera quizá algunas oportunidades de mejora.

En resumen, es necesario que el equipo de desarrollo y el cliente conciban la seguridad como una propiedad intrínseca del sistema y no como algo a validar sobre el final para complacer al departamento de seguridad informática con los resultados positivos de un *pentesting*. Por otro lado, es necesario que la seguridad se incorpore como parte de un proceso y que la organización valide con cierta regularidad la situación de sus aplicaciones a los fines de minimizar los riesgos de ser vulnerados. La seguridad nunca es una acción puntual, sino que es un continuo debe ser *parte* de cada etapa del proceso de desarrollo al igual que la calidad.

Como puede apreciarse, esta experiencia sirvió como motor para cuestionar la validez de este enfoque tan común en nuestros ciclos de desarrollo y nos motiva a pensar modificaciones en el *framework* de desarrollo ágil que fueron incorporadas en subsiguientes proyectos y que son expresadas en este trabajo.

Mejorando Scrum

Esta sección del trabajo se enfoca en revisar varios cambios en el *framework* de Scrum con el objeto de atender ciertos problemas identificados en los capítulos anteriores y que surgen fruto de varios años de experiencia, interés y experimentación en las diversas expresiones de la seguridad informática en las organizaciones.

Creando lo imposible - System Stories

En la comunidad de practicantes de Scrum existe un concepto muy arraigado y ampliamente difundido que refiere al uso de *User Stories* como método fundamental para la captura de requerimientos de una aplicación en particular. En las *User Stories* se

obtiene información sobre lo que un usuario en particular o, más precisamente, lo que un tipo de usuario puede necesitar o esperar que haga el sistema.

El uso de *User Stories* es muy poderoso en términos de capturar lo que el usuario realmente quiere obtener más allá de cómo el sistema va a lograrlo. Esta forma de captura brinda a los equipos de desarrollo gran flexibilidad para solucionar los problemas de los usuarios, pues los requerimientos son expresados en función de la necesidad y no se le dicta al programador cómo se debe manifestar la solución.

Esta flexibilidad es uno de los componentes fundamentales del desarrollo ágil, y probablemente motor de grandes aplicaciones que fueron desarrolladas en los últimos años, en las que surge una nueva forma de resolver un problema existente. Las *User Stories* son muy poderosas para comprender el problema real y fomentar la creatividad en la solución.

Por otro lado, la experiencia demuestra que aplicaciones desarrolladas con este enfoque tienden a ser menos seguras, pues rara vez el usuario va a incorporar una *User Story* que describa la seguridad de la aplicación, ya sea porque desconozca el tema y las potenciales amenazas o porque simplemente no es algo que le parezca relevante destacar. De igual forma, el usuario no siempre define la “calidad” o la “performance” de una aplicación, aunque estas dos últimas están mucho más desarrolladas como conceptos en la mente de los usuarios que la seguridad.

Con la presente información es que este autor propone la incorporación de *System Stories,* cuyo objetivo es acompañar la captura de requerimientos, pero desde el punto de vista técnico. Es formalizar un proceso que pone de manifiesto la visión técnica del producto y acerca la posibilidad de contemplar requerimientos técnicos que el personal no especializado en la materia de seguridad desconoce.

Esta discusión lleva varios años entre los practicantes de Scrum, quienes debaten si lo importante es lo que los usuarios perciben como valioso y en consecuencia todo lo demás debe alinearse con lo que el usuario plantea. Sucede que ya hemos recorrido este camino por años y la experiencia nos muestra que el usuario, ya sea por desconocimiento o por falta de recursos técnicos, es normalmente incapaz de presentar

requerimientos que refieran a la seguridad un sistema y de la información que este contiene.

Es por ello que se plantea la necesidad de incorporar al *backlog* las necesidades técnicas y de seguridad que no van a surgir de los usuarios pero que son fundamentales de tener en la lista de "cosas para hacer" (léase, el *backlog* del sistema) para ser priorizadas y discutidas entre equipos técnicos y de negocios.

Con lo mencionado anteriormente, se plantea la necesidad de utilizar *System Stories* sin alejarse de lo que resulta valioso para los usuarios, pues ello significaría destruir el concepto mismo de Scrum, que debe, sin dudas, ser liderado por los requerimientos de los usuarios y sus necesidades. Para profundizar la comprensión de la idea de *System Stories* se promueve el uso de la siguiente sintaxis:

*To allow <**functionality**>, we need <**task**>, so we can support <**User Story**>.*

En español: Para permitir la **FUNCIONALIDAD**, necesitamos esta **TAREA**, para poder realizar el **USER STORY**

Nótese cómo, independientemente de la creación de esta nueva entidad en el *backlog* llamada *System Story*, ella debe en todo momento guardar relación con una necesidad planteada por el usuario en la forma de una *User Story*. Esta modificación sutil pero efectiva transforma la realidad del *backlog* de una lista de pedidos y necesidades de usuarios, a una entidad mixta que contiene también los deseos y necesidades de los equipos técnicos.

Con este cambio, logramos devolverle voz a las ramas técnicas que habían quedado silenciadas bajo la complacencia de utilizar su tiempo únicamente para generar valor a través de *User Stories* que brindan funcionalidades a los usuarios. Ahora en las discusiones sobre qué se debe hacer en cada *Sprint*, al menos están listadas las necesidades técnicas del equipo para satisfacer esas funcionalidades con mayor racionalidad.

Para el lector acostumbrado a los procesos de desarrollo, la presente modificación no parecerá significativa, pues equipos maduros de programadores suelen tener estas consideraciones por fuera del *backlog* a modo tal de que el negocio decida cuáles son las prioridades de los usuarios y ellos internamente durante el desarrollo se encargarán de generar todas las tareas asociadas con la factibilidad técnica de la funcionalidad requerida por los usuarios.

En este sentido, es normal considerar que las mejoras propuestas no son suficientes para generar un cambio significativo en la seguridad de las aplicaciones. Y cualquier lector que llegue a esta conclusión estaría probablemente en lo cierto. No obstante, el siguiente paso es incorporar al equipo personal técnico con conocimientos en seguridad informática, y a este punto se refiere la siguiente sección.

Finalmente, resulta importante destacar que el rol del *Product Owner* y la gestión de dependencias/prioridades se vuelve esencial para el éxito de este enfoque. A menos que el equipo sea capaz de transmitir la importancia de las *System Stories*, y como consecuencia el *Product Owner* las vea como dependencias directas para las distintas *User Stories*, el valor de este enfoque está en riesgo, dada la baja prioridad que se le puede asignar a las *System Stories* asociadas a la seguridad.

Sumando a los expertos en Seguridad

Ningún proyecto va a incorporar a la seguridad informática a menos que haya alguien que, contando con los conocimientos en la materia, traiga a la mesa las posibles amenazas a las que la aplicación puede estar expuesta, o bien comprenda el escenario digital en el que la aplicación estará corriendo, a los fines de exponer los potenciales desafíos de seguridad que enfrentará durante sus años en producción.

En un futuro, es esperable que los usuarios ganen mayor conocimiento sobre la seguridad informática y desde su propio saber e interés soliciten funcionalidades relacionadas con ella. De momento, en ausencia de lo anteriormente expuesto, lo más seguro y recomendable es traer a la discusión a expertos en la materia que puedan apoyar y fomentar intercambios que permitan entender las necesidades de proteger un sistema y el valor de la información que este maneja.

Para sumar a los expertos en seguridad, es necesario comprender que actualmente no cuentan con la formación en metodologías ágiles ni en los procesos tradicionales de desarrollo, por lo que se vuelve necesario crear un marco procedural que fomente su participación pero que no implique demoras o dificultades para el resto del desarrollo a realizarse.

Con estas consideraciones en mente es que el autor de este documento pone en valor la idea de *System Stories* en conjunto con la incorporación de personal de seguridad en los equipos de desarrollo a tiempo parcial. Nótese que es muy costoso y poco aplicable el tener un recurso experto en seguridad informática en cada proyecto de desarrollo, y que normalmente un experto va a participar en más de un proyecto a la vez. Esto, en algún punto, atenta contra la idea tradicional de Scrum, pero algunos sacrificios han de ser considerados en función de mejorar la seguridad de nuestras aplicaciones.

Para alinear lo anteriormente mencionado, vemos entonces que el experto en seguridad ahora se puede dedicar a revisar las *User Stories* y en base a ellas crear una lista de *System Stories* para que el resto del equipo las revise e implemente a medida que van seleccionando las *User Stories* a realizar. De esta forma se alcanzan varios objetivos en simultáneo, a saber:

- Gestión de progreso
    - Con un backlog de seguridad claramente definido es mucho más fácil medir el progreso de las tareas que el equipo de desarrollo va realizando tanto en materia de *User Stories* como de *System Stories*. Esto facilita la tarea del experto en seguridad, quien puede monitorear solo aquellas *System Stories* que refieren a la seguridad informática.
- Gestión de prioridades
    - Permite al experto manifestar sus conocimientos y plantear las exposiciones a riesgos, mientras que el *Product Owner* será el responsable final de definir lo que será incorporado en cada *Sprint* y, por lo tanto, responsable final del desarrollo, teniendo a la vista una

representación de los requerimientos en seguridad (y otras temáticas) expuestas de forma clara y comprensible.

- Gestión de dependencias
    - Al visibilizar claramente el vínculo entre los requerimientos del usuario (*User Stories*) y los requerimientos del sistema para satisfacer los primeros (*System Story*), el *Product Owner* y el cliente pueden ponderar prioridades con mejor información que si dicha relación es mantenida en la oscuridad del área técnica.
- Gestión de riesgos
    - Al definir las *System Stories* de seguridad, puede apreciarse que casi se está definiendo la lista de riesgos de seguridad informática a los que está o estará expuesta la aplicación. Esto servirá para facilitar el análisis de exposición al riesgo al momento de salir a producción, permitiéndole a todos los *stakeholders* tomar decisiones informadas en lugar de esperar los resultados de un *pentesting* dos semanas antes de salir a producción.

La seguridad como concepto de calidad

Conforme a lo descrito anteriormente, el lector estará familiarizado con la idea de que la seguridad debería asemejarse a la calidad en el sentido de que los mecanismos que rigen la calidad no son siempre solicitados por el usuario o cliente y, sin embargo, son incorporados en los procesos pues se considera inaceptable un sistema que tiene *bugs*.

Proponemos, por eso, el alineamiento entre la gestión de la calidad y la gestión de la seguridad en Scrum.

En esta metodología, existen varios conceptos asociados con la calidad que propongo repensar como herramientas complementarias para mejorar la probabilidad de desarrollar aplicaciones más seguras.

Definition of Done (Definición de Terminado)

Esta es una pieza fundamental de la gestión de la calidad, pues el equipo debe ponerse de acuerdo en cuál es la *Definition of Done* (DoD), a modo tal que todos comprendan y puedan asegurarse de que, cuando algo está considerado *terminado*, cuenta con una serie de estándares que garantizan su calidad.

Aquí debemos tratar de incorporar algunas medidas de seguridad, pero sin perder de vista que lo *terminado* debe ser entendido desde el punto de vista del requerimiento y enfocado al usuario. Es por ello que poner aquí una validación, como *penetration testing* completo, no tendría sentido, pues ningún equipo sería capaz de terminar una tarea a tiempo. Quizá es posible pensar en herramientas de automatización para trabajar estas problemáticas y lentamente incorporar cuestiones de seguridad a la DoD, pero en el corto plazo esto es poco probable.

Con esto en mente, se propuso que los aspectos realmente técnicos relacionados a una tarea que sean necesarios, pero no fundamentales sean capturados y manejados como una *System Story*. De esta forma, quienes deben tomar decisiones pueden evaluar la importancia de los diferentes aspectos relacionados con una funcionalidad.

Unit Testing

Estas líneas refieren más a un desafío técnico que a uno metodológico, pero rescatan la necesidad de comenzar a incorporar el testeo de vulnerabilidades como parte de las validaciones que se realizan de forma automática a las aplicaciones, previamente a cada pasaje a producción (u otro ambiente, según sea necesario).

Las automatizaciones son un aliado fundamental de las metodologías ágiles y deben, por lo tanto, ser gradualmente incorporadas en los procesos de desarrollo facilitando la visualización de las vulnerabilidades más obvias.

El rol de Tester

En los equipos de Scrum, se acostumbra a decir que no existen roles definidos como testers o desarrolladores, sino que simplemente todos son parte de un equipo y

todos deberían poder realizar todas las tareas. Lo cierto es que existe lugar para la especialización, pero en el ideal de Scrum está el concepto de miembros de equipo y no el de roles asignados, ya que cada uno es lo que el equipo y el proyecto necesitan. Ese es el verdadero espíritu de la metodología.

Sin embargo, en la práctica, los profesionales nos encontramos con realidades muy distintas en la mayoría de las organizaciones, donde el personal ha sido contratado con funciones específicas y, por lo tanto, su rol dentro del equipo y el proyecto está definido de antemano.

Por otro lado, ¿qué pasaría con los *pentesters*? Resulta mucho más difícil asignar un *pentester* a un único proyecto. Para empezar, son terriblemente escasos y costosos, de modo tal que pocos proyectos podrían justificar la inversión. Además, lo más probable es que no dediquen totalidad de su jornada al *pentesting* y sería muy complejo asignar el resto de sus horas a tareas más económicas como desarrollo, pues el costo de su hora es muy superior a la del desarrollador, lo que significaría una gestión incorrecta de recursos. Su tiempo sería mejor aprovechado si el profesional realizara *pentesting* para otro proyecto que si desarrollara dentro del mismo proyecto otras funcionalidades no asociadas a la seguridad informática.

Alternativas de implementación

Deberá ser estudio de otro trabajo de investigación el cómo incorporar *pentesters* y hacerlos económicamente rentables dentro de los presupuestos de los proyectos, por el momento este documento se limita a identificar esta dificultad y subrayar los desafíos que se le presentan a quien busca acercar la seguridad como concepto a la calidad. Se enfrenta con la complejidad de gestionar recursos muy escasos y costosos que son a su vez prácticamente fundamentales para validar las medidas de seguridad incluidas en la aplicación que se está desarrollando. Una posibilidad sería incorporarlos al menos en un *Sprint* completo, quizá una vez cada dos, tres o cuatro *Sprints*, de acuerdo con la duración del proyecto, la disponibilidad de recursos y la criticidad del sistema.

Desde la experiencia, se hace evidente que contar con un *pentester* solamente por unos días no es suficiente y, desde luego, contar con unas semanas sobre el final de proyecto ya fue descartado como práctica respetable en la gestión de proyectos para el desarrollo de software seguro. La mejor opción, según la mirada de este autor, es incorporarlos un S*print* completo (normalmente un periodo de 2 a 4 semanas dependiendo lo definido por el equipo), a modo de facilitar una integración más eficiente con el equipo de desarrollo y trabajar junto a ellos por un periodo que tiene un objetivo concreto y que facilita la posibilidad de medir su progreso y valor agregado.

Conclusiones

El lector experimentado en gestión de desarrollos comprenderá que en algún punto el planteo que se busca enfatizar es que hay que devolver un poco de poder a los equipos técnicos, que en los últimos años le fue cedido a los equipos de negocios o a los usuarios, fijando prioridades y tiempos con el objeto de maximizar el *time-to-market* y el ROI de un desarrollo. Vemos esto como una significativa mejora en los productos en general, pues permite a los usuarios ser dueños de las aplicaciones definiendo las prioridades y las funcionalidades a ser desarrolladas; no obstante, la situación se ha desbalanceado permitiendo la salida a producción de sistemas que cumplen con todos los requerimientos de usuarios pero que difícilmente serían validados por personal técnicamente avanzado en ciencias de la computación y la gestión de la información.

Analizando la realidad actual, y tomando como base la experiencia personal, es evidente que resulta realmente complejo incorporar personal con experiencia en seguridad en procesos de desarrollo ágil, pues generalmente carecen formación en desarrollo de aplicaciones tradicionales y, más crítico aun, en procesos ágiles de desarrollo. Esta situación, conocida por la mayoría de los profesionales que lideran proyectos, repercute en la impresión que el equipo de desarrollo tiene del personal de seguridad: es visto como un factor de demoras y retrasos en lugar de un factor imprescindible de éxito para la organización y principalmente para el cliente.

Si la incorporación de la seguridad es percibida como una molestia que afecta significativamente la idea de velocidad que se tiene de un proyecto de desarrollo de

software, es probable que sea uno de los primeros recortes que reciba un proyecto que está bajo presión. Por lo tanto, nuestro trabajo es el de incorporar la seguridad tan fuertemente en el proceso, que su ausencia este considerada como una mala práctica.

Este documento pretendió acercar al lector algunos consejos prácticos sobre cómo incorporar la seguridad informática en procesos de desarrollos ágiles, particularmente Scrum. Estos se orientan a favorecer un giro en la visión que se posee del área, para que se reconozca en ella una aliada estratégica del negocio y un pilar fundamental del éxito y la supervivencia de las organizaciones en el largo plazo.

Glosario

Término	Definición
Framework	Marco o esquema. Utilizado en el contexto de este documento como: esquemas de gestión o desarrollo. Presentan un esquema de procesos o metodologías que apoyan y aceleran la gestión del desarrollo de software.
Penetration testing	Practica utilizada por personal de seguridad informática donde se intenta encontrar vulnerabilidades en un sistema mediante la prueba de diversas técnicas de acceso. Generalmente contratados por la empresa que financia el desarrollo para la búsqueda de vulnerabilidades en el sistema, sus reportes permiten hacer más seguro el producto, implementando sus recomendaciones.
Roadmap	En el marco de este documento, refiere al plan o a los próximos pasos a seguir para la realización de un determinado proyecto.
Performance	En tecnología, normalmente este término expresa la velocidad de respuesta de un sistema en particular, dada una interacción con el usuario. Por ejemplo, el tiempo que toma al sistema mostrar 1000 resultados luego que el usuario presionó un botón.
Manager	En el marco de este documento, refiere al responsable de un proyecto o un equipo. Aquel que toma las decisiones y lidera el desarrollo del proyecto.
Stakeholders	Refiere a interesados en el proyecto que no necesariamente están activamente involucrados, pero que se ven afectados por sus resultados.
Apps	En el marco de este documento, son sistemas desarrollados específicamente para dispositivos móviles. En los últimos tiempos esta definición se ha extendido y también se refiere a otro tipo de sistemas de tamaño limitado, que pueden o no correr solo en dispositivos móviles.

Término	Definición
Feedback	En el marco de este documento, refiere a la retroalimentación de información con respecto a la satisfacción con el producto o servicio, generalmente por parte de los usuarios de un sistema.

Referencias

[1] N. Wirth, "A Brief History of Software Engineering," *Methodic and Didactic Challenges of the History of Informatics, Austrian Computer Society,* p. 178, 2007.

[2] G. E. Moore, «Cramming more components onto integrated circuits,» *Electronics,,* vol. 38, nº 8, 1965.

[3] E. W.Dijkstra, *How do we tell truths that might hurt?,* 1975.

[4] D. R. G.T. Gangemi, Computer Security Basics, O'Reilly Media, 1991.

[5] C. Herley, «So Long, And No Thanks for the Externalities:,» *NSPW,* 2009.

[6] C. Herley, «Unfalsifiability of security claims,» *Microsoft Research,* 2016.

[7] K. S. a. J. Sutherland, «The Scrum Guide,» Scrum.Org and ScrumInc, 2013.

[8] T. S. Group, «CHAOS,» *Project Smart,* 1995.

[9] M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley Professional, 2009.

[10] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley Professional;, 2012.

[11] K. Mitnick, The Art of Deception, Wiley, 2003.

[12] O. S. J. R. J. W. Pekka Abrahamsson, «Agile Software Development Methods: Review and Analysis,» *VTT Publications,* 2002.

[13] P. K. Konstantin Beznosov, «Towards Agile Security Assurance,» *University of British Columbia,* 2004.

[14] M. Dlamini, *Information security: The moving target,* 2009.

[15] R. A. a. T. Moore, «The Economics of Information Security,» *Science,* 2006.

[16] R. J. Anderson, «Why Information Security is Hard,» *ACSAC,* 2001.

[17] C. H. P. C. v. O. F. S. Joseph Bonneau, «The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes,» *IEEE Symposium on Security and Privacy,* 2012.

[18] C. H. B. C. Dinei A. F. Florêncio, «Do Strong Web Passwords Accomplish Anything?,» *HotSec,* 2007.

[19] C. H. Dinei A. F. Florêncio, «Password Rescue: A New Approach to Phishing Prevention,» *HotSec,* 2006.

[20] C. H. Dinei A. F. Florêncio, «Where Do Security Policies Come From?,» *SOUPS,* 2010.

[21] J. W. M. T. S. J. R. Pekka Abrahamsson, «New Directions on Agile Methods: A Comparative Analysis,» *ICSE,* 2003.

[22] P. M. Institute, Project Management Body of Knowledge - 6th Edition, PMI, 2017.