Seguridad (Resiliencia) en el ciclo de vida del software

Recomendaciones y buenas prácticas de seguridad, continuidad y privacidad de la información que deben ser tenidas en cuenta en el proceso de ciclo de vida del software

Milton Quiroga

Primera impresión, Ene 2018

Tabla de Contenido

1. Introducción

> Without privacy there was no point in being an individual
>
> *Jonathan Franzen*

La seguridad de la información es un atributo de calidad de cada sistema de información y debe ser tratado como un atributo de primer orden. Sin embargo, comúnmente se asume que puede ser tratado como una "capa arquitectónica" -una especie de *"band-aid"* que se añade al final de la implementación- "si queda tiempo".

Esto se traduce con frecuencia en varias debilidades. La más evidente se produce por la mentalidad tras la frase "si queda tiempo". Los controles de seguridad se dejan para el final y cuando un requerimiento funcional se extiende más de lo normal, tiende a consumir el tiempo que se habría podido dedicar a actividades de seguridad. Esto se traduce en el despliegue en ambientes productivos de aplicaciones con seguridad débil o -con más frecuencia- abiertamente vulnerables.

Pero no es el único efecto. Es posible -por ejemplo- encontrarse con la idea de buscar una solución de seguridad universal que asegure todos los componentes del sistema ante todos los ataques que puedan efectuar todos los adversarios. No solamente esa visión es utópica, sino que aquellos que la promueven tienden a ignorar quiénes son los adversarios, qué pueden querer y cómo pueden comprometer la aplicación.

Pero, con toda certeza, el mayor problema está en el diseño de seguridad o en su ausencia. Nadie diseñaría una aplicación seria sin considerar antes los requerimientos de concurrencia o latencia, por citar algunos ejemplos. ¿Por qué, entonces, pensar en seguridad solamente cuando el producto está terminado? Es necesario comprender que todas las fases del proceso de construcción, implementación y aun operación del software son fases en las que se pueden introducir bugs y vulnerabilidades. La mayoría de metodologías permite lidiar con los primeros de forma razonable, pero tiende a hacer caso omiso de los segundos.

Para evitar estos errores en la aproximación a la seguridad, que normalmente se traducen en controles insuficientes o inexistentes, este documento busca presentar la seguridad como un atributo de calidad del sistema de información.

Dicho de otra forma, no se trata de afirmar que la seguridad es el elemento regente que debe determinar todas las decisiones que se tomen en un sistema dado, sino de hacer notar que se trata

de un factor muy importante y que debe ser tenido en cuenta desde el principio de un proyecto de construcción de software. Habrá compromisos en el diseño (en adelante referidos como *tradeoffs*): en ocasiones, habrá que sacrificar seguridad para favorecer -por ejemplo- tiempos de respuesta. Lo importante es que cada situación sea considerada y las decisiones se tomen de manera objetiva y acorde a las necesidades de la solución, sin que quede ninguna suposición tácita o no expresa.

Sin embargo, es importante notar que de la misma manera en que se debe considerar la seguridad como un atributo de calidad presente en todo el ciclo de vida del software, es importante considerar también los aspectos de continuidad y cumplimiento como atributos de calidad, que juntos componen un problema más completo: el problema de la gestión de riesgo operacional en todo el ciclo de vida del software.

Este documento presenta un marco teórico de los principios de gestión de riesgo operacional aplicados al ciclo de vida del software. Este marco teórico está basado en dos grandes conjuntos de ideas: los principios de software seguro y los principios de desarrollo ágil tomados de *"lean manufacturing"* y *"scrum"*.

El resto de este documento está organizado en seis capítulos como sigue:

- En el capítulo 2 "Conceptos de Seguridad, Privacidad y Resiliencia" se definen algunos de los conceptos relacionados con seguridad, privacidad y resiliencia organizacional.
- En el capítulo 3 "Riesgo Operacional" se presenta una descripción detallada de los principios de gestión de riesgo operacional en una organización típica.
- En el capítulo 4 "Descripción de Procesos y BPMN", se presenta una descripción de la notación BPMN de procesos. Esta notación es usada de manera intensiva en el documento.
- En el capítulo 5 "Desarrollo moderno de software", se presentan algunas de las tendencias más importantes en ingeniería de software: "lean", desarrollo ágil, "scrum", TDD (*test-driven development*), y CI (*continuous integration*).
- En el capítulo 6 "Seguridad (Resiliencia) y el ciclo de vida del software", se muestran los conceptos del ciclo de vida del software y cómo la tendencia a incorporación temprana de requerimientos coincide bastante bien con las necesidades de gestión de riesgo operacional de una organización.
- En el capítulo 7 "Conclusiones", se presentan las principales conclusiones de este estudio.
- Finalmente, en el capítulo 8 "Bibliografía", se presentan las principales referencias bibliográficas seguidas.

1.1 Convenciones Tipográficas

Dado el carácter técnico de este documento, en él se hace necesario el uso de contenido textual que se desvía del español tradicional. Estos desvíos incluyen anglicismos, términos técnicos cuya definición difiere del significado tradicional de una palabra castiza, ejemplos de sintaxis de lenguajes de programación o representación, entre otros. Con el ánimo de hacer estas diferencias más explícitas y facilitarle al lector la lectura, a lo largo del documento haré uso de una serie de convenciones tipográficas, que se ilustran a continuación:

El uso de caracteres `monoespaciados` representa contenido literal, incluyendo, por ejemplo, nombres de variables, ejemplos de código, contenido XML, URLs, entre otros.

Los términos en inglés que el Autor prefiere no traducir, aparecen usualmente en *itálicas*.

En la versión PDF de este documento, todas las URLs, referencias a otras secciones del mismo documento y referencias bibliográficas contienen enlaces al contenido referido.

Para mayor legibilidad, los ejemplos de código o documentos en texto plano utilizan diversos

colores para resaltar diferentes elementos (palabras clave, valores literales, etc.). Adicionalmente, cuando los ejemplos abarcan varias líneas, el inicio de cada línea está numerado. En algunos casos, cuando las líneas son más largas que el espacio de la página, se hace uso del caracter ↪ para representar la combinación de una línea demasiado larga.

En expresiones entre paréntesis se hace uso con frecuencia de abreviaturas literarias. Aunque su uso es común en la literatura técnica, se describen en este punto para prevenir ambigüedades:

- *v.g.* - Ejemplifica la frase que precede al paréntesis. Puede ser interpretada como "por ejemplo:".
- *i.e.* - Aclara la frase que precede al paréntesis. Puede ser interpretada como "es decir," o "en otras palabras,".
- *a.k.a.* - Esta abreviatura se toma de la lengua inglesa, al no haber un equivalente semántico en el español. Se deriva de la locución *also known as* (que puede traducirse como "también conocido como") y describe otros términos que, en el uso popular, se comportan como sinónimos o equivalentes del término que precede a los paréntesis.

En algunos puntos del documento se introducen anotaciones que se salen del flujo del documento con el objetivo de resaltar un aspecto específico del tema que se está tratando. Estos *comentarios* están identificados con una letra `C` encerrada en un círculo azul.

1.2 ¿Cómo usar este documento?

Este es fundamentalmente un documento de referencia y consulta para cualquier proyecto de software que tenga que ver con "seguridad por diseño", "privacidad por diseño" y "seguridad en el ciclo de vida del software".

1.3 Contactar el Autor

Todos los comentarios son bienvenidos vía mail a `mquiroga@cyte.co`, o alternativamente a `mquiroga@uniandes.edu.co`.

1.4 Colofón de Derechos de Autor

Las imágenes de este documento fueron en buena parte elaboradas por el Autor, algunas fueron descargadas de Wikimedia usando la licencia CC (el autor correspondiente se halla debidamente acreditado bajo la imagen) y otras más fueron pagadas y licenciadas comercialmente de Shutterstock (`https://www.shutterstock.com/home`), de acuerdo con la licencia estándar (`https://www.shutterstock.com/license-comparison`).

Los correspondientes créditos son "Tom Wang/172626641", "Gajus/341112518", " Yuganov Konstantin/105970766", "Maksim Kabakou/127894739", "Christos Georghiou/242206567", "Dushkapampushka/421143067", "Elnur/561677989" y " Maksim Kabakou/349139351".

Este documento fue compuesto en su totalidad utilizando las herramientas libres de LaTeX (`https://www.latex-project.org/lppl/`) creadas inicialmente por Leslie Lamport, utilizando el IDE Eclipse (`https://www.eclipse.org/legal/epl-v10.html`), con el plugin TeXlipse (`http://texlipse.sourceforge.net`).

El autor manifiesta su agradecimiento público a los creadores de estas fabulosas herramientas fuente abierta.

2. Conceptos de Seguridad y Privacidad

National security is the fig leaf against freedom of information

Ralph Nader

2.1 ¿Qué es seguridad?

La seguridad de la información (a.k.a. *infosec*) es una disciplina que busca prevenir el uso, el acceso no-autorizado, la divulgación, la interrupción, la modificación, la inspección o la destrucción de información considerada sensible, independientemente de si esta información se almacena en medios electrónicos (discos duros) o medios físicos (papel).

El énfasis fundamental de la disciplina infosec es conseguir el balance adecuado de protección de la triada CIA (*Confidenciality*, *Integrity* y *Availability*) y la productividad de los procesos de una organización.

A su vez, de acuerdo con [27016], la triada CIA se define como:

- la Confidencialidad consiste en la propiedad que permite que la información no es visible para individuos, entidades o procesos no autorizados. Es la propiedad más frecuentemente asociada a **seguridad**.
- la Integridad o Integridad de datos consiste en la propiedad que permite que la información sea precisa y completa durante todo su ciclo de vida y no pueda ser modificada de manera no autorizada o no detectada.
- la Disponibilidad es la propiedad que permite que la información esté lista cuando se necesite. Por tanto requiere que funcionen correctamente tanto los sistemas de almacenamiento de información, los controles de acceso y los canales de comunicaciones de acceso a dicha información. Un sistema con alta disponibilidad requiere no solo evitar interrupciones por apagones y fallas técnicas sino también controlar ataques de denegación del servicio.

De esta primera definición de seguridad es importante destacar el énfasis en “la productividad” de los procesos de negocio de una organización... i.e. prima la funcionalidad sobre la seguridad.

El reto de la gestión infosec de una organización consiste en conseguir que sobre un servicio informático se tengan los controles que garanticen que además de ser un servicio útil, sea seguro.

Por otro lado se habla también de uso "no-autorizado" de "información sensible". Este aspecto de infosec tiene que ver con el hecho que un **incidente de seguridad** se define como una violación a una política de seguridad definida previamente, ya sea de manera tácita o de manera expresa.

Estas políticas de seguridad están definidas por el mismo negocio, algunas nacen directamente de la misión de la organización, otras -quizás más sútiles- se definen de manera explícita mediante un ejercicio de Análisis de Riesgos. Otras son tácitas y no completamente comprendidas, hasta que sucede un incidente que las pone de relieve.

Finalmente, la última implicación de la definición hecha tiene que ver con la estrecha relación entre seguridad y **continuidad**... en efecto ya habíamos mencionado que la disponibilidad tiene que ver con mantener la información lista aún ante eventos adversos relacionadas con apagones y fallas tecnológicas. En efecto, a propósitos de la presentación de los conceptos de este documento, se considera que la seguridad y la continuidad son aspectos complementarios de un concepto más amplio llamado **Resiliencia Organizacional** que abordaremos más adelante en el capítulo 3.

2.2 ¿Qué es privacidad?

La Privacidad de un individuo[1] consiste en el derecho que tiene cada cual a tener completo control de su información íntima y compartirla solo con quien considere pertinente bajo unas condiciones definidas por el individuo.

Es el derecho de un individuo a ocultar información acerca de si mismo que pueda ser usada ventajosamente por otros.

La legislación Colombiana es particularmente clara y concreta en el derecho a la privacidad o «habeas data». Al respecto la Constitución de 1991 en su artículo 15 (modificado por el Acto Legislativo 02 del 2003) señala específicamente "Todas las personas tienen derecho a su intimidad personal y familiar y a su buen nombre, y el Estado debe respetarlos y hacerlos respetar".

Aunque a primera vista, pareciera que la privacidad y la confidencialidad están estrechamente ligadas, en realidad se trata de principios diferentes que se traslapan muy tangencialmente: la privacidad es un derecho de un individuo, mientras la confidencialidad es una propiedad de cierto tipo de información.

Un individuo puede escoger voluntariamente entregar información privada, quizás en intercambio de algún tipo de beneficio: comparto mi fecha de nacimiento con Avianca para facilitar la expedición electrónica de tiquetes, pero este acto de compartir información se supone surge después de una decisión razonada acerca de los beneficios y los riesgos asociados.

En ningún caso debiera el estado difundir información identificable acerca de sus ciudadanos. Es claro que para su funcionamiento el estado está permanentemente recogiendo información de impuestos, salud, criminalidad, etc. de sus ciudadanos. Sin embargo esta información no debiera hacerse pública de forma accidental (a causa de un incidente de seguridad) o de forma intencional (en alguna iniciativa de "datos abiertos" como los descritos en la sección 2.3), sin pasar antes por un proceso de "saneamiento" (*«sanitizing»*) que garantice que a partir de los registros públicos no es posible identificar ciudadanos individuales.

La tecnología criptográfica que permite este saneamiento de datos se conoce como *differential privacy* (véase [Cyn14]) y consiste básicamente en un conjunto de procedimientos matemáticos

[1] se trata de un derecho de individuos, no de grupos sociales o de corporaciones

cuyo objetivo es maximizar la precisión de las consultas sobre una base de datos estadística, minimizando las posibilidades de identificar un registro individual con una probabilidad inferior a un valor dado.

La distopía futurista que puede resultar de una sociedad sin privacidad para sus ciudadanos es referida e imaginada por múltiples autores como George Orwell ([Orw61]) y Aldoux Huxley ([Hux06]). Más recientemente la Profesora Shoshana Zuboff de Harvard ha acuñado el término «Surveillance capitalism» (véase [Zub15]), para referir el fenómeno creciente de grandes organizaciones privadas que obtienen ingresos analizando grandes cantidades de información captadas mediante la vigilancia masiva de sus clientes.

2.3 Datos Abiertos

El objetivo de los datos abiertos es conseguir que toda aquella información del estado que no sea sensible, esté completamente disponible para ser usada y re-publicada por cualquier ciudadano, sin ningún tipo de restricciones de derechos de autor.

El uso de esta información pública y especialmente aquella captada por los gobiernos ha sido reconocida mundialmente como un factor importante de crecimiento económico (véase [CD07]) ya sea a través de emprendimientos privados basados en datos o simplemente como factor facilitador de procesos de Gobierno Abierto.

Consciente de este enorme potencial MinTICs en Colombia elaboró una «Guía de Datos Abiertos en Colombia» (véase [Min16]), con el propósito de proporcionar orientaciones y buenas prácticas acerca de la apertura y reuso de datos abiertos que generen valor social.

El marco legal de los datos abiertos en Colombia está basado en la ley 1712 del 2014 (véase [Rep14]). Si bien esta ley es clara en su énfasis en la transparencia de datos y derecho de acceso, también define dos tipos de información sensible que debieran estar exentos en toda iniciativa de datos abiertos. Estos tipos de datos corresponden a «información pública clasificada» (cuyo acceso podría afectar a individuos específicos) y «información pública reservada» (cuyo acceso podría afectar intereses públicos)[2].

En estos casos de información sensible, la guía [Min16] prescribe:

> "los datos abiertos no son contrarios a la protección de datos personales: si existe información asociada a datos personales pero que puede ser valiosa como dato abierto, adelante acciones de anonimización para eliminar aquella información sensible que afecta a personas u organizaciones y cuya identidad debe protegerse legalmente.
>
> Antes de publicar información que sea sensible, considere otras fuentes de información disponibles y evalúe si la combinación de éstas pueden presentar algún riesgo. Con esto puede prevenir el efecto mosaico, que ocurre cuando la información de una base de datos por sí sola, no genera un riesgo para la identificación de individuos, pero al combinarse con otra información disponible, puede generar tal riesgo".

Esta prescripción de MinTICs contiene varios elementos interesantes que es especialmente importante resaltar. En primer lugar habla de anonimizar información sensible. La forma correcta de anonimizar (en inglés *sanitize*) estos conjuntos de datos sensibles ya se mencionó en este documento y corresponde a las técnicas de *differential privacy* (véase [Cyn14]).

[2] Esta reglamentación de datos sensibles contrapuestos a datos abiertos no es exclusiva de nuestro país. El documento de la OECD ([CD07]) menciona por ejemplo que "Privacy concerns may require that access to data is limited to specific users or to sub-sets of the data".

En segundo lugar se utiliza el término «riesgo». En seguridad de la información la palabra riesgo tiene un significado muy preciso y supone un proceso previo de «Análisis de Riesgos»en el que el responsable de la gestión de cierto tipo de información realiza un estudio prospectivo **previo** acerca de los adversarios, amenazas y escenarios adversos que se pueden enfrentar en el caso en que esta información llegen a ser datos abiertos. Más adelante en la sección 6.1 se detallará un poco más acerca de las actividades de Análisis de Riesgos y su relación con la gestión de seguridad de la información.

2.4 Emprende con Datos

Por todos los beneficios que trae para Colombia el aprovechamiento de los datos abiertos, MinTICs creo la iniciativa «Emprende con Datos» que busca "fomentar el desarrollo de productos digitales a través de acompañamiento profesional en mentorías para emprendedores TIC, ayudando a resolver problemáticas de interés público y social a partir del uso de Datos abiertos de Gobierno" (véase `http://emprendecondatos.gov.co`).

Este programa le permite a emprendedores y empresas del estado, recibir apoyo de MinTICs en la concepción y ejecución inicial de proyectos de tecnologías de información que usen datos abiertos, para desarrollar productos y servicios que sean útiles para la economía Colombiana.

Sin embargo, tal como se había anotado en la sección anterior, estos emprendimientos deben pasar por una adecuada revisión de seguridad que garantice que los principios de uso de la «Guía de Datos Abiertos en Colombia» (véase [Min16]), se sigan con el adecuado rigor y cumplimiento.

2.5 Seguridad y Privacidad por Diseño

Los principios de «privacidad por diseño» (*Privacy by Design*) buscan incorporar los requerimientos de privacidad en todo el proceso de ingeniería de software (véase la definición en wikipedia [Wik17a]).

Estos principios hacen parte de la corriente filosófica de «*value sensitive design*»(véase [Wik17b]). Esta orientación de diseño plantea una serie de guías que buscan la incorporación en el diseño de cualquier tipo de tecnología, valores humanos claramente establecidos y compartidos por los involucrados.

Uno de los primeros estudios acerca de privacidad en el diseño de tecnología se debe a Cameron en su obra "*Laws of Identity*" ([Cam05]). En este blog Cameron plantea siete leyes esenciales que debiera tener un sistema de gestión de identidad digital, en las que los aspectos de privacidad juegan un rol primordial.

Posteriormente estas ideas fueron tomadas por el Gobierno de Ontario, que en un artículo muy breve de sólo dos páginas (véase [Cav11]), plantea una serie de siete principios fundacionales que constituyen la base de los principios de privacidad por diseño.

Estos principios son:

1. Proactivo, no reactivo: los problemas de seguridad deben anticiparse mediante un análisis de riesgos efectivo, no esperar a que el evento adverso suceda para tratar de remediarlo.
2. Privacidad pre-definida: El usuario no tiene que hacer nada para proteger su privacidad, toda su información personal está protegida.
3. Privacidad incluida en el diseño de cualquier aplicación: la privacidad de los usuarios es un componente esencial de la funcionalidad de una aplicación. No se trata de "un parche" que se pone al final para tratar de arreglar un diseño defectuoso.
4. Beneficios completos: el incluir protección de privacidad no debe de ninguna manera limitar la funcionalidad de la aplicación. Es una falacia considerar que hay un compromiso privacidad-funcionalidad o privacidad-seguridad.
5. Protección extremo a extremo durante todo el ciclo de vida.

6. Visibilidad y Transparencia: todos los controles de protección de privacidad deben ser visibles y susceptibles de ser verificados sin ninguna clase de limitaciones. Privacidad no es oscuridad.
7. Orientado al usuario: siempre los intereses del usuario están presentes en el diseño.

Sin embargo, aún cuando estos principios son intelectual y filosóficamente muy atrayentes, desafortunadamente su implementación desde el año 1995 (véase [ver95]) en que fueron sugeridos a la fecha, ha sido realmente lento y quizás decepcionante.

Por ejemplo el reporte "Privacy by design: delivering the promises" (véase [Hus10]) afirma textualmente:

> *we can observe a disappointing lack of progress in the uptake and practical use of PET in relevant areas*

.

Es decir, aunque se trata de un concepto adoptado y aceptado, sólo han habido avances muy incipientes en algunos proyectos en Canada, Alemania, Holanda y Reino Unido.

Es posible que una de las razones tenga que ver con la dificultad de poner estos principios en la práctica... Nótese como los siete principios son bastante abstractos y aunque describen claramente "el qué" de privacidad por diseño, faltan en el detalle de implementación que describe "el cómo" estos principios pueden ser incorporados en un ciclo de vida de software para datos abiertos.

Este documento busca precisamente aportar con una serie de guías metodológicas que constituyen "el cómo" los principios de privacidad por diseño (o más generalmente "resiliencia por diseño"), se pueden incorporar en un proceso de fábrica de software moderno.

Pero antes es necesario formalizar los conceptos relacionados con riesgo, seguridad de la información y resiliencia, temas de nuestro siguiente capítulo.

3. Gestión de Riesgo Operacional

The first line of Defense from insider threats is the employees themselves

CMU CyLab

En el año 2002, la escuela de negocios Sloan del Instituto Tecnológico de Massachusetts[1], realizó varias encuestas a ejecutivos de empresas de varios sectores de la economía, buscando identificar con ellos las principales preocupaciones que tenían respecto a las vulnerabilidades que amenazaban su negocio.

De los resultados de estas encuestas (documentadas en [She07]), se obtuvo un panorama bastante esclarecedor de las dificultades que tienen las organizaciones compitiendo en un ambiente cada vez más globalizado. Gráficamente estas preocupaciones tienen la apariencia de la figura 3.1.

Nótese como estas vulnerabilidades fueron organizadas en cuatro grandes grupos: vulnerabilidades financieras, vulnerabilidades estratégicas, vulnerabilidades que tienen que ver con vidas humanas (*hazard - safety*) y vulnerabilidades operacionales.

Las vulnerabilidades de tipo financiero en una organización tienen que ver con problemas de liquidez (caja), reformas tributarias adversas, problemas de convertibilidad de moneda, fluctuaciones en tasas de interés y en general todo tipo de problemas relacionados con la viabilidad financiera de una organización.

Por otro lado, las vulnerabilidades de tipo estratégico tienen que ver con el mercado en que la organización se mueve, la rivalidad de los competidores, productos sustitutos, fortaleza de proveedores y consumidores y en general las dimensiones de análisis de industria que surgen al considerar las fuerzas fundamentales de Porter[2].

Así también, las vulnerabilidades tipo *hazard* tienen que ver con los riesgos de seguridad industrial (*safety*) relacionados con pérdidas de vidas humanas y daños ambientales. Estos riesgos son especialmente relevantes en compañías de fabricación industrial y un ejemplo reciente de qué tan importantes son lo podemos encontrar en el incidente de seguridad industrial de Ecopetrol en Dos-quebradas (Risaralda)[3].

[1] http://mitsloan.mit.edu/

[2] http://www.investopedia.com/terms/p/porter.asp

[3] http://www.portafolio.co/economia/ecopetrol-lamento-tragedia-dosquebradas-risaralda

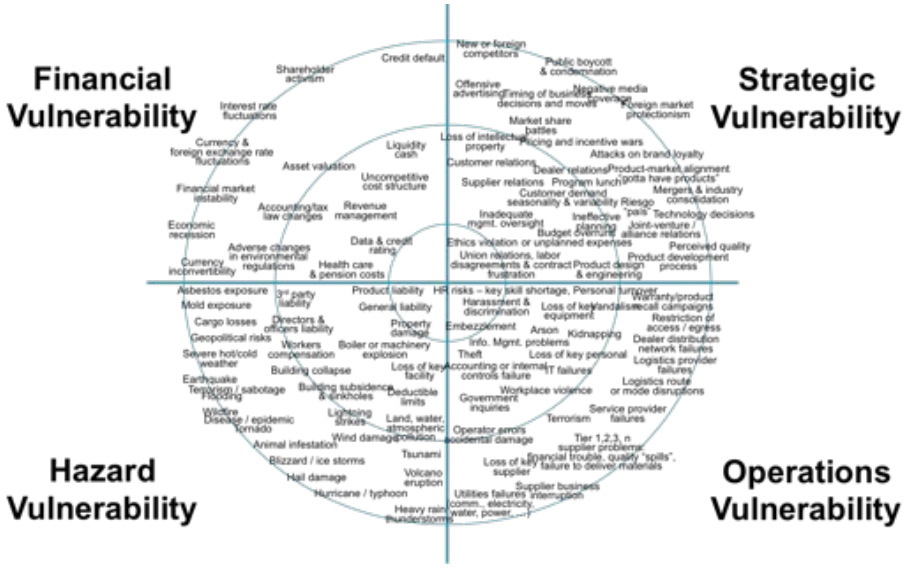

Figura 3.1: Vulnerabilidades y Riesgo Empresarial

Finalmente, las vulnerabilidades de tipo operacional son aquellas que tienen que ver con el día a día en la organización. Dentro de estas vulnerabilidades aparecen por ejemplo: problemas relacionados con fallas en la tecnología, problemas en seguridad de la información, desastres naturales, terrorismo y el impacto que tienen en las operaciones, fallas en recursos humanos y en general todas aquellas vulnerabilidades que están relacionadas con las operaciones que realiza una organización para cumplir su misión.

Esta interesante clasificación de los diferentes riesgos de una organización promedio fue adoptado para el sector financiero en el 2004 como los "acuerdos de Basilea" [Ris14], en donde se definieron dentro del primer pilar el manejo adecuado de Riesgos Operacionales, de Mercado, etc., como se muestra en la figura 3.2.

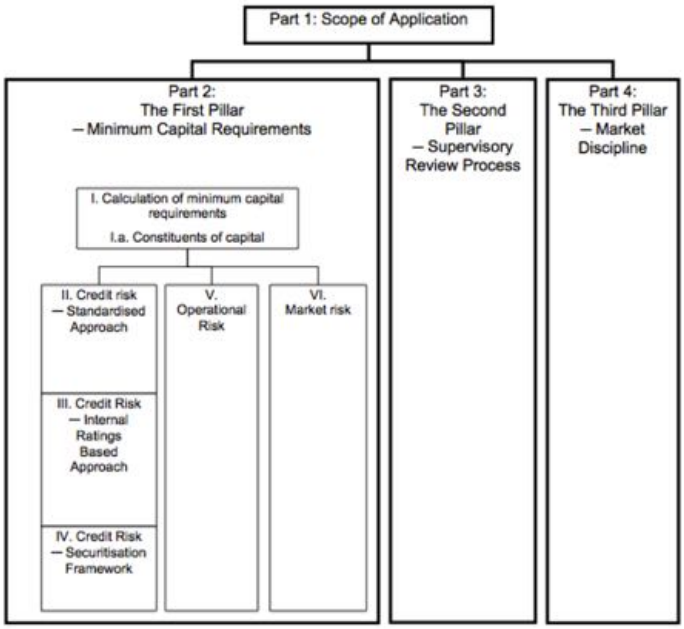

Figura 3.2: Riesgos y Basilea

Aunque si bien muchas de estas vulnerabilidades se traslapan y aparecen en ocasiones en dos grupos, son de particular interés para este estudio las vulnerabilidades de naturaleza operacional, destacadas en la figura 3.3.

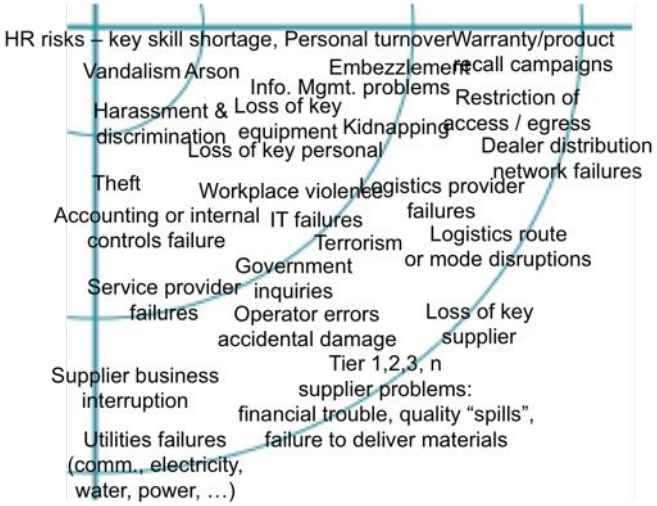

Figura 3.3: Vulnerabilidades y Riesgo Operacional

Estas vulnerabilidades de naturaleza operacional, fueron adoptadas en Basilea en el apartado 644 [Ris14], en donde se define Riesgo Operacional como *"the risk of loss resulting from inadequate or failed internal processes, people and systems or from external events"*.

De acuerdo con esta definición de Basilea y las conclusiones del estudio del MIT, se destaca la importancia de estas vulnerabilidades operacionales, toda vez que sobre el adecuado manejo que se dé a estas gravitan elementos vitales para la supervivencia de la misión de la organización como por ejemplo: confianza en la organización por parte de clientes y empleados, imagen y reputación externa, retención y crecimiento de clientes, cumplimiento del marco regulatorio y productividad y rentabilidad de la organización.

Es decir, aún cuando el manejo del riesgo operacional es un subconjunto de todo el manejo del riesgo corporativo, es sin embargo, un subconjunto bastante significativo del Universo completo de riesgos que una entidad financiera debe gestionar para cumplir su misión.

Gráficamente se muestra en la figura 3.4.

Figura 3.4: ORM es un subconjunto muy importante de ERM

De acuerdo con [al11], las fuentes principales de riesgo operacional en una organización se pueden categorizar a su vez en cuatro grandes grupos de problemas así:

- Fallas en tecnología: ocasionadas por vandalismo (hacking), mal mantenimiento de software y hardware, malas prácticas de ingeniería de software, inadecuada gestión de cambios o por inadecuada ingeniería de disponibilidad.

- Eventos externos: como por ejemplo desastres naturales, huracanes, terremotos, inundaciones, terrorismo, apagones, etc.
- Acciones de personas: que afectan negativamente los procesos productivos de la organización ya sea accidental o intencionalmente, de manera directa o indirecta, por error o por omisión, por ignorancia o por deshonestidad o por ausencia de liderazgo del personal.
- Procesos internos fallidos: por mal diseño o mala ejecución, errores, omisiones, mala o inexistente planeación de capacidad, fallas en el gobierno de un proceso o fallas en los procesos de soporte.

Es especialmente importante resaltar en este último grupo que con la tendencia a la tercerización existen posibilidades crecientes de fallas en los procesos de la organización, causados por mala planeación de la cadena de suministro o fallos en los servicios contratados con terceros. Basta con imaginar algún tipo de deslealtad por parte del "motorizado" empleado de una compañía de mensajería encargado de distribuir los extractos, para inferir toda una serie de ataques de graves consecuencias para una entidad financiera.

Gráficamente, estas cuatro categorías de Riesgo Operacional tienen la apariencia de la figura 3.5.

Figura 3.5: Categorías de Riesgo Operacional

Así el manejo adecuado de los Riesgos Operacionales es una característica muy deseable en una organización. Esta característica se conoce como Resiliencia Operacional y refiere a la capacidad que tiene una organización para sobreponerse a alteraciones en su operación causadas por:

- Problemas de seguridad de la información, vandalismo informático y hacking.
- Problemas de gestión de tecnología, como fallas y productos sub-estándar.
- Problemas de seguridad física en activos de tecnología críticos.
- Problemas de continuidad causados por apagones, terrorismo, desastres naturales, inundaciones, huracanes, etc.
- Impacto de nuevas regulaciones, como por ejemplo una nueva circular de la SuperFinanciera, una nueva ley, etc.

Como quiera que la Resiliencia Operacional está íntimamente ligada a la capacidad que se tiene para gestionar el Riesgo Operacional, entonces podemos concluir que para que una organización sea Resiliente, debe contar con un mecanismo adecuado y efectivo de gestión de Riesgos Operacionales.

Una organización con una adecuada gestión de riesgos operacionales, podrá planear por anticipado la materialización de estos riesgos sin que causen interrupciones graves. Así mismo, una organización con una gestión inadecuada de riesgos operacionales, no solamente es menos ágil y flexible, sino que también puede tener problemas de supervivencia, ya que la probabilidad

de interrupciones graves en sus operaciones se ve sustancialmente incrementada.

Por otro lado, y enfocando el tema desde una perspectiva más abstracta, es posible afirmar que una organización ofrece una serie de servicios que colectivamente componen la Misión de la organización. Así una organización es resiliente cuando los servicios que ofrece son resilientes, en el sentido en que son capaces de operar y cumplir su misión aún bajo efectos adversos y ser capaz de regresar a su operación usual una vez el efecto adverso es eliminado.

Simplificando de manera notoria estos componentes, podemos decir que los componentes de Resiliencia Operacional para una organización típica tienen la apariencia de la figura 3.6.

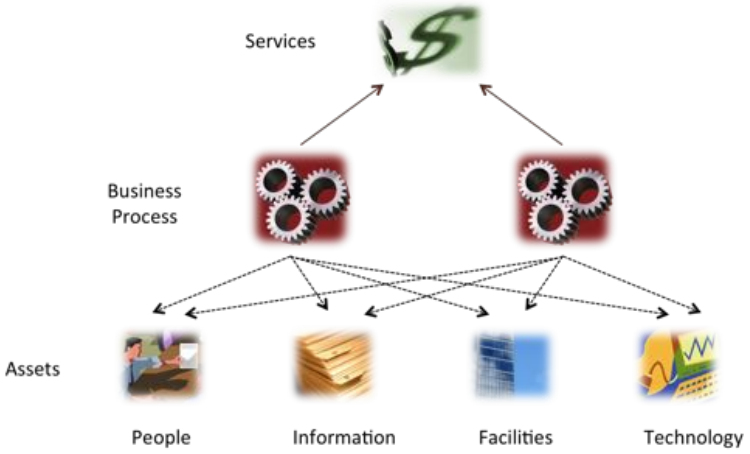

Figura 3.6: Los bloques fundamentales del Riesgo Operacional

En esta figura vemos una serie de activos (personal, información, instalaciones físicas y tecnología), que colectivamente apoyan una serie de procesos de negocios, que a su vez también una serie de servicios, que colectivamente componen la Misión de la organización.

Como quiera que es frecuente que una organización apoye alguno de sus procesos en servicios de terceros, entonces los bloques que típicamente componen la resiliencia operacional de la organización son los que aparecen a continuación en la figura 3.7 (nótese en la figura como un proceso de negocios puede estar apoyado en el servicio que ofrece un tercero).

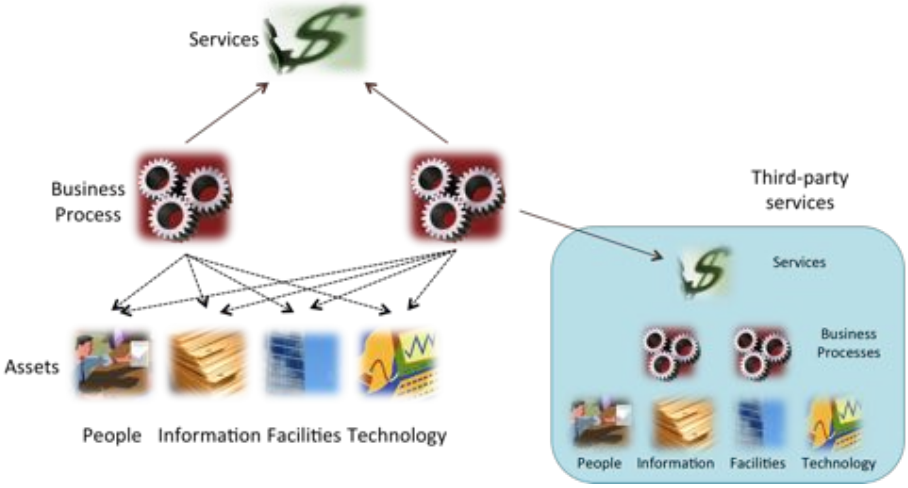

Figura 3.7: Riesgo Operacional y Terceras Partes

Es decir, desde otra perspectiva, es posible imaginar un esquema similar al de la figura 3.8,

en el que una serie de procesos de negocios, ofrecen una serie de servicios apoyados en activos organizacionales y colectivamente estos servicios componen la misión de la organización.

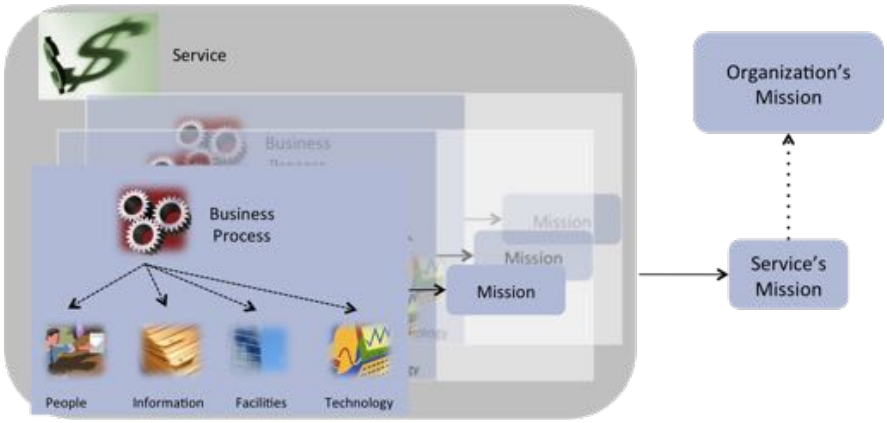

Figura 3.8: Los servicios, los procesos de negocio, los activos y la misión de la organización

Y entonces, volviendo a los conceptos de Resiliencia Operacional, una falla en un activo ocasionará interrupciones en los procesos que apoya, lo cual a su vez causará suspensión de los servicios ofrecidos por estos procesos, que por tanto pueden terminar impactando el cumplimiento de la misión de la organización en un claro efecto "cascada" de fallos e interrupciones, como se muestra a continuación en la figura 3.9.

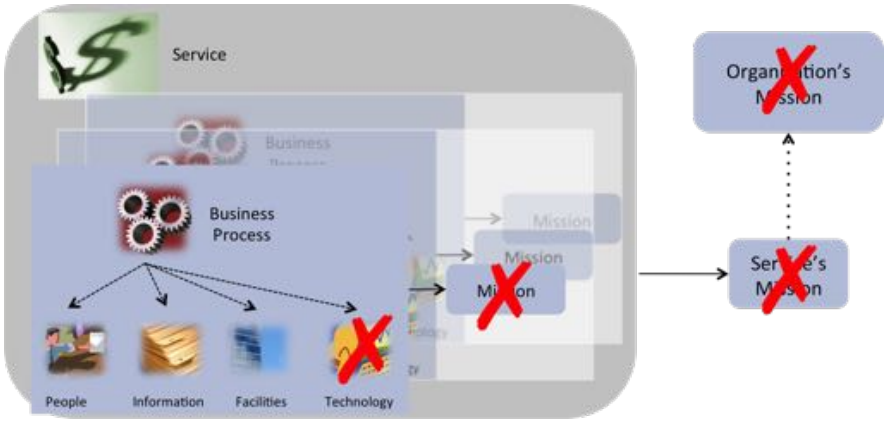

Figura 3.9: El efecto cascada de la falla de un activo en la misión de la organización

En resumen, todos los esfuerzos de gestión de riesgo operacional gravitan en torno a preservar los activos que soportan los procesos de negocio, que a su vez soportan la misión de la organización. Si la organización garantiza que los activos se mantienen operacionales, aún ante circunstancias difíciles relacionadas con seguridad o con continuidad, la organización será capaz de garantizar el cumplimiento de su misión.

Es decir, las actividades de gestión de riesgo operacional están relacionadas con actividades de protección (típicamente asociadas a seguridad de la información) y con actividades de preservación (típicamente asociadas a continuidad del negocio), como se muestra a continuación en la figura 3.10.

De forma más detallada estos dos grandes grupos de actividades se pueden descomponer en las siguientes tareas:

- Gestión de Activos

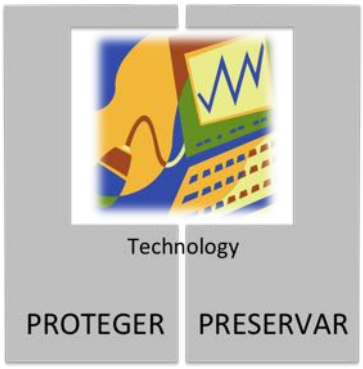

Figura 3.10: Seguridad de la información y continuidad de procesos

- Gestión de Requerimientos de Resiliencia
- Resiliencia y el ciclo de vida del software
- Gestión de Riesgos
- Gestión de Identidades
- Gestión de Acceso
- Gestión de Terceros
- Gestión de Incidentes
- Gestión de Vulnerabilidades
- Continuidad del Servicio
- Monitoreo y Mediciones

De estas tareas, las más estrechamente ligadas al ciclo de vida del software son Gestión de Activos, Gestión de Riesgos y Continuidad del servicio. A continuación se describen estas tres actividades.

3.1 Gestión de Activos

El propósito fundamental de los procedimientos de Gestión de Activos es identificar, documentar y administrar los activos organizacionales durante todo su ciclo de vida, para asegurar su correcto funcionamiento en el soporte de la misión de la organización.

El éxito en el cumplimiento de los servicios que soportan la misión, depende de la disponibilidad, productividad y en última instancia de la resiliencia de los activos que soportan los servicios: información para alimentar las decisiones del servicio, tecnología para soportar la automatización de los servicios, personas para ejecutar y monitorear el servicio e instalaciones en las cuales se pueda operar el servicio.

Por estas razones, la organización debe identificar los activos más valiosos, documentarlos y estimar su valor de acuerdo con los procesos que soporta y el valor de los servicios que dichos procesos ofrecen a la organización.

C La identificación de activos no es nueva para las entidades financieras Colombianas. Hace parte de las regulaciones exigidas por la SuperFinanciera y hace parte también de la norma ISO 27001, en la que se prescribe el requerimiento de hacer inventario de activos, establecer propiedad de los activos, definir uso aceptable, establecer directrices de clasificación y etiquetar los activos de acuerdo con sus requerimientos de seguridad.

Así dentro de esta tarea, es muy importante tener en cuenta los siguientes tres grandes grupos

de actividades:

- La primera actividad tiene que ver con identificar y definir los activos organizacionales relevantes para cumplir la misión. Para esto se debe realizar la identificación de los activos organizacionales para luego priorizarlos, documentarlos y generar el correspondiente inventario. Como se destacaba anteriormente, el éxito en la ejecución de la misión de una organización recae en las dependencias críticas entre servicios, procesos y activos de la organización. La falla de uno de los activos –por cualquiera de los eventos adversos asociados a riesgo operacional-, causa un efecto cascada que puede terminar impidiendo el logro de la misión de la organización. Por esta razón el primer paso en este bloque constructor consiste en identificar y establecer los activos del negocio y documentar sus requerimientos de seguridad / resiliencia.
- La segunda actividad a realizar tiene que ver con establecer y definir la relación entre activos, procesos y servicios. Encontrar esta relación ayuda a la organización a encontrar las dependencias críticas que son esenciales para determinar estrategias efectivas para proteger y sostener los activos.
- Por último la tercera actividad consiste en mantener este inventario y clasificarlo, durante el ciclo de vida de los activos en la organización. Esta actividad consiste entonces en identificar criterios de cambio y actualizar el inventario con los activos nuevos y con los activos que se dan de baja.

En conjunto estas tres actividades permiten determinar el ADN de cada activo: descripción, ubicación, propietario, custodio, requerimientos de resiliencia, procesos que apoya, servicios que apoya y valor relativo para la organización. Esta información se recopila en un formato denominado “Hoja de Vida del Activo”, que a manera de ejemplo se presenta a continuación en la figura 3.11.

Aunque la mayoría de los campos de la tabla son auto-explicativos, es conveniente mencionar el concepto de “contenedor” que aparece en el formato. Un contenedor es un activo capaz de contener a otro activo. La idea es que usualmente en una organización típica se presenta un anidamiento de activos, similar al de la figura 3.12.

Nótese en la figura cómo se tienen activos de tipo información que están contenidos en activos de tipo tecnología, que a su vez están contenidos en activos de instalaciones, al cual se acercan las personas a trabajar.

Los procedimientos de clasificación de activos no son sólo una buena práctica de industria, constituyen también la herramienta fundamental con la que hace gestión el área de seguridad informática y no sólo un informe que se tiene por ahí por si acaso hay una auditoría de cumplimiento de la SuperFinanciera. Sin contar con una clasificación de activos correcta y actualizada que refleje los requerimientos de seguridad de los activos de información, ¿cómo puedo saber qué debo proteger?

Estos “inventarios de activos” hacen parte de los insumos del “modelaje de adversarios” que se presentarán más adelante en el capítulo 5.

3.2 Gestión de Riesgos

El propósito de la Gestión de Riesgos es identificar, analizar, y mitigar riesgos operativos para los activos de la organización que puedan afectar de manera desfavorable la operación y entrega de los servicios.

Se entiende como análisis de riesgos un procedimiento sistemático y repetible que permite exhaustivamente identificar amenazas que enfrenta una organización y una vez identificadas, calificarlas en términos de probabilidad que suceda e impacto causado.

Este procedimiento de Análisis de Riesgos no es solamente una buena práctica, es también un requerimiento de PCI sección 12.1.2: *"(IS policy) Includes an annual process that identifies threats, vulnerabilities, and results in a formal risk assessment"*.

Este análisis de riesgos debe estar basado en una metodología formal que garantice un proceso repetible y exhaustivo, toda vez que todo aquello que se escape durante la identificación de amenazas, constituirá un riesgo no considerado, que en el caso de materializarse tomaría la organización por sorpresa.

Esta es precisamente la diferencia más grande que se tiene entre el Análisis de Riesgos y los procedimientos de auditoría tradicionales. La auditoría tradicional está fuertemente basada en la experiencia del auditor y es de esperar que el número de "hallazgos" de un auditor senior sea sustancialmente mayores y mejores que los "hallazgos" de un auditor junior. En el caso del Análisis de Riesgos, los resultados son siempre consistentes.

En este orden de ideas las organizaciones deberían institucionalizar un proceso de Análisis de Riesgos basado en alguna de las tres metodologías hoy en día existentes y aceptadas por la industria:

- CRAMM *(CCTA Risk Analysis and Management Method)* fue creada por el gobierno del Reino Unido, como un esquema basado en tres pasos de valoración de riesgos (http://www.cramm.com/). Posteriormente fue adoptada por Siemens y es ampliamente usada en Europa, tanto en instituciones militares (v.g. la OTAN), como en compañías comerciales como Unisys®, RAC® y el gobierno holandés.
- Octave *(Operationally Critical Threat, Asset, and Vulnerability Evaluation)* es una metodología de Análisis de Riesgos desarrollada por el Computer Emergency Response Team de Carnegie Mellon University para el Departamento de Defensa de los Estados Unidos[4]. Incluye un esquema de tres fases que dimensionan aspectos tanto organizacionales como tecnológicos del riesgo operacional de una organización y es ampliamente usado por el sector Defensa en USA y en varias entidades financieras, el Bank of America entre otros.
- *Magerit* es una metodología de análisis y gestión de riesgos de Sistemas de Información elaborada por el gobierno Español[5]. Tiene un fuerte sabor derivado de los procedimientos de la metodología Octave.

Para conseguir el propósito de este bloque constructor, es preciso realizar la siguiente lista de actividades:

- La primera actividad está relacionada con las tareas de identificación de amenazas. Siguiendo una de las metodologías recomendadas, la organización debe identificar de manera exhaustiva las amenazas de resiliencia a nivel de activo, a nivel de proceso y a nivel de servicio a que está expuesta su operación.
- La segunda actividad tiene que ver con enmarcar las amenazas identificadas respecto a resiliencia organizacional. Es decir, hay que establecer la relación entre amenazas – servicios, amenazas - procesos y amenazas – activos.

[4] http://www.cert.org/octave/

[5] https://www.ccn-cert.cni.es/publico/herramientas/pilar43/magerit/index.html

- La tercera actividad está relacionada con la evaluación de las amenazas identificadas y la calificación en términos de impacto y probabilidad de materialización de cada amenaza. El resultado de esta actividad es una lista priorizada por criticidad de los riesgos a atender.
- La cuarta actividad tiene que ver con las tareas de tratamiento de riesgos y la elaboración de los planes de mitigación de los riesgos identificados y sus tendencias en el tiempo.

Independiente de la metodología de Análisis de Riesgos seguida, lo importante es obtener un artefacto "Mapa de Riesgos" que como veremos más adelante es uno de los insumos más importantes para el modelaje de adversarios en el capítulo 5.

Un mapa de riesgos busca de manera exhaustiva categorizar las amenazas a las que está expuesto un activo, junto con una calificación en términos de probabilidad que suceda y daño causado. Grosso modo hay cuatro escenarios de amenazas relacionadas con actores humanos a través de la red, actores humanos con acceso físico, problemas técnicos y otro tipo de problemas. Estos cuatro escenarios se muestran a continuación en la figura 3.13, figura 3.14, figura 3.15 y figura 3.16.

Nótese como este artefacto Mapa de Riesgos logra reunir -en una misma notación- tanto amenazas relacionadas con seguridad de la información (figura 3.13), como amenazas relacionadas con seguridad física (figura 3.14) y amenazas relacionadas con continuidad (figura 3.15 y figura 3.16).

3.3 Continuidad del Servicio

El propósito de la Continuidad del Servicio es asegurar la continuidad de operaciones esenciales de los servicios y sus procesos de soporte, en el caso en que llegara a suceder un incidente, desastre o algún otro tipo de evento adverso.

Para conseguir este propósito es preciso realizar la siguiente lista de actividades:

- La primera actividad consiste en identificar y priorizar los servicios de alto valor para el cumplimiento de la misión de la organización, junto con sus dependencias internas y externas, los procesos sobre los que se apoyan y los registros vitales que hacen parte de estos servicios. Precisamente sobre estos servicios de alto valor es que se deben enfocar los esfuerzos de continuidad.
- La segunda actividad tiene que ver con la identificación y posterior desarrollo de los planes de continuidad de los servicios que se clasificaron como de alto valor para la organización.
- La tercera actividad tiene que ver con la validación y pruebas de los planes de continuidad, para verificar que son pertinentes a los servicios que buscan sostener y para verificar que estén operativos cuando se necesiten.
- La cuarta actividad está relacionada con el mantenimiento de los planes de continuidad y los procedimientos que debe seguir la organización para incluir nuevos riesgos y actualizar los planes de continuidad en la medida en que hayan cambios tecnológicos en los procesos de apoyo a los servicios críticos.
- Finalmente, la quinta actividad tiene que ver con la gestión de cambios a los planes de continuidad, la inclusión de nuevos planes (posiblemente para nuevos servicios) y el retiro de planes por fin de vida útil (posiblemente para servicios que se retiraron del mercado).

Sin embargo, desde el punto de vista del ciclo de vida del software, el artefacto más importante es el BIA *(Business Impact Analysis)* del proceso que soporta la tecnología que se

está adquiriendo[6]. En la metodología del DRI *(Disaster Recovery Institute)*, el BIA se obtiene en la etapa 2.

Veremos más adelante en el capítulo 5 cómo este artefacto BIA es uno de los insumos más importantes para la definición de requerimientos no funcionales asociados a continuidad.

El BIA define para cada proceso de negocio sus requerimientos de recuperación. Gráficamente estos requerimientos tienen la apariencia de la figura 3.17.

Nótese en la figura el transcurrir del tiempo de izquierda a derecha y los parámetros MTD *(Maximum Tolerable Downtime)*, RTO *(Recovery Time Objective)*, RPO *(Recovery Point Objective)* y WRT *(Work Recovery Time)* de un proceso de negocio hipotético.

El MTD corresponde al máximo tiempo de interrupción que el proceso de negocio puede tolerar. Numéricamente equivale a RTO+WRT.

Por otro lado, el RTO (Recovery Time Objective) indica el tiempo máximo aceptable para recuperar la tecnología de soporte al proceso una vez se ha interrumpido, mientras que el RPO indica la máxima cantidad de datos (medidos en unidades de tiempo) que el proceso de negocios puede tolerar perder. Finalmente, el WRT es el tiempo máximo disponible para recuperar la información del proceso, una vez la tecnología de soporte vuelve a estar operativa.

Desafortunadamente estos esfuerzos de resiliencia en muchas organizaciones se encuentran dispersos en silos organizacionales, en ocasiones inconexos: seguridad de la información, seguridad tecnológica / seguridad informática, continuidad y cumplimiento.

Dado que todas estas áreas tienen que ver con aspectos de gestión de riesgo operacional, el enfoque de resiliencia presentado en este documento unifica estos esfuerzos de manera coherente en términos de procesos de negocio, activos de soporte y amenazas.

En el siguiente capítulo se presenta la notación BPMN *(Business Process Management Notation)*, una herramienta supremamente poderosa y estándar de descripción de procesos de negocio, que constituye la «lingua franca» para definir procesos de fábrica de software.

[6] Adquirir Software en el sentido *CobiT* se trata tanto del desarrollo interno, como del desarrollo a la medida, como el software licenciado a un tercero

Hoja de Vida del Activo			
Nombre del Activo			
Fecha de Creación		# de Versión	
Participantes en la Identificación	Nombre		Cargo
	Fulano, Zutano, Mengano, Mengano, Perencejo y Perencejo		
Justificación de Selección			
Descripción	Este activo ...		
Tipo de Activo	Información / Tecnología / Persona / Instalación		
Etiquetado	Etiqueta de Confidencialidad	Etiqueta de Integridad	Etiqueta de Disponibilidad
Propietario	Cargo	Área	
	Fulano de Tal	Área Organizacional	
Custodio	Cargo	Área	
	Mengano de Tal	Área Organizacional	
Usuario	Cargo	Área	
	Fulano de Tal	Área Organizacional	
Procesos que Soporta	Procesos A, B, C		
Servicios que soporta	Servicio X, Y, Z		
Estado del Activo	En Uso / Dado de baja		
Requerimientos de Resiliencia			
Requerimiento	Nivel	Políticas	
Confidencialidad	Valor	*Política de seguridad asociada*	
Integridad	Valor	*Política de seguridad asociada*	
Disponibilidad	Valor	*Política de seguridad asociada*	
Otros	Valor	*Política de seguridad asociada*	

Contenedores más importantes del Activo			
Sistemas y Aplicaciones	Nombre	Actor	Propietario
✓ Aplicaciones de Negocio			
✓ Sistema Operativos			
✓ Bases de Datos			
✓ Otros sistemas y Aplicaciones			
Hardware	Nombre	Actor	Propietario
✓ Servidores			
✓ Redes LAN / WAN			
✓ Elementos activos de red			
✓ Dispositivos Electrónicos personales (Estaciones de trabajo, portátiles, teléfonos inteligentes).			
✓ Discos duros y cintas			
Personas	Nombre	Actor	Propietario
✓ Personal técnico			
✓ Otros Terceros			
Otros Contenedores	Nombre	Actor	Propietario
✓ Papeles y archivos físicos.			
✓ Puesto de trabajo, y otros ubicaciones.			

Figura 3.11: Hoja de Vida de los Activos

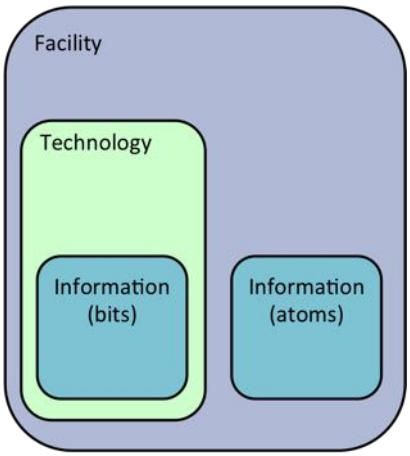

Figura 3.12: Contexto de los Activos

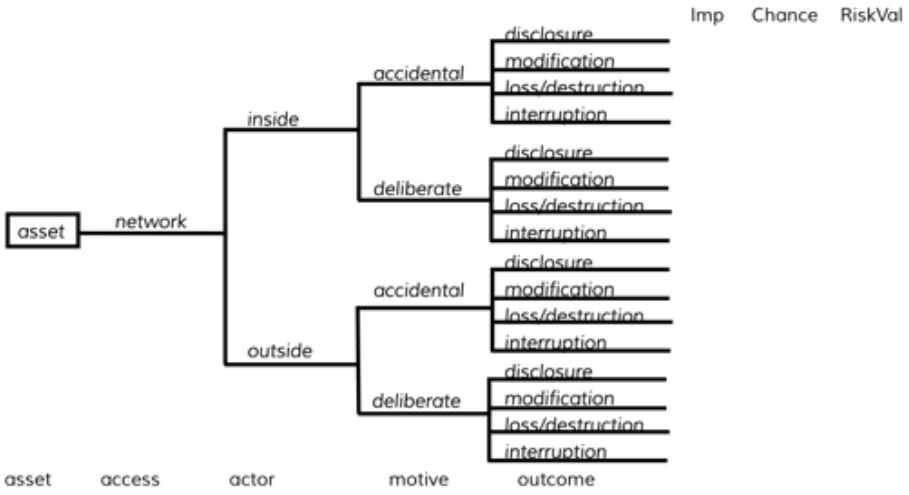

Figura 3.13: Riesgos causados por actores humanos a través de la red

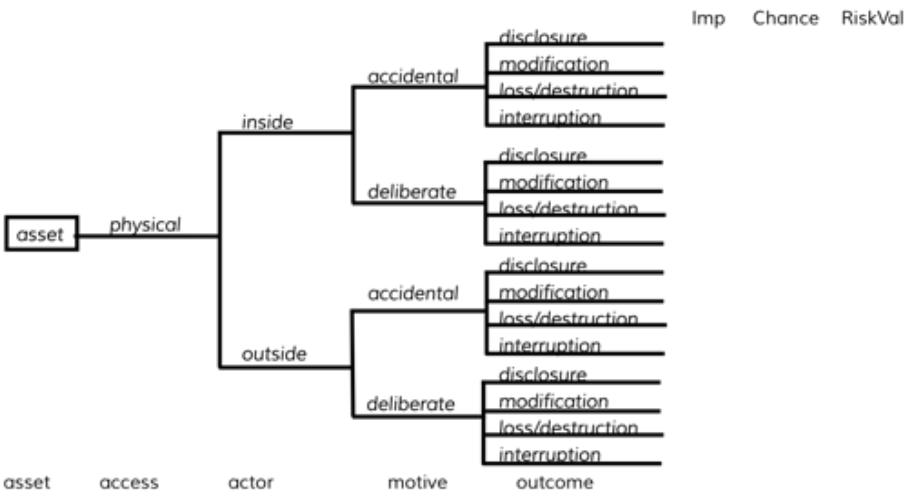

Figura 3.14: Riesgos causados por actores humanos con acceso físico

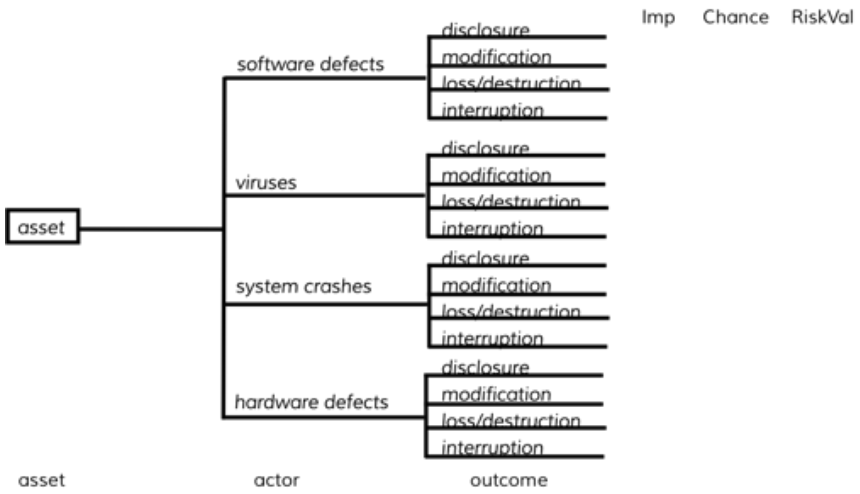

Figura 3.15: Riesgos causados por problemas técnicos

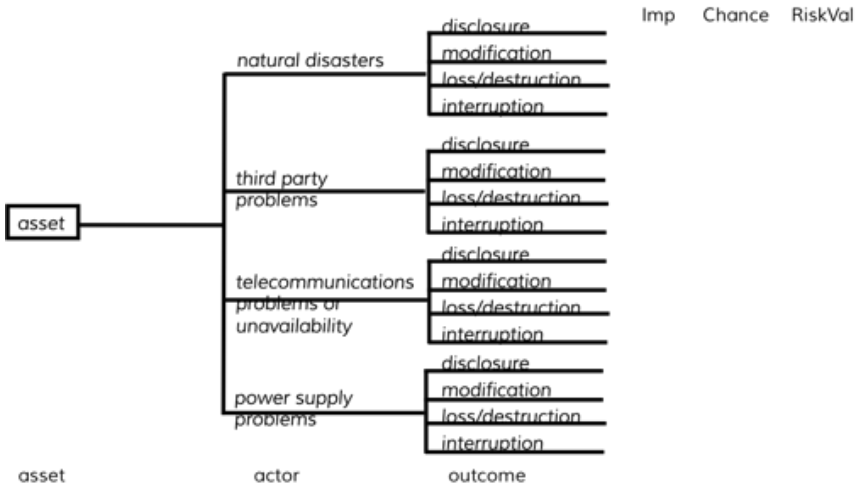

Figura 3.16: Riesgos causados por otros problemas

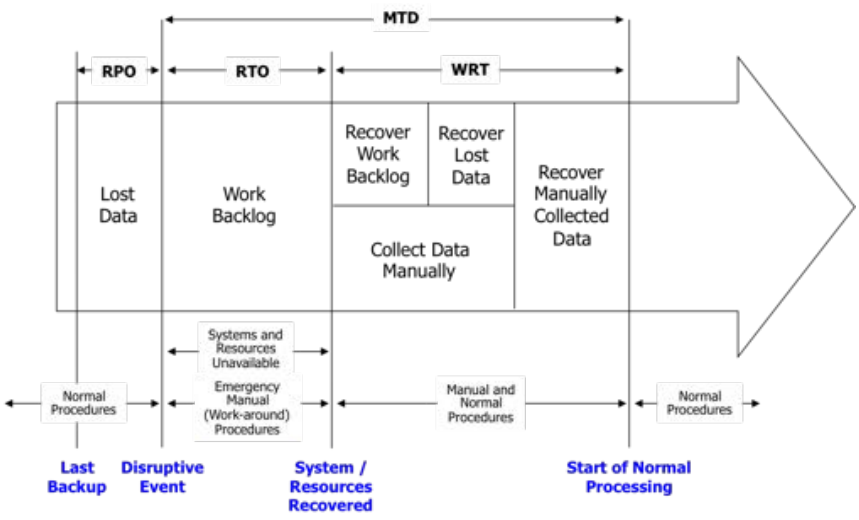

Figura 3.17: Parámetros del BIA de un proceso de negocio

4. Descripción de Procesos y BPMN

> Business acumen is quickly becoming the eleventh domain of information security. To adapt, security professionals must align with business management and develop depth and breadth within business
>
> *Gideon Rasmussen*

La gestión de procesos de negocio *(Business Process Management)* es una disciplina que busca la optimización de los procesos con los cuales una organización cumple sus objetivos. Para ello se busca poder entender el proceso de negocio a cabalidad, y una vez comprendido poder tomar decisiones de mejora que haga que el proceso se ejecute de manera más rápida y eficiente. La mejor manera de entender el proceso es poder representarlo de una forma visual, que sea fácil de entender para los responsables y participantes del proceso.

Para representar un proceso de negocio existen varias notaciones, algunas textuales, otras gráficas, y otras mixtas. Para buscar un consenso y una opción que permitiera tanto representación visual como la posibilidad de traducción a algún lenguaje de automatización de procesos haciendo uso de un motor de procesos, se propuso el estándar de Notación y Modelaje de Procesos de Negocio *(Business Process Model and Notation - BPMN)*. Haciendo uso de BPMN se puede diseñar y representar gráficamente cualquier proceso de negocio de una forma visual para que quien lea el diagrama pueda comprender los pasos a seguir para ejecutar el proceso y quién es el responsable de cada tarea.

A continuación se presenta una introducción a los elementos básicos de la notación BPMN. Se trata de un tutorial simple con los elementos más comunes y expresivos. Esta no pretende ser una guía extensiva y completa de todas los elementos definidos en el estándar, y se recomienda para quienes deseen tener un dominio mayor de BPMN que una vez se comprendan los elementos básicos de esta guía se haga una profundización del estándar BPMN 2.0[1] o [al12].

En el transcurso de este capítulo se detallarán los elementos más comunes y necesarios para poder crear y entender un diagrama BPMN. Para crear diagramas BPMN se pueden utilizar herramientas gráficas no especializadas como PowerPoint u editores gráficos, sin embargo existen herramientas especializadas como visio, draw.io, bizagi, entre otras.

[1] http://www.omg.org/spec/BPMN/2.0/

4.1 Actividades

En BPMN se considera actividad a cualquier tarea o subproceso que debe ser realizado para cumplir el objetivo del proceso. Se representan utilizando rectángulos de esquina curva como está detallado en la figura 4.1. Una tarea es una acción simple o puntual que debe ser realizada para que el proceso pueda ser llevado a cabo. Por ejemplo, si se desea preparar un jugo de naranja, debe existir una tarea que sea "Exprimir naranjas" y podrían existir otras como "Comprar naranjas" o "Servir jugo".

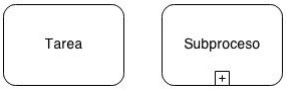

Figura 4.1: Actividades en BPMN

Ahora bien, si el proceso a modelar fuera el de preparar un desayuno, vemos que la actividad "Preparar jugo de naranja", la cual hace parte del proceso padre "Preparar desayuno", contiene varias tareas a su interior, por lo que se modelaría como un subproceso. La figura de subproceso cumple un papel de simplificación visual, pues permite resumir muchos elementos gráficos en uno solo, sin embargo se requerirá modelar dicho subproceso en su propio diagrama BPMN aparte.

En sí el definir si una actividad es una tarea o un subproceso depende del nivel de granularidad con el que se quiera modelar. Por ejemplo uno podría elegir modelar la tarea "Comprar naranjas" como un subproceso con las tareas "Ir a la tienda", "Elegir naranjas", "Pagar naranjas", "Volver a casa". Tal nivel de granularidad podría ser adecuado para la persona responsable de preparar el jugo, pero la persona que se va a tomar el jugo encontraría tal nivel de detalle engorroso. Es por ello que definir si una tarea vale la pena ser modelada como subproceso depende del público objetivo del modelo (¿necesita saber cómo se realiza la tarea?) y de la complejidad de la tarea (el como exprimir una naranja no debería necesitar de un nuevo modelo para ser explicado). Al final depende de la organización y del contexto el definir una actividad como tarea o subproceso.

4.2 Eventos

Los eventos en BPMN determinan algo que sucede que permite dar inicio a un proceso, y determinan el final del mismo. Gráficamente se representan con círculos como se muestra en la figura 4.2.

Figura 4.2: Eventos en BPMN

Los eventos pueden ser de tres tipos:

- Eventos de inicio: Se representan dejando el perímetro del círculo como una línea delgada e indican desde donde se inicia el flujo del proceso.
- Eventos de fin: El círculo tiene un perímetro con línea gruesa y una vez el flujo del proceso llega a este elemento el proceso se da por terminado.
- Eventos intermedios: Se representan con un doble perímetro y sirven para indicar que dada una condición cumplida el flujo del proceso se altera, o sucede una tarea específica.

Cada proceso modelado deberá tener al menos un evento de inicio, y deberá tener un evento de final. Los eventos intermedios son opcionales y se utilizan para representar situaciones específicas que pasan a mitad de proceso (por ejemplo una excepción o error). La notación de eventos se puede ampliar incluyendo una imagen dentro del círculo para detallar un tipo de evento específico, como se muestra a continuación en la figura 4.3.

Figura 4.3: Ejemplos de Eventos extendidos

Nótese en la figura ejemplos de eventos como:

- Un evento con un ícono de reloj interno denota que este evento sucede a una hora o fecha específica o con una determinada periodicidad.
- Un evento con un ícono de mensaje a su interior denota que el evento está recibiendo o arrojando un mensaje. Esto tiene que ver con algo que se verá en la siguientes secciones, y es la comunicación entre procesos. El ícono de mensaje es blanco cuando recibe, y negro cuando envía.
- Un evento de error tiene un icono específico y detalla la posibilidad de fallo durante el proceso. Por ejemplo, “se quemaron los huevos” , “se fue la luz”, etc.

Los anteriores tipos de evento son sólo unos de los más comunes, pero en la especificación de BPMN 2.0 se detallan más tipos, sin embargo son de uso escaso y no afectaría de manera sensible el no conocerlos para leer la mayoría de diagramas BPMN.

Algo importante que hay que entender de los eventos es que estos se consideran interruptores, es decir, cuando suceden obligan a desviar el flujo normal del proceso. Por ejemplo cuando sucede un problema (evento tipo “Error”) hay que hacer otras actividades para atenderlo, y el flujo normal del proceso no se resume hasta que estas actividades sean atendidas. Sin embargo existe la posibilidad de tener eventos que no interrumpen el flujo normal del proceso, es decir, inician un flujo paralelo de actividades sin interrumpir el flujo normal del proceso. Estos tipos de eventos se detallan haciendo el perímetro del círculo con una línea punteada.

4.3 Piscinas y Carriles

La piscina es un contenedor de los elementos de un proceso. Al igual que una piscina real, se pueden crear divisiones o carriles dentro de la misma para diferenciar a los participantes de un proceso. En el ejemplo del proceso “Preparar desayuno” podríamos tener una piscina que se llame “Preparación de desayuno” en la cual metemos los eventos de inicio y fin y las tareas definidas (hacer jugo de naranja, fritar huevos, preparar café, etc.).

Si quien prepara el desayuno es una sola persona, podríamos tener una piscina con un carril llamado “Cocinero” quien es el que ejecuta todas las tareas, pero si quisiéramos modelar por ejemplo que hay un cocinero que hace ciertas tareas y un asistente que hace otras, tendríamos 2 carriles (uno de cocinero, y uno de asistente). A continuación vemos en la figura 4.4 como se representan gráficamente estos elementos.

En el ejemplo del cocinero y el asistente, ¿Por qué no hacer dos piscinas, una para cada participante, en vez de una piscina con dos carriles? Esto sólo es posible cuando lo que queremos es hacer un diagrama de colaboración. En este tipo de diagramas BPMN se quiere mostrar es la relación entre varios procesos y participantes. Sin embargo lo que nos atañe en esta guía es

Figura 4.4: Piscina y Carriles

ver el modelaje diagramas de proceso, en los cuales se modela el flujo de un sólo proceso. Pues bien, bajo esta óptica cada piscina representa un proceso en sí, por lo que al hacer dos piscinas estaríamos dando a entender que son dos procesos diferentes que dependen entre sí para ser llevados a cabo. En el ejemplo el proceso es “preparar desayuno”, así que si modelamos a cada participante con su propia piscina estaríamos diciendo erróneamente que hay dos procesos de por medio. ¿Cuándo utilizaría varias piscinas? Cuando los límites de dos procesos que se comunican están demarcados claramente, es decir, cuando las acciones de un proceso interfieren con el otro, pero ningún participante tiene manera de saber cómo se desarrolla (ni influir en el desarrollo) del otro. Un ejemplo que podría aclarar el panorama es el siguiente: suponga que el cocinero y el asistente trabajan para un restaurante, y un cliente llegó a pedir un desayuno. El cliente no podría ser un carril dentro de la piscina “Restaurante” pues no hace parte de la organización, no sabe cómo se prepara el desayuno, ni toma activa participación en la creación del desayuno. El cliente es una piscina aparte que cumple tres tareas: “Ordenar desayuno”, “Consumir desayuno”, y “Pagar la cuenta”. Vemos que un proceso es el del restaurante al preparar el desayuno, y otro el del cliente al solicitar y consumir el desayuno.

4.4 Elementos de conexión (flujos y asociaciones)

Hasta este momento tenemos dos tipos de elementos para representar un proceso: actividades y eventos. Y sabemos que dichos elementos deben de ir dentro de un carril, que a su vez va dentro de una piscina. Sin embargo no tenemos forma de representar el flujo u orden en que las tareas serán realizadas. Para ello se utiliza una línea continua con una flecha al extremo la cual indica el flujo secuencial, como se muestra a continuación en la figura 4.5.

Figura 4.5: Elementos de Conexión

Es así como con estos elementos se podría representar un proceso completo de la manera en que se muestra en la figura 4.6.

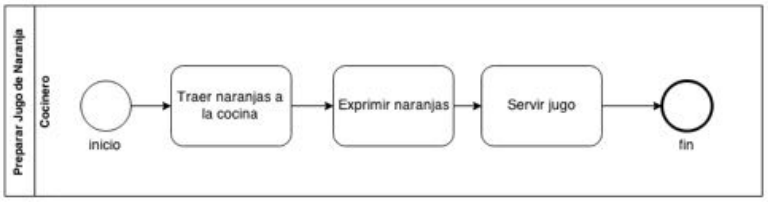

Figura 4.6: Ejemplo de diagrama de Proceso

En este caso la piscina contiene el proceso “Preparar jugo de naranja”, existe sólo un participante (carril único), y el flujo de eventos es el siguiente: el proceso inicia con la tarea “Traer naranjas a la cocina”, y una vez esta tarea es completada el cocinero exprime las naranjas, para luego servir el jugo, momento en que el proceso termina. Las tareas se ejecutan de manera secuencial, por ello el flujo se conoce como “flujo secuencial”. Vale resaltar que dicho flujo es unidireccional, es decir que no se puede crear una línea con una flecha en cada punta para permitir que el flujo se devuelva.

Existe una regla importante en el flujo secuencial, y es que este no puede salir de los límites de una piscina. Es así que si en el flujo de un proceso este tiene que comunicarse con otra piscina hay que utilizar el flujo de mensajes (línea punteada). Esto se utiliza para dar énfasis a que el flujo secuencial es para determinar el flujo dentro del proceso, y al ser dos piscinas diferentes dos procesos diferentes en sí mismo, la comunicación queda restringida al envío de mensajes entre los dos. Asimismo, un flujo de mensajes no se puede utilizar para comunicar dos elementos dentro de la misma piscina.

En el ejemplo del cliente que llegaba al restaurante a solicitar un desayuno, se modelarían dos piscinas, una para el cliente y una para el restaurante. El cliente enviaría un flujo de mensaje al restaurante solicitando el desayuno, el restaurante lo prepara y envía un flujo de mensaje al cliente entregando el desayuno, y finalmente el cliente consume el desayuno y envía un flujo de mensaje con el pago de la cuenta y termina el proceso general, como se muestra a continuación en la figura 4.7.

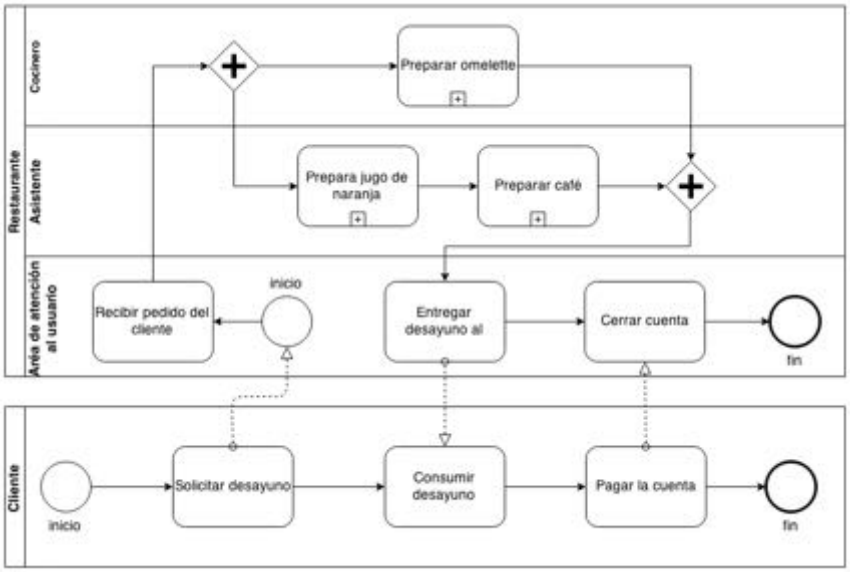

Figura 4.7: Ejemplo de comunicación entre piscinas

Al ver el anterior diagrama vemos la utilización de un rombo. Dicho elemento se conoce como “compuerta” y será explicados en la siguiente sección.

4.5 Compuertas

Las compuertas son elementos representados por rombos y cuya función es determinar los distintos posibles caminos por los cuales un flujo de eventos puede ir. Las compuertas se representan con rombos, como se muestra a continuación en la figura 4.8.

Las compuertas sirven para representar situaciones en las cuales el flujo principal se divide. Por ejemplo cuando se quiere determinar un punto del proceso en el cual dependiendo de una

Figura 4.8: Compuertas en BPMN

decisión se realizan dos tareas mutuamente excluyentes, se utiliza un rombo con una equis en su interior[2]. A este tipo de compuerta se le conoce como compuerta excluyente, y dada una condición determinada decide cual camino seguirá el flujo de eventos. Un ejemplo de cómo utilizar esta compuerta se muestra en la figura 4.9.

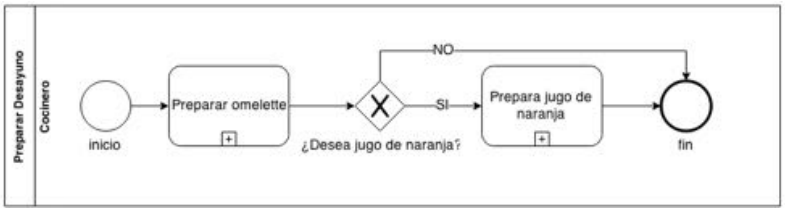

Figura 4.9: Ejemplo utilizando compuerta excluyente

En este caso la realización de una tarea (y el camino que sigue el flujo de eventos) depende de la condición "¿se desea jugo de naranja?". Dadas dos o más opciones el flujo sólo puede ir por uno de los caminos posibles.

Ahora supongamos que se desean realizar varias actividades en paralelo. Para ello se puede utilizar la compuerta de actividades paralela, la cual indica que el flujo de proceso se divide y a partir de la aparición de la compuerta las tareas se llevan a cabo al mismo tiempo. Para ejemplificar lo anterior supongamos el caso en el que el proceso principal es "Preparar desayuno" y contamos con un cocinero y un asistente. Queremos modelar que mientras el cocinero prepara un *omelette* el asistente prepara el jugo de naranja y el café. El correspondiente diagrama sería el de la figura 4.10.

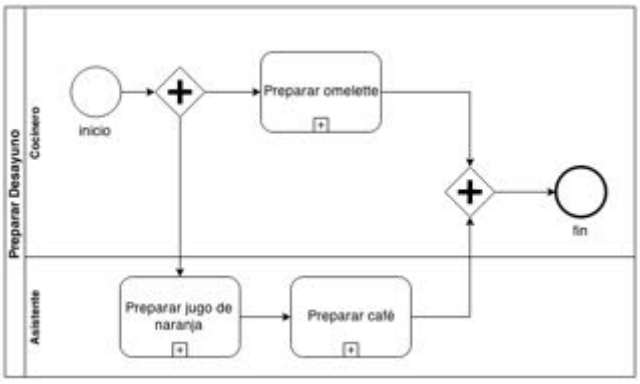

Figura 4.10: Ejemplo utilizando compuerta paralela

Nótese que en el ejemplo los dos flujos que se habían generado en la primera compuerta convergieron a otra compuerta igual. Sin embargo en el ejemplo de la compuerta exclusiva los flujos convergen directamente al evento de fin. ¿Cuándo convergen varios flujos en una compuerta, y cuando pueden converger a una actividad o un evento? Todo depende de la forma

[2]Vale hacer la anotación que el estándar BPMN permite representar la compuerta excluyente con el rombo vacío

en que se bifurcó el flujo de eventos: Si se crearon dos o más flujos excluyentes la convergencia se puede dar en una actividad o evento pues la naturaleza exclusiva indica que sólo uno de los flujos fue llevado a cabo. Por otro lado, si los flujos son paralelos y todos llegan a una misma actividad, dicha actividad (y todas las subsecuentes) serán ejecutadas por cada flujo paralelo existente, por lo que se haría necesario crear una compuerta de convergencia que espere a que cada flujo realice sus actividades para luego permitir el flujo normal del proceso.

La figura 4.11 ejemplifica dos casos en que se da la convergencia a partir de 2 flujos paralelos. En dicha figura se puede apreciar que cuando se utiliza la compuerta para unir los flujos, la tarea 3 no se ejecutará sino hasta que hayan sido ejecutadas las tareas 1 y 2. Cuando no se utiliza la compuerta de convergencia, la tarea 3 se ejecuta dos veces (una vez cuando la tarea 1 termina, y otra cuando la tarea 2 termina).

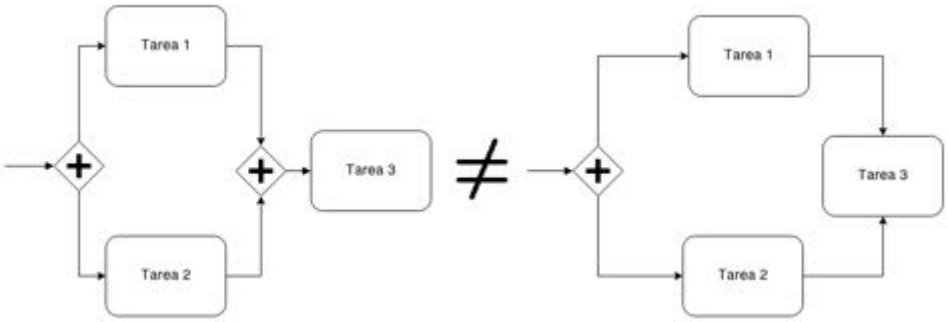

Figura 4.11: Diferencia en la convergencia de flujos

El estándar BPMN 2.0 define otros tipos extendidos de compuertas que no serán explicados en este documento pues no son de uso común en la mayoría de diagramas ni son vitales para poder entenderlos.

4.6 Artefactos y Notación Avanzada

Para terminar este tutorial vale la pena anotar que el estándar BPMN es más complejo y tiene variaciones de los símbolos antes vistos para permitir modelar situaciones en el proceso de negocio que tienen un nivel de sofisticación más avanzado. El alcance de este marco teórico no permite cubrir a cabalidad todo el estándar, además que para el usuario promedio no es necesario llegar hasta tal nivel de detalle para poder entender el flujo de un proceso.

Un tipo de elemento que valdría la pena mencionar serían los artefactos personalizados para modelar objetos que el estándar no cubre y que puede que sean específicos para la organización. Asi mismo se permiten hacer anotaciones a un objeto del diagrama de ser necesario. Para ello se utilizan líneas punteadas sin flecha unidas a una caja de texto. Un ejemplo de como hacer esto se muestra en la figura 4.12.

Haciendo uso de estas anotaciones y artefactos personalizados se puede extender el diagrama BPMN de un proceso para detallar los riesgos asociados a este. Por ejemplo en una tarea vital para la resiliencia de un proceso de negocios, se puede hacer una anotación mencionando que dicha tarea “depende de un equipo ubicado en una zona poco accesible” o “maneja información sensible” o “requiere por auditoría de separación de privilegios”. Integrando la notación BPMN, mediante anotaciones o comentarios en los flujos, al análisis de riesgos se podrían utilizar valores del BIA para señalar información de recuperación del proceso. Por ejemplo detallar el máximo tiempo de interrupción del proceso (MTD), señalando el tiempo máximo aceptable para recuperar la tecnología de soporte (RTO) y el tiempo para recuperar la información del proceso (WRT) en

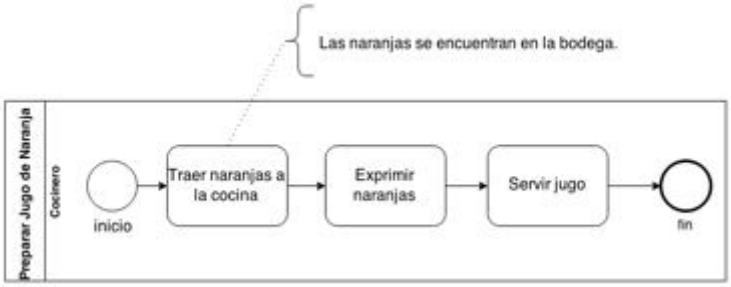

Figura 4.12: Anotaciones en BPMN

ciertas tareas.

Cabe destacar que el nivel de detalle y complejidad del diagrama varía de acuerdo al público objetivo del mismo, es decir, un diagrama muy general puede ser útil para presentaciones comerciales o a público general, pero no va a servir para detallar el proceso. Un diagrama muy técnico, complejo y especializado puede permitir a un experto del proceso detallar claramente todo, pero la mayoría de miembros de la organización tendrían dificultad para entender el diagrama y el proceso en sí.

Finalmente es importante resaltar las ventajas de la orientación de procesos en las fábricas modernas de software. Es claro que sin procesos maduros claramente definidos, con ciclos PDCA de mejoramiento constante, no es posible escribir software de calidad y menos software resiliente ([CMU10a], [CMU10b], [CMU10c]). Precisamente este tema “desarrollo moderno de software” es el tema principal que se abordará en el siguiente capítulo.

5. Desarrollo Moderno de Software

> If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology
>
> *Bruce Schneier*

En 1994 el reporte CHAOS realizado por el *Standish Group* encontró que de 8000 proyectos de software analizados sólo el 16% fue exitoso, en tanto los demás no fueron completados o lo hicieron después de sufrir serios problemas (no cumplir fechas de entregas, pasarse del presupuesto, software no completamente funcional). Diez años después se repitió el mismos estudio -esta vez con 40000 proyectos- y la tasa de proyectos exitosos había aumentado a 29%. Posteriormente, para el año 2010 se encontró que sólo el 37% de los proyectos eran exitosos (¡sólo uno de cada tres proyectos de software era terminado exitosamente!)[Int11].

Al buscar una causa que explicara el porqué de este panorama, se llegó a la conclusión que había que replantear la manera tradicional de desarrollo de software. El modelo tradicional conocido como cascada o *"waterfall"* se basaba en intentar avanzar una sola vez por cada una de las fases de desarrollo de forma secuencial y restrictiva. Es decir, en la primera etapa se definían los requerimientos o especificaciones del sistema (en su totalidad) y una vez se hiciera esto se cerraba esta fase y se procedía a la siguiente sin volver a mirar atrás. El problema grave que este esquema trae consigo es que no se permite retro-alimentación por parte de los diferentes actores durante el desarrollo del software a la planeación inicial realizada. De la misma forma que un plan de batalla tiene que ser re-formulado en el fragor de la contienda de forma inmediata para alcanzar los objetivos prioritarios, el desarrollo del software debe acomodarse rápidamente a los nuevos conflictos y problemáticas que van surgiendo durante su desarrollo.

5.1 Lean Software Development

Es por lo anterior que se planteó una nueva forma de pensar el desarrollo de software basado en las técnicas de manufactura "lean". Lean es una concepción de los procesos de manufactura ideado originalmente por Taiichi Ohno de Toyota, que ha sido acogido en mayor o menor medida por la industria del software. Los principios lean están basados en las siguientes ideas:

- Valor hacia el cliente

- Valor en toda la cadena de producción
- Flujo de la cadena de producción
- Hale
- Perfección

La primera idea "Valor hacia el cliente" busca que todo proceso de negocio existente en la organización se oriente a producir valor hacia el cliente. Es claro que para una organización lo más importante es el cliente y toda aquella actividad no estrechamente ligada con el cliente, puede considerarse como "desperdicio", un término que se definirá más adelante.

La segunda idea es "Valor en toda la cadena de producción" tiene que ver con buscar que las mejoras apliquen a toda la empresa y no solamente al departamento o sección que lo está implementando. El propósito de este principio es evitar que una micro-optimización afecte negativamente otra parte de la cadena de producción, al no ser tenida en cuenta las condiciones de otras partes de la cadena. Si esto llegara a ocurrir, no sólo se reduce el valor producido hacia el cliente, sino que todo el esfuerzo producido para lograrlo fue nuevamente "desperdicio".

La tercera idea "Flujo de la cadena de producción" busca conseguir que haya armonía entre cada actividad dentro de un proceso y su función dentro del todo. Es muy importante considerar el flujo de fabricación como un flujo armonioso que ni la cultura interna de la empresa o el proceso de desarrollo impida realizar.

La cuarta idea "Hale" se refiere al hecho que no se debe producir por producir. Toda actividad realizada dentro de la empresa debe haberse iniciado desde el final de la cadena de producción como una cadena que hala el cliente. Esto significa que sólo cuando el cliente lo necesite, se produce algo. Esto va desde procesos de desarrollo y producción hasta los procesos administrativos necesarios para lograrlo. Por lo tanto, todo lo que se haga anticipando al cliente es "desperdicio". Esto permite que no existan grandes inventarios acumulados ni esfuerzo anticipado que se puede aprovechar en otros aspectos importantes de la empresa.

La quinta idea "Perfección" consiste en la búsqueda de procesos sin errores. Naturalmente siendo un proceso una actividad humana, es normal que se presenten errores. Pero el tener la perfección como objetivo implica muchos cambios a nivel organizacional. Desde cómo se manejan errores cometidos a cómo estos se solucionan incluyendo analizar el ambiente cultural y técnico bajo los cuales ocurrió el problema. Esto garantiza que el error no se vuelva a cometer y cada mejora en este sentido es un paso hacia la perfección.

5.1.1 Terminología lean

Basados en las ideas lean anteriormente mencionadas que permiten analizar críticamente los procesos de la empresa, se introducirán a continuación algunos de los términos y conceptos claves de los principios lean. Estos son:

- *Just-In-Time*
- Autonomatización
- Desperdicio
- Señal
- Mejoramiento Incremental
- *Kanban*

El primer término "Just-In-Time" se refiere a que cada actividad sólo se ejecuta cuando se necesita. Esto puede referirse a actividades dentro de un proceso o a consumo de insumos o inventarios. Esto va de la mano con lo que mencionamos anteriormente como la filosofía del

"Hale". La única diferencia es que dentro de la literatura lean es común a que se refieran a esta filosofía como "Just-In-Time", "JIT" o "Pull" por su significado en inglés.

El segundo término "Autonomatización" o Jidoka en japonés, es la principal herramienta para lograr los valores lean que en principio parecen inalcanzables. Aunque suena como un error gramatical, es realmente el concepto de mezclar Automatización y Autonomía. Las cosas autónomas son aquellas que toman decisiones de forma instantánea e independiente, como la transmisión de carros 'Automáticos' que dada ciertas condiciones, decide en qué engranaje debe andar el carro sin pedir autorización de nadie. Por otro lado, la automatización es el hecho de realizar actividades en serie que antes eran realizadas por humanos y 'Automatizarlas', ya sea por medio de robots o aplicaciones que eliminan las tareas repetitivas.

Bajo la filosofía lean se dice: "Todo lo que puede autonomatizarse se autonomatiza"[Jav06]. Esto significa que cualquier esfuerzo humano en tareas repetitivas o de bajo valor es desperdicio, aún cuando es necesario prever que si la cadena de fabricación está introduciendo defectos, el humano debe poder detenerla inmediatamente (y autónomamente) para realizar correcciones inmediatas. Es decir, siempre que se pueda, todas las tareas y procesos dentro de la empresa deben implementarse autonomatizadas pues, además de ahorrar costos, evitan defectos que a su vez, son otra forma de desperdicio.

El tercer término "Desperdicio" o Muda en japonés, es considerado el crimen más grande que puede existir dentro de una empresa o comunidad según las metodologías lean. El desperdicio no es solo el mal uso de inventarios o insumos. El desperdicio incluye todo lo que no debe hacerse, ya sea porque está mal, o puede hacerse mejor, o puede ser autonomatizado. Por esta razón, es indispensable que toda la empresa piense de forma lean y empiece a optimizarse como un todo. Esto implica que cada problema o defecto es un desperdicio y como tal, se debe tratar de resolver de forma permanente, así implique cambiar todo proceso de negocio, incluyendo los que estén certificados. Si no se hace así, el problema seguirá ocurriendo y será un defecto, o sea, un desperdicio permanente y constante que le costará dinero a la empresa.

El cuarto término "Señal" o Andon en japonés, es el hecho de tener a todos los *stakeholders* informados del proceso o procesos que realiza la empresa. En empresas que no siguen estas metodologías, existe el riesgo de información oculta o controlada. Esto implica desperdicios en la comunicación entre stakeholders y tiempo desperdiciado en el traspaso de información de un grupo o departamento al otro. En este sentido, las metodologías Lean implican que se deben adoptar mecanismos de señalamiento del estado de todos los procesos dentro de la empresa. De esta manera, todo el que necesita saber en qué estado está un grupo o departamento se entera automáticamente sin interrumpir al grupo que está trabajando en ello. Esto puede implementarse de varias maneras y este documento hace algunas sugerencias al respecto. Pero es importante que cada departamento adopte la mejor estrategia para lograr un señalamiento de tal manera que se reduzcan los tiempos de traspaso de información, haya mayor transparencia del proceso y eviten interrupciones por preguntar cosas obvias.

El quinto término "Mejoramiento Incremental" o Kaizen en japonés, es uno de los términos más usados en las empresas lean. Como su nombre lo indica, significa el hecho de tener procesos que mejoran con el tiempo. Esto viene de la idea, de que los defectos no son culpa del producto sino de su entorno. En este caso, los defectos en software no es culpa de malos programadores sino de un ambiente que permite fácilmente introducir defectos. Esto implica que cada defecto que se encuentre en un proceso, debe ser atendido y arreglado de manera permanente e implementarse controles (autonomatizados en lo posible) que impidan que vuelva a ocurrir dicho defecto y no afecte el flujo de valor hacia el cliente.

El último término "kanban", es una forma de señalamiento introducida por primera vez en las fábricas de Toyota. Literalmente traduce "banderillas"y su objetivo es organizar cada paso de una cadena de producción de manera que cada tarea tiene un límite de entradas y salidas.

Esto garantiza que la tarea es atómica y permite ver con facilidad, qué estaciones de trabajo están represando el flujo de toda la cadena productiva. Aunque existen varias implementaciones dependiendo de la industria a la que se aplica, más adelante en este documento se hacen recomendaciones de las mejores prácticas dirigidas hacia el desarrollo de software.

5.1.2 Objetivos lean

Ya con la terminología base definida, se pueden definir los objetivos que se deben cumplir para lograr implementar con éxito la metodología lean. Estos objetivos son:

- Eliminar Desperdicios
- Construir Calidad hacia adentro
- Crear Conocimiento
- Procrastinar decisiones
- Entregas Rápidas
- Respetar Personas
- Optimizar el Todo

El objetivo "Eliminar Desperdicios" inicialmente parece obvio, pero los desperdicios en muchos casos no son evidentes. En ocasiones se ignoran de forma inconsciente debido a que no se tenía una noción clara de qué realmente significa un desperdicio. Gracias a lean, se definen categorías generales que permiten identificar mejor los ámbitos donde se pueden esconder desperdicios que pueden estar costando a la empresa dinero y valor hacia el cliente. Estas categorías pueden resumirse por sus siglas en inglés como "DOTWIMP" que se describen a continuación:

- *Defects*
- *Overproduction*
- *Transportation*
- *Waiting*
- *Inventory*
- *Motion*
- *Process*

Dado que es evidente que algunas de estas categorías no aplican al desarrollo de software, se ha creado su equivalencia para el desarrollo de software [Cur09] como se explica a continuación:

- *Defects* → Defectos en software: es importante reducir todos los defectos de software, integración, seguridad, etc.
- *Overproduction* → Funcionalidades Adicionales: es conocida la regla de Pareto que afirma que el 80% del valor de una aplicación viene del 20% de las funcionalidades ofrecidas. Esto significa que se debe velar por tener la posibilidad de que solo sea desarrollado ese 20% de la aplicación dado que el resto puede ser desperdicio.
- *Transportation* → Entrega de documentación o información: En los modelos clásicos de desarrollo como la "cascada", los requerimientos son recogidos por unos ingenieros o personal de ventas y este se los pasaba a los arquitectos de software y estos pasaban el diseño de la aplicación hacia los desarrolladores y así sucesivamente, con varios pasos burocráticos en el medio. El paso de información vital entre un equipo al otro es un desperdicio que se puede evitar si todos los interesados se encuentran presentes en la toma de decisiones.
- *Waiting* → Demoras en decisiones: Este defecto es mejor visto en desarrollo de software

cuando un requerimiento no es claro y es necesario consultar un supervisor o al cliente directamente. Dado que es posible que las otras personas no estén todo el tiempo disponibles para responder preguntas de desarrollo, estos tiempos de espera son considerados desperdicios.

- *Inventory* → Funcionalidades parcialmente completas: Es típico que durante el desarrollo se cambien las prioridades y se dejen funcionalidades parcialmente incompletas y que por el entorno cambiante, esta funcionalidad nunca se desarrolle. Esto es un desperdicio de esfuerzo y debe ser minimizado. Este documento más adelante describe las mejores prácticas para reducir este desperdicio.
- *Motion* → Cambios de tareas o interrupciones: Cada vez que una persona cambia de tarea, debe contextualizarse y esto puede implicar hacer preguntas a otros u orientarse en otro proceso de negocio totalmente distinto. Ese tiempo es considerado desperdicio y es crónico en personas que se creen con capacidades multi-tarea.
- *Process* → Procesos innecesarios: Estos son los procesos que son completamente desperdicio. Van desde documentación o burocracia innecesaria hasta procesos manuales que deberían ser autonomatizados.

Por otro lado, el segundo objetivo "Construir Calidad hacia adentro", significa que la calidad del producto a ofrecer está en el proceso y no en el producto entregado. En lean, se cree que los defectos son introducidos por procesos que los permiten. Por esta razón, cada mejora necesaria para lograr mejor calidad en los productos o servicios ofrecidos, deben implementarse a nivel del proceso que lo realiza. Por esta razón se conoce como "calidad hacia adentro".

El tercer objetivo "Procrastinar decisiones" no se trata de promover la irresponsabilidad, sino por el contrario representa la mejor estrategia a la hora de tomar decisiones. Las empresas que siguen las técnicas lean no deben tomar decisiones de afán. Por el contrario, deben tener la mayor cantidad de información posible antes de tomar decisiones. En este sentido, la experiencia muestra que entre más tiempo transcurra, más información se va a obtener. Esto en desarrollo de software puede representarse en decidir desarrollar bajo una tecnología u otra. Las personas encargadas de las decisiones, deben tomarse la mayor cantidad de tiempo considerada responsable para obtener información empírica, de modo que cuando sea la fecha límite, hayan podido tomar la mejor decisión posible, evitando desperdicios por malas decisiones.

El cuarto objetivo "Entregas Rápidas", es otro de los objetivos importantes dentro de las técnicas lean[Els08]. Este concepto se refiere a que los cambios grandes generalmente implican desperdicios grandes. Por lo tanto, una de las tantas mejores prácticas incluidas en las técnicas lean es que todo cambio sea pequeño e incremental. De esta manera, se pueden detectar defectos de forma oportuna y antes de que se conviertan en problemas grandes. Esta técnica es avalada por las metodologías ágiles de desarrollo de software de las cuales este documento menciona más adelante.

El quinto objetivo "Respetar Personas" de las técnicas lean se basa en apoderar a las personas para que puedan cumplir sus metas dentro de la organización. Para que esto se pueda dar, debe existir una cultura de respeto y confianza que permita cometer errores de forma pública sin temor a represión o despidos. Se debe confiar que la persona a la que se contrató tiene las capacidades necesarias para solucionar problemas sin ser sometida a la sobreprotección (*micromanagement*) de sus superiores.

El último objetivo "Optimizar el Todo", es el concepto de que los cambios se hagan a nivel global. Es fácil caer en la tentación de organizar una empresa como grupos ultra-aislados que a pesar de que hacen parte de la misma cadena de valor, sólo comparten entregables pero nunca se comunican. Esto puede ocasionar que los esfuerzos de un equipo vayan en contravía de otro,

en especial si hacen parte de una cadena productiva. Por tal razón, no solo es suficiente que los grupos o departamentos de trabajo puedan comunicarse entre sí, sino que también sean tomados en cuenta en el momento de realizar cambios en los procesos de negocio[Bai12].

5.2 Scrum

El capítulo anterior describe los principios básicos de las mejores prácticas lean. Es importante notar sin embargo que lean es una filosofía organizacional que permite enfocar esfuerzos de toda la empresa para eliminar desperdicios pero no describe detalladamente cómo se debe implementar.

Existen varias propuestas que buscan incorporar las ideas lean al desarrollo de software. Scrum es una de estas metodologías comprobadas que funcionan y optimizan no sólo los procesos de desarrollo de software sino cualquier actividad humana dentro de una organización. Scrum se enfoca en la productividad por equipos y no por individuo. Esto, aunque suene utópico, permite mayor transparencia y cooperación inter-disciplinaria pues elimina las recompensas individuales y esto a su vez elimina pretensiones egoístas de las personas que promueven la burocracia como objetivo personal y los abusos de poder en las organizaciones.

Scrum es inspirado en las técnicas Lean[Sut14]. Define tanto actividades como actitudes que deben tomar los equipos que lo implementan. Así las cosas, también permite la flexibilidad e independencia que los miembros del equipo necesitan para lograr sus objetivos, pero logra organizarlos de manera que exista una comunicación amplia entre todos los stakeholders del proyecto. Esta flexibilidad también permite que en ciertas ocasiones se pueden implementar otras metodologías complementarias que promuevan más valores Lean o que permitan cumplir regulaciones de ley.

5.2.1 Filosofía Scrum

Ya que conocemos el origen de Scrum, empezaremos a describir sus filosofías principales y derivadas que ayudan a cumplir los objetivos Lean. Algunas pueden sonar redundantes pero realmente son más aplicadas al contexto de trabajo en grupo. En resumen, estas filosofías son:

- Grupos Pequeños
- Entregables pequeños, iteraciones pequeñas
- *Single-Piece-Flow*
- Autonomía, trascendencia y multi-funcionalidad
- Planeación Razonable
- El Manifiesto Ágil

La primera de estas filosofías es “Grupos Pequeños”. Scrum enfatiza desde el principio que los grupos de trabajo no deberían ser grandes. Jeff Bezos, CEO de Amazon hizo famosa “la regla de 2 pizzas” que indica que los grupos deben ser no mayor a la cantidad de personas que se pueden alimentar con 2 pizzas (aproximadamente 9 personas)[1]. De la misma manera, Jeff Sutherland, uno de los creadores de Scrum, dice en uno de sus libros que el tamaño máximo de los grupos debe ser de 7 personas, aunque dependiendo del caso, se le puede sumar o restar 2 personas. Esto es debido a que la comunicación del grupo después de 7 personas se vuelve complejo, lo que implicaría imponer esfuerzo y herramientas de facilidad de comunicación, lo cual resulta ineficiente y por lo tanto un desperdicio.

Por otro lado, de la misma manera en que lean propone tener entregas rápidas, Scrum propone que toda entrega intermedia deba ser un producto funcional que produzca valor al cliente. Esto

[1] http://www.businessinsider.com/jeff-bezos-two-pizza-rule-for-productive-meetings-2013-10

es debido a que el cliente (en especial en el desarrollo de software) no siempre tiene claro qué es lo que quiere o realmente necesita, pero si una idea inicial. El manejar entregas parciales como productos terminados o demostraciones funcionales (demo) ofrece bastantes mejoras. Un ejemplo es que los desarrollos tengan un alto nivel de calidad y control contra defectos desde el principio. Pero más importante, que el cliente pueda cambiar desde temprano el rumbo del proyecto, evitando desperdicios en forma de funcionalidades que creía que necesitaba pero en realidad nunca iba a utilizar.

La tercera filosofía Scrum es *"Single-Piece-Flow"*. Aunque es un término que se origina del modelo original Toyota, SPF consiste en conseguir que cada desarrollador debe hacer solo una tarea a la vez. Esto es para evitar desperdicios en forma de interrupciones y contextualización. La idea proviene del problema en manufactura cuando se tiene un solo departamento encargado de crear varias partes distintas. En este contexto, existe un desperdicio grande en tiempo cuando los operarios terminan una parte y deben pasar a desarrollar otra distinta, pues esto implica cambiar de herramientas o alistar maquinaria distinta. Bajo esta idea, los desarrolladores de software también pierden tiempo cuando se está concentrado y cambian de contexto dada la complejidad de los problemas normales en desarrollo de software. Adicionalmente, estos cambios pueden implicar otros desperdicios de tiempo como cargar aplicaciones y herramientas distintas y pesadas, cambiar de computador presencial y/o remotamente o iniciar máquinas virtuales.

La cuarta idea Scrum es "autonomía, trascendencia y multi-funcionalidad". Esto significa que además de ser pequeños los equipos de trabajo, Scrum recomienda a que los equipos tengan habilidades diversas y no necesariamente especialistas en un tema específico. Esto implica que los equipos deben estar en la capacidad de desarrollar todos los aspectos de forma independiente y no depender de otras áreas o personas que no hacen parte de él. Adicionalmente, todos los miembros del equipo tienen la misma importancia y nivel de cargo en la empresa, evitando conflictos burocráticos. Esto también implica que deben contar con el poder o las facilidades dentro de la organización de cambiar ciertos aspectos de su trabajo que puedan llegar a estar impidiendo el desarrollo de sus actividades.

También se debe dejar claro que por esta misma razón, no se puede culpar individuos del equipo cuando hay problemas. En Scrum, todo el equipo es responsable de las acciones de sus integrantes. Por lo tanto, el equipo completo debe responder por tanto los éxitos, como los errores cometidos y las lecciones a aprender. Esto también implica que el equipo debe creer en el proyecto y estar altamente motivado, no sólo para desarrollar sus actividades, sino también para proponer ideas y mejoras tanto al producto como al proceso de desarrollo. A esto se refiere Scrum con la trascendencia del proyecto pues para lograr los niveles necesarios de motivación, se necesita que los miembros del equipo se sientan haciendo un proyecto trascendental no sólo para la empresa sino para sus vidas.

La quinta idea Scrum "Planeación Razonable" proviene de un problema común en cualquier tipo de proyecto, que las proyecciones hechas en planeación no concuerden con la realidad una vez empieza. Esto es porque es imposible predecir todos los problemas que un proyecto se puede encontrar en el camino. Por esta razón, Scrum no planea de antemano todas las actividades necesarias para realizar el proyecto. Solo se planea lo que se está haciendo en ese instante. Debido a esto es imposible saber desde el inicio cuándo va el proyecto a terminar. Pero a medida que se desarrolla, se van sacando métricas más precisas y alineadas con la realidad, como la velocidad del equipo para las tareas restantes. De este modo, después de unos periodos de tiempo se pueden empezar a ver fechas predictivas de terminación del proyecto y a medida que avanza, estas predicciones se van volviendo más allegadas con la realidad y por lo tanto más útiles que si se intentara adivinar desde un principio.

La última idea Scrum acá mencionada es conocida como "El Manifiesto Ágil". Scrum tiene muchas filosofías y principios que no son fáciles de recordar, en especial aquellas compartidas

con las ideas lean. Por este motivo se escribió lo que se conoce como el "Manifiesto Ágil" que resume los principios más importantes a seguir. Son cuatro frases que indican qué debe ser lo más importante en la toma de decisiones en Scrum:

- Los individuos e interacciones priman sobre procesos y herramientas.
- El software funcionando prima sobre la documentación extensiva.
- La colaboración con el cliente prima sobre la negociación de contratos.
- La respuesta al cambio prima sobre las planeaciones previas.

En la literatura es posible encontrar muchas más recomendaciones y abreviaciones filosóficas con respecto a Scrum. Pero las que recomendamos acá son las que consideramos más importantes y aplicables a nivel de desarrollo de software sin requerir mayor conocimiento inicial de procesos ágiles.

5.2.2 Implementando Scrum

Implementar Scrum implica implementar ciertos artefactos, procesos y actividades dentro del equipo. A continuación se hará una breve descripción de aspectos básicos de Scrum. Se debe tener en cuenta que si el equipo lo decide, están en la libertad de agregar o quitar componentes si es necesario y evaluar si aumenta la productividad y/o el valor hacia el cliente.

Actividades Scrum

Scrum se diseñó para ser lo más liviano posible en reuniones y mecanismos de compartir información. Por lo tanto, existen como base cinco tipos de reuniones que se describen a continuación:

- Planeación del *Backlog*
- Planeación del *Sprint*
- Scrum Diario
- Revisión del *Sprint*
- Retrospectiva del *Sprint*

La primera reunión "Planeación del Backlog" se hace solo al principio del proyecto. Aquí se reúnen todos los stakeholders del proyecto y se ponen de acuerdo en las "historias de usuario". Las historias de usuario son los requerimientos de la aplicación a desarrollar desde el punto de vista del usuario final y escritas en prosa como una historia corta. Se recomienda escribirlas en formato "Siendo un X, quiero hacer un Y, para lograr Z". A estas historias se les debe asignar una prioridad pero es opcional asignarles un grado de dificultad puesto que al principio del proyecto es difícil prever qué inconvenientes ocurrirán durante la implementación de cada historia de usuario.

Una vez definido este documento conocido como "Backlog", se define el tiempo de los ciclos de desarrollos conocidos como "Sprints". Estos Sprints definen cada cuanto debe haber un pedazo de la aplicación funcionando y utilizable por el cliente o "Demo". Se recomienda que sean entre 2 y 4 semanas aunque depende más del tipo de proyecto que se va a realizar.

La segunda reunión "Planeación del Sprint" se define antes del comienzo de cada Sprint. Acá el equipo Scrum decide qué "historias de usuario" del backlog se van a implementar y deben quedar listas al final del Sprint. Una vez escogidas, se decantan las actividades necesarias para implementar cada historia de usuario escogida. Estas actividades pueden ser en lenguaje técnico pero deben ser claras para cualquiera que vea el documento. En los mejores casos, se les pueden asignar a las tareas un número predictivo de horas que se demoraría desarrollando la actividad y

un nivel de dificultad.

La tercera reunión “Scrum Diario” se hace cada día. El equipo Scrum se debe reunir por quince minutos para responder tres preguntas:

1. ¿Qué se terminó a hoy?
2. ¿Qué se va a hacer para mañana?
3. ¿Qué obstáculos nos han impedido nuestro trabajo?

Estas preguntas a diario ayudan al equipo tener control y estar formalmente informados de la situación del grupo sin necesitar papeles o controles burocráticos.

La cuarta reunión “Revisión del Sprint” se hace al final del Sprint y se revisa el Demo terminado. Esta reunión ocurre con todos los stakeholders de manera que se puede obtener retroalimentación valiosa antes de continuar con el proyecto.

La quinta reunión “Retrospectiva del Sprint” ocurre también al final de cada Sprint y puede preceder a la “Revisión del Sprint”. En esta reunión sólo es necesario que esté el equipo Scrum. En ella se revisa la retroalimentación de los stakeholders y las lecciones aprendidas durante el Sprint que transcurrió. De esta manera, se utiliza esta información para volver a priorizar, añadir o quitar las historias de usuario. Como los cambios de historias de usuarios ocurren tempranamente en el proyecto, se minimiza el esfuerzo desperdiciado.

Artefactos Scrum

Dado que Scrum se define como una metodología liviana, esta define muy pocos artefactos de administración. Claro está que a veces es necesario incluir mecanismos de control para mantener el gobierno del proyecto lo más transparente y rastreable posible.

A continuación se describirán los cinco artefactos considerados más importantes en la práctica de Scrum. De todas maneras, se puede dar el caso en que el equipo necesite más o menos documentación dependiendo de la necesidad de registro y comunicación que el equipo, certificación o la ley requiera.

- Backlog del Producto
- Backlog del Sprint
- Burndown Chart
- Scrum Board o *Kanban*
- BPMN del proceso

El “Backlog del Producto” contiene todas las historias de usuario del proyecto. Puede tener distintas versiones a medida que el proyecto cambia. Existen muchas formas de realizarlo dependiendo del proyecto pero puede bastar con una hoja de cálculo accesible a todos.

El “Backlog del Sprint” contiene la lista de qué historias de usuario se van a implementar en el sprint descrito y las actividades necesarias para lograrlas. Esta lista puede contener más información técnica que la historia de usuario dependiendo del nivel de control o granularidad de la información a ser comprometida a desarrollarse. Al igual que el Backlog del producto, este puede implementarse de varias maneras o con una hoja de cálculo sencilla.

Si se optó por predecir horas en las tareas del Backlog del Sprint y también de registrar las horas que realmente se demoró el equipo en desarrollar cada actividad, entonces es posible construir una gráfica de Tiempo restante en horas para terminar el proyecto sobre el tiempo transcurrido. Esto es conocido en la literatura como *“Burndown Chart”* y tiene la apariencia de la figura 5.1.

Nótese en la figura la pendiente de la “burndown chart”, la cual muestra la velocidad del equipo. Debe ser claro que cualquier cálculo de pendiente solo es posible una vez exista información de algunos días.

Figura 5.1: Ejemplo Burndown chart

Siguiendo con la filosofía lean de Señalamiento, el "Scrum Board" o "Kanban" como también es conocido, es una tabla que indica públicamente el flujo y estado de las actividades definidas en el Sprint. Esto simplifica la gestión del desarrollo pues se puede identificar instantáneamente quienes o donde se están estancando tareas necesarias para cumplir las historias de usuario. Este artefacto puede implementarse de muchas maneras y depende del proyecto y del equipo la complejidad necesaria para publicar la información necesaria. Por ejemplo, un esquema muy simple puede ser el mostrado en la figura 5.2.

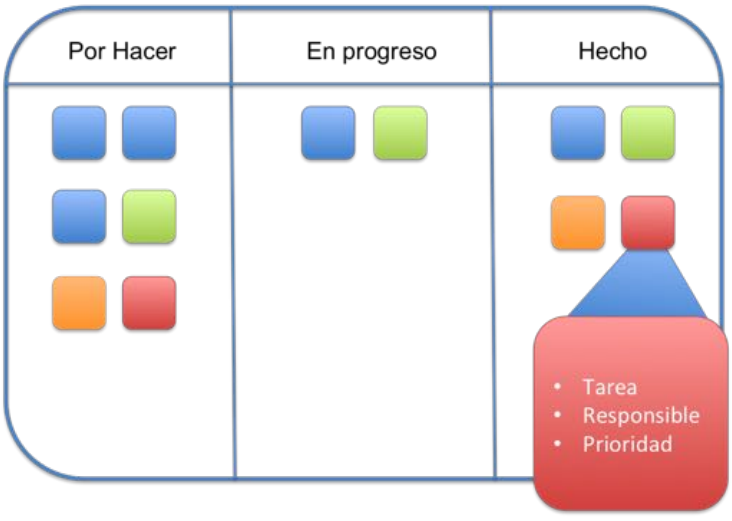

Figura 5.2: Scrum-board simple

O un ejemplo más complejo[2] se muestra en la figura 5.3.

Por último, si el equipo es grande o cambia con frecuencia, es fácil ver la necesidad de formalizar el proceso de desarrollo para guiar y contextualizar a los miembros del equipo. La mejor manera de lograr esto es definir claramente los procesos, usando alguna notación estándar

[2]tomado de https://www.flickr.com/photos/plutor/5260265039

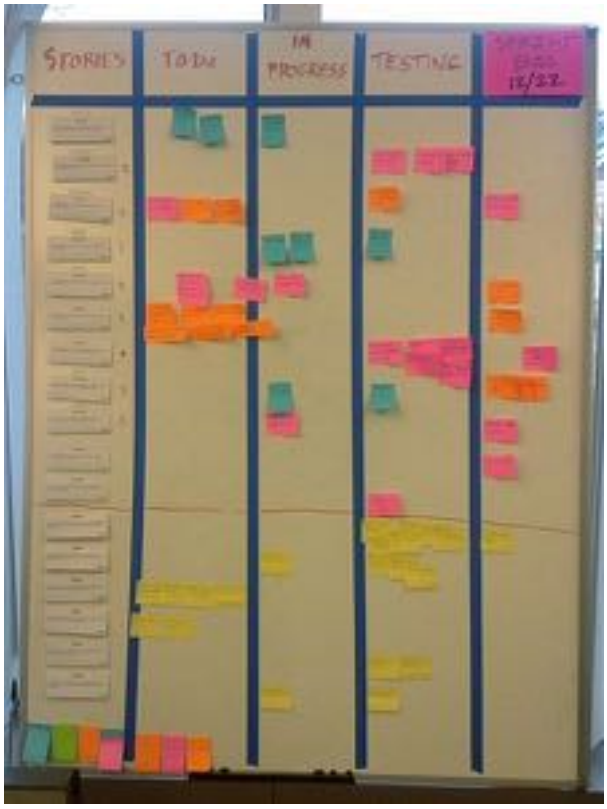

Figura 5.3: Scrum-board más complejo

como por ejemplo BPMN. El contar con una representación gráfica del proceso puede convertirse en una herramienta visual muy poderosa de identificación de problemas, cuellos de botella y posibilidades de mejora en los procesos de adquisición de tecnología de la organización.

Proceso Scrum

Ya con los conceptos introducidos en los capítulos anteriores, a continuación se muestra el flujo de actividades que típicamente compone un desarrollo Scrum. Gráficamente usando la notación BPMN el proceso tiene la apariencia de la figura 5.4.

Nótese en la figura la naturaleza iterativa de Scrum que permite que los proyectos se realicen a tiempo y permiten el cambio de forma temprana en el desarrollo. Este diagrama muestra las actividades que deben ocurrir para la ejecución de cada Sprint. Es importante tener en cuenta que el diagrama no muestra todas las actividades del proceso Scrum. Esto es debido a que aunque hacen parte del proceso Scrum, no generan documentación o ocurren durante la ejecución del Sprint mostrado acá como el subproceso llamado *"Sprint Execution"*.

También es importante ver que las actividades son definidas de forma genérica ya que es el equipo el que debe definir la forma más eficiente de desarrollarlas. Gracias a esta flexibilidad, es que Scrum permite que se incluyan otras metodologías según sea el caso, ya sea para mejorar el proceso o para incluir procesos normativos o de certificación. Esto también indica que el equipo es responsable de definir cuánto debe durar cada sprint y cómo se deben ejecutar. En el próximo capítulo, se describe una metodología para hacer más eficiente la ejecución se Sprints en proyectos de desarrollo de software.

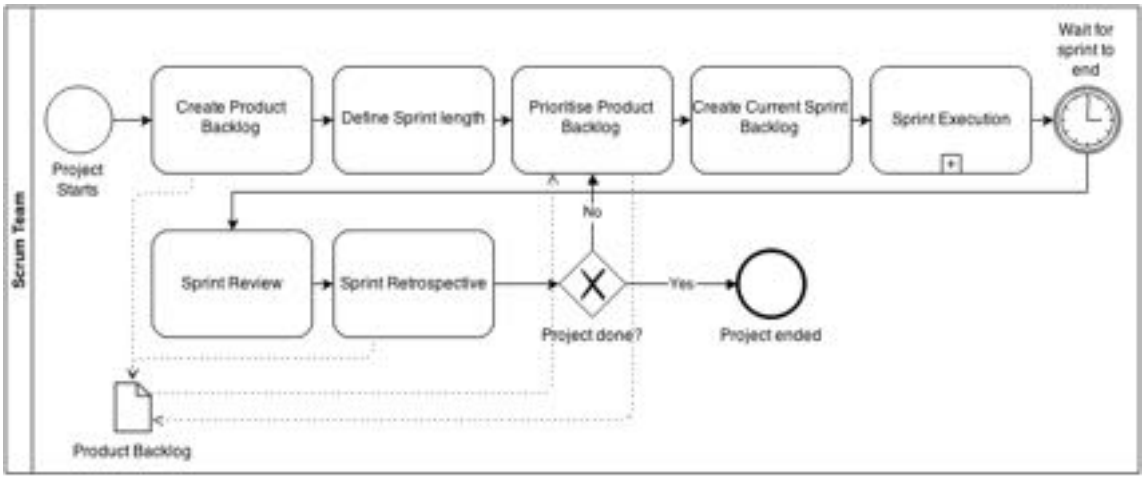

Figura 5.4: BPMN general del proceso Scrum

5.3 Pruebas & Test Driven Development (TDD)

Probar el código realizado es una tarea fundamental en el desarrollo de cualquier proyecto de software. Sin embargo definir el alcance y el tipo de pruebas a realizar es una tarea que no se suele hacer de forma óptima y hace que el código llegue hasta el usuario final con problemas no detectados. El objetivo de esta sección es determinar diferentes tipos de pruebas de software de acuerdo al alcance de su evaluación para con base a esto poder determinar las pruebas que hay que realizar en un proyecto de software.

5.3.1 Tipo de pruebas

Básicamente a la hora de probar se busca verificar que el software cumple con sus funciones, y que lo hace bajo ciertos parámetros de calidad definidos con el cliente. Es así que los tipos de pruebas se clasifican en dos grandes categorías:

Pruebas funcionales: Este tipo de pruebas son las más claras de entender, pues se enfocan en verificar que el software cumple con los requerimientos funcionales establecidos. Es decir, el software hace lo que se le pide. Si se estableció que debía sumar dos números, la prueba debe verificar que dados 2 números el resultado devuelto es el correcto.

Pruebas no funcionales: Son las pruebas enfocadas en medir aspectos relacionados con el rendimiento, seguridad, usabilidad, escalabilidad, mantenibilidad y portabilidad del producto. Estas pruebas van asociadas a verificar la calidad del producto. ¿De qué me sirve que el programa haga lo que se le pide, si se demora mucho al hacerlo? El programa funciona bien si lo están usando 10 usuarios, pero cuando lo utilizan más de 15 empieza a ponerse lento o se bloquea. Un usuario pudo ingresar al programa sin digitar la clave de usuario. Estos y otros casos son ejemplos de lo que se considera requerimientos no funcionales, pues apuntan a aspectos que aunque no son relacionados con la función principal del programa (quiero que sume dos números), si pueden ser importantes para el éxito del proyecto (debe poder sumar billones, debe tener un resultado en menos de 1 milisegundo, debe poder atender a 1000 usuarios concurrentemente...).

Estos tipos de pruebas van enmarcadas en diferentes niveles de acuerdo al alcance de la misma. El nivel más bajo son las pruebas unitarias, destinadas a probar la función de un componente o una sección del código. Son desarrolladas por el programador y son la primera línea de batalla para encontrar problemas en el código. El siguiente nivel son las pruebas de integración, en las cuales se verifica que los diferentes módulos y componentes anteriormente probados unitariamente interactúan de forma correcta. En la sección de integración continua

o *Continuous Integration* se abordará más específicamente este nivel de pruebas. En el nivel de pruebas de sistema se pretende verificar el funcionamiento total, o end-to-end, del software como un todo. Usualmente involucra el despliegue del software en un ambiente de pruebas, y se busca que las pruebas correspondientes a este nivel sean realizadas por un área diferente a la que desarrolló el código. Si el software pasa los anteriores niveles de pruebas se considera que puede llegar al nivel de aceptación por parte del usuario, en donde el usuario final verifica que el software cumple con todos los requerimientos pactados.

Test Driven Development

En cada nivel usualmente se desarrolla primero el código y luego se prueba. Es básicamente lo que la intuición nos dice, pero, ¿Qué pasaría si primero se desarrollaran las pruebas unitarias y de integración y luego se creara el código? Esto es precisamente lo que la metodología de desarrollo orientado a pruebas (Test Driven Development o TDD) nos dice, pues permite encontrar errores de diseño sin que se haya digitado una sola línea de código. En concordancia con los objetivos de lean y Scrum, TDD es una técnica complementaria al desarrollo de software basado en Scrum. Esta metodología viene de las mejores prácticas según la documentación de desarrollo ágil y agrega valor al cliente en varias formas sin incrementar burocracia ni desperdicios en artefactos innecesarios.

TDD como su nombre lo indica, tiene un énfasis especial en pruebas de software. Su objetivo principal es definir los requerimientos y especificaciones de software en forma de pruebas funcionales. La forma recomendada de lograrlo es escribiendo las pruebas antes de que se escriba el código de la aplicación y entonces el desarrollador tiene un objetivo claro, lograr conseguir que las pruebas sean exitosas. Esta técnica se conoce como *Test First Development* o TFD por sus iniciales en inglés.

Una ventaja de usar esta aproximación es que al definir primero las pruebas estas se pueden utilizar como un artefacto entre el cliente y el equipo de desarrollo para definir los requerimientos y el comportamiento que el software va a tener. Esto tiene la ventaja que el cliente entiende exactamente qué necesita de la aplicación y que entradas y salidas necesita para aprobar el proyecto como terminado.

Adicionalmente, TDD fuerza a los desarrolladores a tratar las pruebas automatizadas como un requisito crítico del desarrollo y no una opción -si dispone de tiempo-, aumentando así la calidad y disminuyendo el número de defectos en el producto final.

Implementando TDD

A continuación, en la figura 5.5, se describe el diagrama de actividades BPMN que típicamente debería seguir un desarrollador usando TDD.

Como se puede apreciar en la figura, el desarrollador primero se asegura de tener la versión más reciente del proyecto y después crea una prueba que demuestre que la funcionalidad a implementar está correctamente implementada. Es claro que inicialmente esta prueba va a fallar, debido a que la funcionalidad todavía no está implementada. Esto se conoce como “Red, Green, Refactor” (RGR), debido a que la mayoría del software para pruebas colorea en rojo una prueba cuando falla y verde cuando es exitosa. También se puede ver que el desarrollador comparte sus pruebas unitarias con las pruebas de los demás desarrolladores y las pruebas automatizadas de otros estilos que existan en el proyecto (por ejemplo de aceptación, etc.) que se encuentren en las librerías de pruebas almacenadas.

Nótese también en el diagrama que el desarrollador solo implementa una prueba a la vez. Una vez esta prueba pase, guarda el resultado en el sistema de control de versiones que esté utilizando. Esto significa dos cosas importantes. La primera es que todo código que existe en el repositorio es funcional y tiene una prueba que lo demuestra. Segundo, que el desarrollador siempre hace implementaciones pequeñas con resultados demostrables y por lo tanto el desarrollo

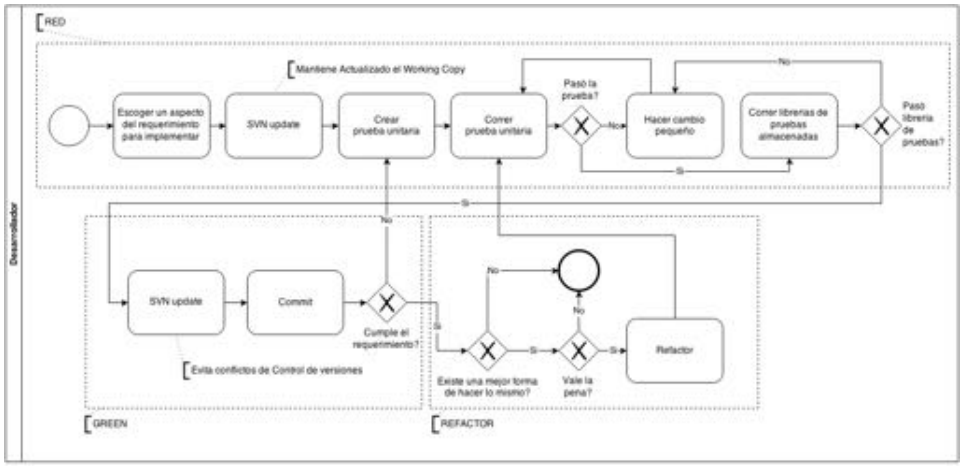

Figura 5.5: BPMN del proceso TDD

es iterativo necesariamente. Esto cumple todos los objetivos lean y Scrum y adicionalmente reduce la cantidad de defectos con las que sale el producto, impulsando la filosofía de perfección de lean. Más aún, el hecho de que el desarrollador ejecute las pruebas almacenadas por los demás desarrolladores implica que este se asegura que sus cambios no afecten lo que los otros han avanzado. Esto evita que el desarrollo de nuevas funcionalidades introduzca nuevos errores en lo que ya se tiene. Además, le otorga más confianza al desarrollador de hacer cambios agresivos que mejoren el desempeño y la calidad del producto terminado, incrementando así su velocidad personal de desarrollo de software.

Por otro lado, la metodología TDD también se debe aplicar a nivel de pruebas de aceptación. Esto significa que tanto los avances de código como los avances en funcionalidades que ofrecerá el producto, se desarrollan iterativamente y con 100% de cobertura de pruebas[Bec02].

La figura 5.6 muestra varios elementos muy importantes. Primero, se puede ver que las pruebas de aceptación siempre las desarrolla el desarrollador del producto, asegurando así que el desarrollador entienda bien lo que realmente se quiere lograr. Segundo, se muestra que a pesar de que una prueba de aceptación resulte exitosa, no significa que el requerimiento esté completo. Esto es debido a que las pruebas de aceptación deberían contemplar varios casos de uso sobre la misma funcionalidad, en particular, deben contemplar los casos normales (con valores con los que cotidianamente se va a utilizar dicho requerimiento) y casos de límite (con los valores máximos y mínimos que el requerimiento dice que debería soportar la aplicación). No es raro pensar que cuando se pone al límite una aplicación, esta puede responder de manera no esperada. Utilizando la metodología TDD a este nivel asegura que los desarrolladores no abarquen grandes tareas en un solo momento, sino que vayan descubriendo los posibles errores que puede tener la aplicación de manera iterativa y en entregas cortas.

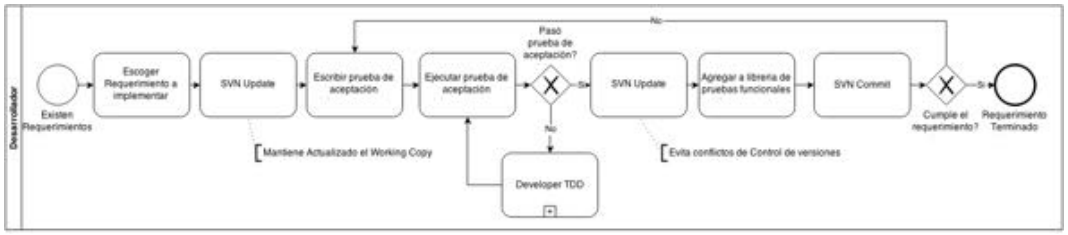

Figura 5.6: BPMN del proceso TDD en procesos de aceptación

Está todavía pendiente la definición de la letra R de Refactor de RGR "Red-Green-Refactor". Esta actividad en TDD hace referencia a que con un conjunto completo y operativo de pruebas, es posible de manera segura modificar el código para mejorarlo, actuando las pruebas como una verdadera "red de protección" del programador, capaz de detener a tiempo los efectos desafortunados que puedan traer los cambios recién introducidos.

Pruebas como documentación

Dado que las metodologías ágiles promueven eliminar documentación innecesaria, se puede considerar las pruebas unitarias como una forma de documentación de código. En realidad, a diferencia de otras formas de documentación de código que con el tiempo tienden a quedar desactualizadas, las pruebas unitarias siempre deben estar actualizadas o si no fallan. Pero si en vez de documentar código, se documentan bien las pruebas, los desarrolladores no solo van a tener toda la documentación en un solo lugar, sino que también contarán con la información de qué valores de entrada y salida espera cada método del código o cada requisito funcional en el caso de pruebas de aceptación, dando así mejor información que la que pueden ofrecer diagramas o documentos por fuera del ambiente de desarrollo.

Si se empieza a aceptar las pruebas como una forma de documentación, es posible implementarlas de tal forma que sean legibles tanto para los desarrolladores como los stakeholders de negocio, dejando así menos y mejor documentación en el proyecto. Existen también varias herramientas que hacen este proceso más transparente para el usuario final y el usuario de negocio, conocidas como herramientas de pruebas de comportamiento.

TDD para manejar código legado

Con frecuencia sucede -especialmente en organizaciones que llevan operando cierto tiempo con aplicaciones a la medida-, que su tecnología está des-actualizada o funcionando parcialmente. Esto es especialmente cierto si se utilizaron malos proveedores de desarrollo de software o si una aplicación lleva más de diez años en funcionamiento. Surge así el problema de manejar el código que todavía sigue funcionando con el código nuevo que no solo utiliza tecnologías nuevas sino que se hicieron con metodologías de desarrollo y estándares de calidad modernos.

Si bien es tentador volver a hacer una aplicación desde el principio y tener dichas funcionalidades "bien hechas" desde el inicio, existen varios problemas con esta aproximación. El primero de ellos es que el tener que hacer algo desde el principio dado que ya existe y parece que funciona es un desperdicio. El segundo problema tiene que ver con que el software, a medida que se va mejorando, va adquiriendo madurez y estabilidad con la depuración de problemas, que al volverlo hacer desde el inicio se pierde. Finalmente el tercer problema tiene que ver con que se trata de perseguir un objetivo móvil, toda vez que lo que parece "bien hecho" en un momento tecnológico dado, estará tristemente des-actualizado en sólo un par de años después.

Por todas estas razones, se debe pensar bien los beneficios contra las posibles desventajas de volver a hacer una aplicación existente desde el principio y esto va a depender mucho del contexto de la aplicación.

Sin embargo, TDD es una técnica muy poderosa que permite abordar los problemas relacionados con aplicaciones legadas. En general, el aplicar TDD en aplicaciones antiguas o proyectos que mezclan código nuevo y viejo implica el manejo de un nuevo estado del código en el proyecto. Este nuevo estado se define así: "todo código que no tenga pruebas unitarias asociadas es considerado código legado".

Si aplicamos este nuevo estado al proceso TDD que ya definimos, podemos ver que igual tiene sentido, pues dado que el desarrollador primero escribe las pruebas, esto lo fuerza a primero escribir pruebas sobre código que no las tiene y verificar si realmente está funcionando. La elaboración de pruebas sobre código legado es así también una "red de protección" que permite

hacer cambios sobre aplicaciones heredadas, sin temer a los efectos colaterales causados por los cambios[Fea04].

El elaborar pruebas sobre código legado puede traer sorpresas gratas al desarrollador, ya que es posible que durante su etapa de exploración del código legado, encuentre que ya existe la funcionalidad que necesitaba desarrollar. Esto implica que ahora solo tiene que desarrollar las pruebas que demuestran que si está funcionando como se espera, o en su defecto, solo debe modificar algo ya hecho para que empiece a funcionar. Son grandes ahorros en tiempo que de otra manera no se hubieran obtenido.

5.4 Integración Continua - CI

La integración continua es un concepto de vital importancia para el desarrollo de software con la filosofía lean. La integración continua busca hacer integraciones automáticas frecuentes a medida de que se desarrolla el software, de tal modo que los cambios de integración son incrementales y siempre se hacen sobre una base funcional. Esto permite encontrar fallos tempranamente, ya que algunos errores solo salen a la luz cuando se integran diferentes módulos y sabemos que entre más tarde sea encontrado un problema, es más costoso de solucionar.

El proceso de integración se hace periódica y frecuentemente, incluso con varias integraciones el mismo día. Así, en vez de hacer una integración enorme, se mantiene siempre una versión disponible para mostrar en caso de que el proyecto esté siendo auditado. Dicha versión además puede ser utilizada por el área usuaria con el fin de obtener retroalimentación. Adicionalmente, se pueden correr otros tipos de pruebas distintas de las unitarias, como por ejemplo pruebas de requerimientos no funcionales, lo que nos brinda un notorio beneficio desde el punto de vista de calidad.

La integración continua, desde el punto de vista técnico, se logra con un servidor de CI (las siglas en inglés de Continuous Integration). Dicho servidor está encargado de descargar la última versión que exista en un administrador de versiones, como por ejemplo Subversion, Mercurial, Microsoft visual source safe o CVS. Dicha descarga se compila y automáticamente se le corren todas las pruebas definidas, es decir, que el software es probado a medida que se va desarrollando, teniendo un control de calidad permanente mientras se avanza en el ciclo de vida. Gracias a esto, cumplimos un principio de lean fundamental, que es incorporar calidad el producto a medida que se va desarrollando, y no al final.

Para el servidor de CI, existen diferente herramientas de software, como por ejemplo Bamboo, Continuum, Jenkins, Anthill (para proyectos Java) o Team Foundation Build para .Net, que se encargan de controlar las ejecuciones, apoyadas en otras herramientas como Ant o Maven.

6. Seguridad (Resiliencia) y el ciclo de vida

> The only thing not digital in cyberwar is the blood
>
> *Taylor Swift*

De acuerdo con los acuerdos de Basilea [Ris14], el riesgo operacional se define como:

> *. . . "risk of loss resulting from inadequate or failed internal processes, people and systems or from external events. This definition includes legal risk but excludes strategic and reputational risk. Legal risk includes, but is not limited to, exposure to fines, penalties, or punitive damages resulting from supervisory actions, as well as private settlements". . .*

Esta definición agrupa eventos tanto de seguridad de la información, como relacionados con continuidad de operaciones (*failed internal processes*) y cumplimiento. Estos eventos, sin excepción, tienen que ver con tres atributos fundamentales: confidencialidad, integridad y disponibilidad.

Prácticamente cualquier requerimiento de riesgo operacional tiene como objetivo asegurar un aspecto que cae dentro de estas tres categorías. Lo que suele pasar desapercibido es que esos términos precisos no son fáciles de aplicar de manera específica: si bien es mucho más claro entender qué es un sistema "confidencial" que un sistema "seguro", sigue siendo demasiado impreciso el concepto como para poder guiar una decisión de diseño, desarrollo o configuración.

Resulta así prudente definir de manera más explícita cada fase del ciclo de vida del software, permitiendo definir para cada una de ellas las entradas, salidas, límites y actores involucrados. Esto permitirá, en particular, identificar las actividades de riesgo operacional que deben ejecutarse en cada etapa.

Sin entrar en el detalle específico –todavía- de los procesos de gestión de tecnología de una organización, en general cualquier proceso bien definido de fábrica de software pasa secuencialmente por las siguientes fases:

- Definición y especificación de requerimientos,
- Arquitectura y Diseño,
- Plan de pruebas,
- Implementación,
- Ejecución y análisis de pruebas,

- Despliegue,
- Mantenimiento y ajustes,
- Fin de la vida del producto

De acuerdo con las validaciones internas de cada fase, será posible devolverse entre fases, pero no será posible avanzar a la siguiente si no se han cubierto y aprobado los entregables de la fase actual.

Nótese que esta secuencia de etapas no necesariamente está ligada al tan criticado modelo "waterfall" de desarrollo de software. Aún las metodologías ágiles / lean / scrum siguen esta secuencia de pasos, sólo que con alcances más chicos en cada iteración o sprint.

A continuación se describe de manera detallada cada una de las fases:

6.1 Análisis de Riesgos

Este es quizás el recurso más valioso en la búsqueda del objetivo de software resiliente. El análisis de riesgos es el procedimiento crítico que permite analizar y detectar las características y recursos de los adversarios y así darle forma y decidir acerca de los mejores controles de protección para cada adversario.

El afirmar que una aplicación es segura es fundamentalmente una afirmación ambigua, y en muchos casos carente de significado, que denota desconocimiento acerca de la manera en que se construye un sistema de información y la forma en que falla. El esfuerzo más serio que un profesional de seguridad puede promover es el de hacer un sistema razonablemente seguro ante cierto ataque o cierta amenaza específica.

Con el Análisis de Riesgos se busca responder preguntas como: ¿ante qué queremos protegernos? ¿qué nivel de protección es necesario? ¿qué pasa si el escenario se materializa? ¿es "no hacer nada" una opción? ¿qué componentes de la aplicación están en juego?

Es necesario escoger una metodología de análisis de riesgos completa, manejable, flexible, comprensible y detallada.

Debe ser completa, ya que cualquier riesgo omitido debe asumirse como un riesgo que va a materializar eventualmente y sobre el cual no hemos tomado absolutamente ninguna acción preventiva o correctiva.

Una metodología adecuada de Análisis de Riesgos debe permitir identificar Activos, Amenazas y debe permitir estimar la Magnitud del riesgo.

6.1.1 Activos

El primer entregable del proceso de análisis de riesgos es la identificación de activos y su relación con los procesos operativos del negocio: ¿qué es lo que buscamos proteger? ¿qué tan fundamental resulta el activo para los procesos críticos del negocio?

Respecto a activos es posible pensar en cuatro categorías de activos:

- Instalaciones: ubicaciones físicas. Habitualmente, son contenedores de otros activos, como personas, tecnología e información.
- Personas: se trata del talento humano que utiliza otros activos de la organización y posee conocimiento acerca del funcionamiento de la misma.
- Tecnología: infraestructura y aplicaciones que soportan la operación de la organización.

- Información: el núcleo de la organización. Es la información la que permite que una organización se diferencie de otra. Mejores procedimientos, experiencia previa, listas de clientes, etc. Puede estar contenida en activos tecnológicos (v.g. servidores) o en instalaciones (v.g. documentos impresos).

El tipo de activo que se busca proteger, en últimas, es la información. No obstante, para lograrlo de manera exitosa, es necesario con frecuencia proteger los otros tipos de activos de igual manera.

En términos generales, se puede decir que para cada tipo de activo aplican algunos de los requerimientos típicos de resiliencia (confidencialidad, integridad y disponibilidad), mientras que otros resultan simplemente inaplicables.

Está el caso de las instalaciones físicas. Es fácil imaginar que sea necesario considerar la integridad de un edificio o una planta de producción. Lo mismo aplica para la disponibilidad. Pero ¿qué sería la confidencialidad de un edificio? (no es lo mismo que la confidencialidad de la información del edificio: la información es confidencial, el edificio no).

La misma lógica aplica para todos los tipos de activos: una empresa debe considerar la disponibilidad de sus empleados (activos de tipo persona, para efectos de la presente clasificación), pero su confidencialidad e integridad no están habitualmente definidas (puede pensarse en excepciones, como el entorno militar, en que la integridad de sus miembros está frecuentemente expuesta y es una preocupación real, pero no es el caso en la mayoría de ambientes).

Por su parte, un activo de tipo tecnología implica preocupaciones de disponibilidad (con gran frecuencia, la única preocupación de seguridad que se atiende) e integridad, pero, nuevamente, resulta inimaginable el sentido de una frase como “este servidor es confidencial”.

Es la información (aún la información acerca de los otros activos -metadatos) la que tiene, con frecuencia, los tres requerimientos: confidencialidad, integridad y disponibilidad.

El primer paso es, entonces, tan claro como exigente. Debe ser riguroso y extenso. ¿Cuáles son los activos que participan en el proceso que está siendo analizado? Es por esto que resulta crucial la participación de los dueños, gestores y participantes del proceso. ¿Cómo podría una sola persona o área conocer el detalle, las dependencias y los casos excepcionales de cada proceso de una organización compleja? En el capítulo 2.2 del presente documento existe más información sobre la Gestión de Activos que puede orientar mejor la elaboración de este entregable.

6.1.2 Amenazas

Ya se tiene el listado de activos que se busca proteger. ¿Contra qué? ¿será la implementación de un canal SSL la manera de evitar un ataque de denegación de servicio? El primer paso para responder esta pregunta es mediante otra pregunta: ¿contra quién? ¿qué actores pueden estar involucrados en un compromiso de riesgo operacional? Debe recordarse que para esta metodología, el término amenaza se refiere a un actor del sistema.

En este punto, es necesario volver a recalcar una frase que se ha usado con anterioridad: pensar como atacante. Para lograr un cubrimiento detallado y riguroso, no es suficiente pensar “¿qué haría yo para quebrar el sistema?”. La pregunta debe ser mucho más maliciosa: “¿quién estaría interesado en hacer que el sistema hiciera algo para lo que no está programado?”.

Hay que ser malicioso. Pensar mal. Ser “sanamente paranoico”. ¿Podría el usuario estándar de la aplicación querer efectuar alguna acción no autorizada? ¿y si el administrador quisiera ejecutar tareas sin dejar registro, podría hacerlo? ¿y un usuario de otra dependencia, que no tiene acceso normalmente a la aplicación?

Pero los actores no son necesariamente humanos. Debe recordarse que la seguridad de la información puede verse comprometida también por casos de mal uso, aún sin ser malinten-

cionados. ¿Y si una tarea de otro servicio se quedara en un ciclo infinito realizando peticiones al sistema, afectaría su disponibilidad? ¿qué pasa si una conexión no se cierra apropiadamente? ¿puede un servicio caído afectar el comportamiento del sistema? Se recomienda revisar el capítulo 2.3 del presente documento donde se aborda el proceso de Gestión de Riesgos para obtener más información.

6.1.3 Magnitud del Riesgo

En este punto, se conoce el sistema y las amenazas que buscan comprometerlo. Combinando los dos conjuntos de información, ¿es posible materializar una amenaza? Esta intersección entre lo que se busca perpetrar y lo que el sistema permite que salga mal constituye un listado de vulnerabilidades.

Considérese una serie de eventos desafortunados:

- Un asteroide cayendo sobre un servidor.
- La caída simultánea de todos los enrutadores del *core* de la red de la organización.
- La interrupción del fluido eléctrico durante 3 días.
- El bloqueo de un servidor de aplicaciones por una excepción mal manejada.
- El daño de una terminal de trabajo de un usuario.
- Un usuario registra erróneamente su documento de identidad en una aplicación particular.

Ahora bien: ¿cuál de los eventos descritos es más probable? ¿y cuál es más grave si llega a ocurrir?

La multiplicación de la probabilidad (de 0 a 1) y el impacto (definido en una escala numérica arbitraria) constituye la magnitud del riesgo que se está evaluando. Si la probabilidad es muy baja, quizás el riesgo se ignore. Nadie construye refugios subterráneos para sus servidores para protegerlos de un improbable meteorito en un servidor. Si el impacto es bajo, será muy probable también que el riesgo se ignore: por ejemplo, es innecesario en muchos casos implementar un corrector ortográfico o un validador de expresiones regulares en el sistema de comentarios de una página. En la sección 5.4.1 se discutirá una metodología de valoración de riesgos de software que puede ayudar en esta tarea.

6.2 Casos de abuso, mal uso, discontinuidad y cumplimiento

Así como un caso de uso convencional busca describir el comportamiento de un sistema de información en escenarios convencionales (v.g. “si el usuario da clic en el botón Aceptar, se debe registrar el nuevo producto”), los casos de abuso y mal uso buscan describir lo que puede salir mal en un escenario, un anti-requerimiento.

Los casos de abuso, en primera instancia, buscan describir un escenario de uso malintencionado de la aplicación que derive en un compromiso en un requerimiento de confidencialidad, integridad o disponibilidad (v.g. “si el usuario da clic en el botón Transferir 10 veces, se realizan 10 transferencias”).

6.2.1 Casos de abuso

En un caso de abuso, se busca describir el escenario “seguro” y los controles que son necesarios para alcanzarlo. Si bien no existe un artefacto estándar para describir un caso de abuso, un buen ejemplo podría ser el que se muestra a continuación en la figura 6.1.

El diagrama contiene la siguiente información:

- Identificador: campo designado para identificar de manera unívoca el caso de abuso.

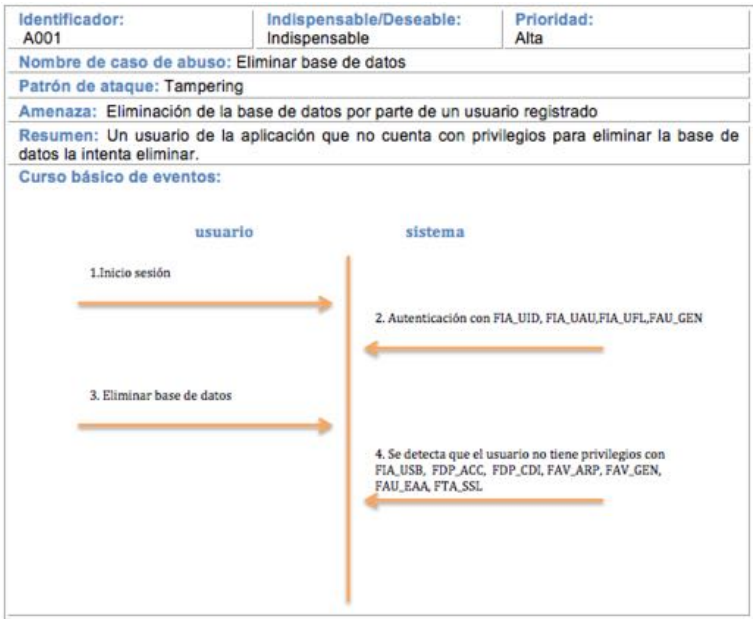

Figura 6.1: Ejemplo de caso de abuso

- Indispensable/deseable: determina la criticidad de la implementación de controles para el caso de abuso: ¿es indispensable que se implemente? ¿se trata sólo de una característica deseable?
- Prioridad: determina la prioridad de atención del caso de abuso.
- Nombre: descripción sucinta del ataque que se busca evitar.
- Patrón de ataque: el tipo de ataque que se está describiendo. Se abordará este tema en detalle más adelante.
- Amenaza: en términos de esta metodología, una amenaza es un actor en relación al sistema (v.g. usuario válido, usuario inactivo, atacante externo, delincuencia organizada, mafia, gobiernos extranjeros, etc.).
- Resumen: descripción detallada del ataque que se busca evitar.
- Curso básico de eventos: es el elemento clave de un caso de abuso. El objetivo es describir la interacción entre varios elementos del sistema (en el ejemplo, entre un usuario y el sistema como un todo), de manera tal que las acciones de los actores, los controles implementados y las respuestas de la aplicación queden adecuadamente documentadas. Usualmente los controles son tomados de la norma ISO/IEC 15408-2(véase [15414], [15411a] y [15411b]) (v.g "FAU_GEN: AUdit data-GENeration", corresponde a la familia Generación de datos de auditoría, "FIA_USB, Identification and Authentication-User Subject Binding", corresponde a Identificación y Autenticación- vinculación de usuario -operador humano- al sujeto o entidad del sistema que representa al usuario).

Para el patrón de ataque es conveniente utilizar STRIDE, el modelo de amenazas de Microsoft®. STRIDE propone, como se mencionó dentro del numeral 5.1, los siguientes patrones de ataque (cada letra de la palabra STRIDE es la inicial de un patrón de ataque): *Spoofing identity* (suplantación de identidad), *Tampering with data* (alteración de datos), *Repudiation* (repudiación o negación de alguna acción), *Information disclosure* (fuga de información), *Denial of service* (denegación de servicio), *Elevation of privilege* (elevación de privilegios). En la sección 5.3.5 se explica este en mayor profundidad.

6.2.2 Casos de mal uso

Los casos de mal uso, por otra parte, corresponden a otro tipo de escenarios que pueden comprometer el sistema. Bajo esta mentalidad, el sistema no está bajo ataque. El sistema está siendo utilizado normalmente, pero el mal uso que se le dé puede comprometer la confidencialidad, integridad o disponibilidad de la información (v.g. un usuario hace clic en el botón de eliminar el último mensaje recibido, la página se tarda en responder, y por ende el usuario hace clic muchas veces en el botón, por lo cual, elimina más mensajes de los que tenía intención de eliminar y que necesitaba para continuar con su trabajo.).

En los casos de mal uso, el flujo debe describir el comportamiento errado del usuario: qué acción ejecuta el usuario y cuál es el efecto no deseado que se obtiene. Por ejemplo, un caso de mal uso puede ser el que se muestra a continuación en la figura 6.2.

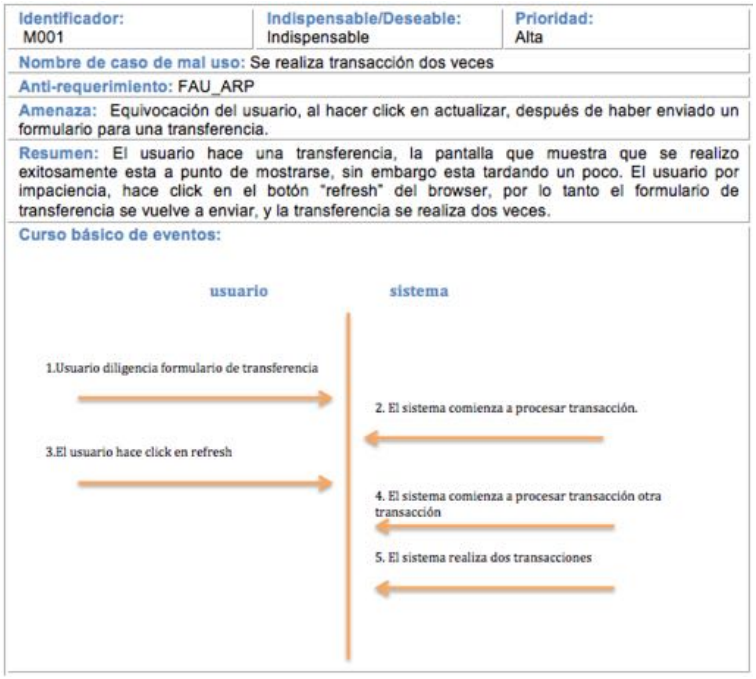

Figura 6.2: Ejemplo de caso de mal uso

El diagrama planteado contiene la siguiente información:

- Identificador: campo designado para identificar de manera unívoca el caso de mal uso.
- Indispensable/deseable: determina la criticidad de la implementación de controles para el caso de mal uso: ¿Es indispensable que se implemente? ¿se trata sólo de un control deseable?
- Prioridad: determina la prioridad de atención del caso de mal uso.
- Nombre: descripción sucinta del ataque que se busca evitar.
- Anti-requerimiento: identificación del control (uno solo) que hace falta para prevenir un incidente asociado al escenario planteado, es decir, un requerimiento que el sistema no tiene . Más adelante se abordará en detalle el tema de los posibles controles que pueden ser utilizados.
- Resumen: descripción elocuente del ataque que se busca evitar.
- Amenaza: en términos de esta metodología, una amenaza es un actor en relación al sistema (v.g. usuario válido, usuario inactivo, personal externo, etc.).
- Curso básico de eventos: es, también, el elemento clave de un caso de mal uso. El objetivo es describir la interacción entre varios elementos del sistema (típicamente, como en el

ejemplo, entre un usuario y el sistema como un todo), de manera tal que los actores realizan acciones y el sistema envía una respuesta que termina en la explotación accidental de la vulnerabilidad planteada.

6.2.3 Casos de discontinuidad

Los casos de discontinuidad son similares a los casos de abuso/mal uso que se abordaron anteriormente, pero a diferencia de estos, no son causados por un usuario malintencionado o una acción equivocada. Son causados por condiciones externas a la organización, como por ejemplo, un desastre natural o una falla de corriente eléctrica en el sector. En estos se muestra como se debería responder a una situación de estas.

Vemos uno a continuación en la figura 6.3.

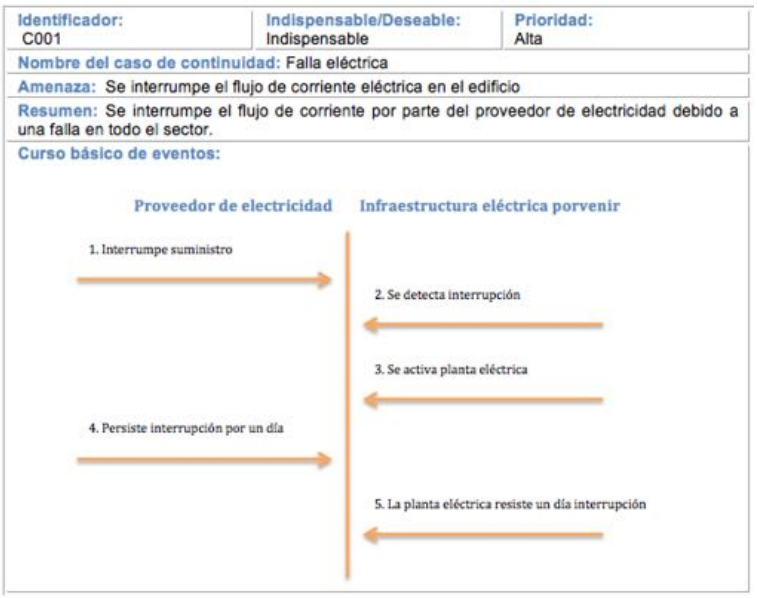

Figura 6.3: Ejemplo de caso de discontinuidad

6.2.4 Requerimientos de cumplimiento

Los requerimientos de cumplimiento, debemos usarlos cuando existe una normatividad que afecta el negocio y que si no se cumple, puede generarse una multa debido a que la legislación considera que es lo correcto y nos debemos ceñir a eso.

Estos requerimientos de cumplimiento los podemos definir en la figura 6.4 en un tabla con los siguientes campos[IMJ11]:

El requerimiento se refiere a una característica que debe tener el software. El objetivo indica la razón de ser de esa característica, y la especificación indicar en la aplicación que partes son afectadas.

6.2.5 Métodos de Ayuda

Existen métodos que nos facilitan encontrar necesidades no funcionales que no se identifican a simple vista. A continuación, se presentan dos, STRIDE y una plantilla de requerimientos, los cuales pueden ser de gran ayuda para no dejar pasar requerimientos que involucran controles importantes para mitigar riesgos.

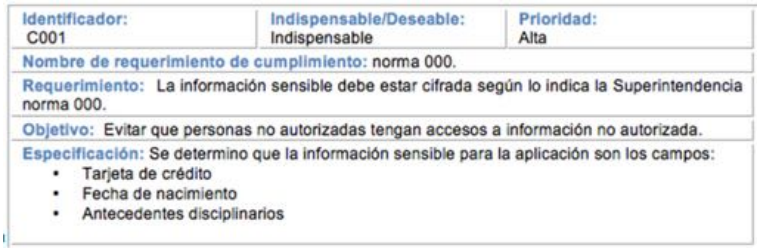

Figura 6.4: Ejemplo de requerimientos de Cumplimiento

STRIDE, patrones de ataque y controles

En la terminología que se ha venido utilizando, hay dos conceptos que posiblemente permanezcan nebulosos en este punto, y resulta necesario prestarles algo de atención antes de continuar: patrones de ataque y controles.

Un patrón de ataque es una descripción de los diferentes perfiles que puede tener un ataque. Las clasificaciones son diversas y hay algunas que tienen decenas de categorías diferentes de ataques. Con el tiempo, estas clasificaciones pueden llevar a análisis de riesgos mucho más detallados y rigurosos. Afortunadamente, hay un patrón ([McG06]) lo suficientemente simple para empezar en estos procedimientos sin cargar con decisiones de clasificación complejas o tener conocimientos técnicos muy profundos sobre cada vector posible de ataque.

Este patrón se llama STRIDE. Este patrón considera seis posibles vectores de ataque, que son los que constituyen la sigla en inglés:

- Spoofing (suplantación): el ataque consiste en la suplantación de identidad de un usuario. Con frecuencia, este ataque se utilizará como trampolín para ejecutar otros.
- Tampering (alteración): la información del sistema es alterada. Se pierden, agregan o alteran registros de manera ilegítima.
- Repudiation (repudio): "yo no fui". El perpetrador de una acción maliciosa puede salir airoso debido a que no hay mecanismos que permitan asociarlo de manera inequívoca con la acción.
- Information disclosure (fuga de información): la información confidencial se hace accesible por usuarios no autorizados (v.g. empleados sin autorización, la competencia, el público general, etc.).
- Denial of service (denegación de servicio): interrupción no programada de la prestación de un servicio particular (v.g. bloquear una página web, impedir el funcionamiento de un access point, etc.)
- Escalation of privilege (escalamiento de privilegios): adquisición no autorizada de privilegios de usuario (v.g. lograr privilegios de administración desde una cuenta de invitado en el sistema operativo, obtener rol de auditoría sin ser auditor, etc.).

Con frecuencia, se verá que más de un vector debe ser utilizado para llevar a cabo un ataque (v.g. lograr acceso no autorizado como invitado (S), obtener privilegios de administración (E) y, finalmente, obtener reportes de pagos clasificados (I)). Como se dijo, habrá muchas variaciones de cada uno de estos vectores, resultando en patrones mucho más complejos.

El otro concepto que conviene tener presente es el de control. No tanto por su definición, porque no es más que cualquier mecanismo, función, política o herramienta que permita reducir la probabilidad o el impacto de un riesgo, sino por su interpretación y elección.

Lo que hace necesario detenerse en el concepto de control es la variedad y dificultad de la elección. No hay controles perfectos, y no hay controles indispensables. Por ejemplo, existen

escenarios en los que un control básico, intuitivo y de amplio uso -como las contraseñas- resulta inválido e incluso peligroso. Adicionalmente, "el todo no es la suma de las partes". Puesto en otros términos: instalar tres antivirus en un mismo equipo no es radicalmente mejor que instalar uno.

Plantilla de requerimientos no funcionales

Muchas veces los requerimientos no funcionales correctos surgen de un análisis de riesgos juicioso. Sin embargo, existe la posibilidad de que por falta de experiencia o alguna razón similar, se omitan elementos fundamentales o importantes. Para esto podemos utilizar la siguiente plantilla, que se puede revisar para recordar una serie de requerimientos que pueden ser solicitados, y que si decidimos no incluirlos, sea de manera consciente. El listado de requerimientos, agrupado por atributos de calidad, se muestra a continuación en la figura 6.5.

Requerimientos de identificación
- Id de usuario único
- Prevención de puertas traseras
- Código de identificador de procesos
- Inhabilitar IDs automáticamente

Requerimientos de autenticación
- Credenciales de seguridad
- Protección contra ataques de repetición
- Protección de adivinación de credenciales
- Autenticación de servidor
- Reautenticación
- Protección de credenciales
- Cambio de contraseñas
- Caducidad de contraseñas
- Cambio de password seguro
- Prevenir reutilización de contraseñas
- Complejidad de contraseñas
- Protección de autenticación biométrica

Requerimientos de autorización
- Permisos de acceso Access Rights
- Protección de biométrico

Información de autenticación
- Bloqueo de cuentas
- Mostrar último inicio de sesión
- Expiración de sesión
- Restricción de acceso
- Privilegios de usuario y grupo
- Control de acceso basado en roles
- Mecanismo de control de recursos
- Sesiones concurrentes

Requerimientos de auditoría de seguridad
- Log de auditoría
- Logging de información de autenticación
- Logging de eventos específicos
- Acciones en fallos de log de auditoría
- Archivación de logs de auditoría Archival
- Revisión de log y reportes
- Logging de información específica
- Protección de log de auditoría

Requerimientos de confidencialidad
- Protección de información sensible
- Seguridad de llaves criptográficas
- Fortaleza de llave criptográfica
- Expiración de llaves criptográficas
- Revocación de llaves criptográficas
- Recuperación de llave criptográfica
- Seguridad de llave de firma
- Seguridad de las llaves que firman

Requerimientos de integridad
- Chequeo de integridad Integrity Checking
- Identificación de fuente
- Integridad de header
- Protección de ataques de repetición
- Integridad de información sensible
- Integridad de logs
- Chequeos de integridad

Requerimientos de disponibilidad
- Escalabilidad de seguridad
- Capacidad de monitorear disponibilidad

Requerimientos de no repudio
- Logging de información específica Secure
- Estampa de tiempo Time Stamping
- Firmas digitales

Requerimientos de inmunidad
- Negación por default
- Limitación de alcance
- Límite de ejecución
- Control de acceso basado en CRUD

Requerimientos de supervivencias
- Protección de buffer overflow
- Registrar alertas de mal funcionamiento
- Registrar reinicios del sistema

Requerimientos de mantenimiento de sistema
- Rastreo de fuente Source Tracking
- Chequeos de integridad del sistema
- Reporte de snapshot del sistema
- Recuperación segura Secure Recovery
- Seguridad de datos de backup/restauración
- Restauración desde checks de backups
- Recuperación de configuración de seguridad
- Parámetros de seguridad a través de los reinicios del sistema

Requerimientos de privacidad
- Clasificación e identificación de información personal
- Deshabilitar uso de información privada durante pruebas y desarrollo
- Provisión de aviso
- Comunicación a individuos
- Consentimiento implícito o explícito
- Consentimiento explícito para información sensible
- Consentimiento para transferencias online
- Comunicación a individuos
- Uso de información personal
- Retención de información personal
- Destrucción de información personal
- Acceso de individuos a su información personal
- Actualización o corrección de información personal
- Divulgación de información personal
- Protección de información personal
- Control de acceso lógicos
- Controles de acceso físicos
- Exactitud y completitud de información personal
- Investigación, quejas y monitoreo de auditoría en marcha

Figura 6.5: Plantilla de requerimientos no funcionales

6.3 Definición y especificación de requerimientos no funcionales

Esta es la etapa en la que nace un requerimiento. A partir de una necesidad de negocio (iniciativa, requerimiento funcional, necesidad de corrección de un error, cambio normativo, entre otras), es necesario definir en términos técnicos la necesidad y el alcance del cambio.

Habitualmente, las organizaciones tienen muy claro cómo definir los aspectos funcionales de un requerimiento (qué debe hacer, cómo lo debe hacer, con qué se debe conectar, etc.). Hay usualmente documentación exhaustiva, personal capacitado y procesos adicionales de revisión y autorización que garantizan altos niveles de refinamiento y gobierno en estos aspectos.

Eventualmente, será posible encontrar en algunos procesos otro tipo de información adicional, conocidos normalmente como requerimientos no funcionales o más frecuentemente, atributos de calidad. Son, grosso modo, características que no constituyen la descripción de la funcionalidad de la aplicación (v.g. agregar un cliente, aprobar un crédito, liquidar una pensión, obtener reporte de transacciones del último mes), pero pueden impactar la percepción de lo que se interpreta como un producto software "de calidad". Algunos de los atributos de calidad más comunes son por ejemplo rendimiento / desempeño, tiempo de recuperación, facilidad de modificación o escalabilidad[FGJ14].

Tristemente con frecuencia obtener una solución informática que pueda ofrecer altos niveles de varios atributos de calidad es costoso o inviable. Por ejemplo, una aplicación modular requiere niveles de desacoplamiento que favoreciendo enormemente la flexibilidad, puede impactar negativamente la eficiencia y los tiempos de respuesta. No se puede tener todo, será necesario analizar las opciones, balancear "tradeoffs" y determinar de manera consciente cuál es el escenario deseado para la organización.

El objetivo fundamental a tener en mente en esta fase es el incluir la seguridad de la información, la continuidad y el cumplimiento, en la lista de atributos de calidad que debe ser analizada en detalle desde la misma concepción de un proyecto de software[Dhi11].

El primer paso necesario es la introducción de una nueva notación. Los diagramas y documentos habituales de requerimientos funcionales y casos de uso se centran en los aspectos funcionales del escenario (excepcionalmente, incorporan elementos de atributos de calidad en general) y no proporcionan herramientas para hablar de riesgo operacional. Resulta necesario introducir los conceptos y artefactos como casos de abuso, mal uso, discontinuidad y requerimientos de cumplimiento.

Una vez comprendido el concepto y la notación asociada a estos dos mecanismos, el siguiente paso es ponerlo en práctica: ¿qué puede salir mal? ¿qué impacto puede tener si sale mal? El análisis de riesgos es el procedimiento más crítico de todo el proceso, desde el punto de vista de riesgo operacional. Porque, ¿cómo saber cómo protegerse, si no se sabe antes a qué se está expuesto?

Finalmente, se incorporará un concepto que, muy de la mano del riesgo operacional, permitirá a la organización prepararse y sobrevivir a escenarios de crisis exitosamente. Se trata del concepto de **resiliencia organizacional** (abordado en el capítulo 2).

Es importante que los requerimientos no funcionales, al igual que los funcionales, estén priorizados. Esto permitirá que se desarrolle primero lo más importante, de tal modo que se obtenga mayor beneficio del software lo más pronto. Esta priorización, es complicada en cuanto a los requerimientos de seguridad, debido a que en ocasiones es difícil apreciar cuál requerimiento es más importante que otro. Una forma de abordar este tema, es dar prioridad a los requerimientos que nos mitigan o eliminan riesgos más grandes. Sabemos que el riesgo, es la probabilidad por el impacto, por lo que simplemente debemos hallar dicha probabilidad e impacto de un riesgo para poder calcularlo, ¿Cómo hacemos esto? No es una tarea del todo sencilla, pero podemos ayudarnos con la metodología DREAD que abordaremos a continuación.

6.3.1 DREAD

DREAD (derivado de la sigla en inglés *Damage Potential, Reproducibility, Exploitability, Affected Users & Discoverability*) es un modelo cuantitativo de valoración de riesgos de software que permite tomar decisiones con base en valores comparables y no en la intuición o experiencia

del evaluador.

Conviene hacer énfasis en un punto interesante: para el evaluador experimentado, la experiencia e intuición son, de hecho, una herramienta muy poderosa, que tiende a ofrecer resultados muy acertados y eficientes. No obstante, existen dos barreras contra el uso exclusivo de las decisiones inspiradas en experiencias o una corazonada: por una parte, son extremadamente difíciles de transmitir ("creo que un bus de servicios vendría bien en este punto" o "me da la impresión que hace falta un balanceador de carga" están lejos de ser argumentos persuasivos) y, por la otra, supeditan su efectividad al conocimiento de una persona en particular. Así, si bien una intuición aguda puede ser una guía poderosa, será fundamental apoyarse en herramientas que sostengan su validez.

Como su nombre lo indica, DREAD propone medir cinco parámetros en relación con cada vulnerabilidad identificada:

- Damage potential (potencial de daño): si la vulnerabilidad se explotara correctamente, de 1 a 10, ¿qué tan catastrófico sería el daño ocasionado?
- Reproducibility (reproducibilidad): de 1 a 10, ¿qué tan fácil es reproducir el efecto descrito, una vez ejecutado el ataque?
- Exploitability (explotabilidad): de 1 a 10, ¿qué tan fácil es explotar la vulnerabilidad para ejecutar el ataque?
- Affected users (usuarios afectados): ¿cuál es el porcentaje (normalizado a una escala de 1 a 10) de usuarios de la aplicación que se verían afectados por un ataque exitoso?
- Discoverability (posibilidad de ser descubierto): ¿cuál es la probabilidad (normalizada a una escala de 1 a 10) que un usuario sin conocimiento profundo del funcionamiento del sistema logre identificar el vector de ataque analizado?

Como se puede ver, todos los componentes se normalizan a una escala de 1 a 10 (aun cuando puede no ser su escala natural, como en el caso de las probabilidades), de manera que cada uno de ellos tenga el mismo peso y sea posible obtener una calificación promedio que sopese de manera comparable su impacto.

El objetivo, como se decía, es cuantificar la atención que merece en términos absolutos cada vector de ataque identificado. Nótese que es un factor comparativo entre escenarios analizados por el mismo evaluador. Lo que para un punto de vista puede resultar catastrófico, para otro puede resultar aceptable o hasta irrelevante. De otra forma: un 8 puede significar cosas muy distintas para dos evaluadores.

No obstante, la inclusión de una escala numérica nos permite comparar con facilidad los elementos dentro del mismo conjunto. La documentación de Microsoft® acerca de DREAD sugiere los valores de la tabla 6.1 (tabla traducida de la versión en inglés):

En general, es una propuesta arbitraria. No todas las metodologías de construcción de software involucran conceptos como boletín o *service pack*. Y, más aún, no es claro qué tan prioritario sea un 7. Lo que sin duda debe resultar de la ejecución de un análisis DREAD juicioso es que un resultado DREAD 7 debe ser atendido antes que un resultado DREAD 6.

Los criterios para asignar cada valor, los límites que determinan con qué prioridad atender cada escenario e incluso el procedimiento indicado según cada valor obtenido dependerán, más que de una tabla o un estándar, de los requerimientos, experiencia y prioridades específicos de cada organización.

A continuación, en la tabla 6.2 y tabla 6.3, se presenta un ejemplo de un análisis DREAD efectuado sobre un vector de ataque puntual. Está basado en una serie de diagramas de arquitectura que evidencian la existencia de un módulo de "auditoría selectiva". Dicho de otra forma, cuando se realizan solicitudes a través de la página web o web services, pasaba por este módulo

	Valor promedio mayor a
Nunca	1
Posiblemente en la siguiente versión	2
En la siguiente versión	3
Considerar la generación de un Service Pack	4
Generar un Service Pack	5
Considerar emisión de un boletín	6
Emitir un boletín	7
Considerar emisión de un boletín prioritario	8
Emitir un boletín prioritario	9

Tabla 6.1: Ejemplo del uso de DREAD para Microsoft®

y mantenía registro de todas las transacciones realizadas. Pero cuando se usaban otros canales, como la aplicación móvil, no había forma de alcanzar siquiera tal componente.

Evaluación Cualitativa del Riesgo	
Amenaza	Usuario interno
Componente Afectado	Los componentes que no pasan por auditoría
Descripción	No existen registros de auditoría ni trazabilidad
Resultado	No queda registro de las operaciones de un atacante
Estrategias de mitigación	Hacer obligatorios logs de auditoría

Tabla 6.2: Evaluación Cualitativa DREAD para un vector de ataque específico

Evaluación Cuantitativa del Riesgo **Criterio**	**Puntuación**
Daño potencial	9
Reproducibilidad	9
Explotabilidad	9
Proporción de usuarios afectados	9
Posibilidad de descubrir la vulnerabilidad	4
Promedio	8

Tabla 6.3: Evaluación Cuantitativa DREAD para un vector de ataque específico

6.4 Arquitectura y diseño

Una vez se han definido los objetivos generales en términos de seguridad (casos de abuso, mal uso, activos, amenazas, adversarios), las siguientes etapas serán más intuitivas y menos críticas siempre y cuando se sea riguroso y se siga una máxima fundamental: todas las decisiones subsecuentes se toman con base en el inventario de activos y el análisis de riesgos realizados. No tiene ningún sentido seleccionar un control o un estilo arquitectural porque "todo el mundo lo está haciendo así". Las necesidades, motivadores y preocupaciones varían entre empresas, áreas e incluso productos y así debe ser.

En este punto, los términos vagos empiezan a ser removidos y en su lugar, empiezan a perfilarse magnitudes y afirmaciones claras que permitan remover cualquier lugar para la

ambigüedad. Frases como “la aplicación debe ser segura” o “la información del sistema debe ser confidencial” resultan inaceptables en los entregables de esta etapa.

Y no son aceptables no porque sean poco deseables, sino porque no significan nada. Representan ideales abstractos que de ninguna manera pueden ser medidos o favorecidos. Por esta razón, es necesario definir los requerimientos de seguridad en términos de métricas tangibles: “el N% de intentos de acceso de usuarios sin credenciales válidas debe ser rechazado y reportado”, “el módulo de atención al cliente debe estar disponible el 99.999% del tiempo”.

Desde el punto de vista arquitectónico y diseño es posible pensar en cinco categorías de requerimientos (las dos últimas categorías usualmente están condensadas dentro de las tres primeras):

- Confidencialidad: “sólo personal autorizado”. La información puede ser leída únicamente por las personas o sistemas autorizados para hacerlo. Casos famosos, como WikiLeaks o el caso Snowden, se resumen en la filtración de información confidencial para que pudiera ser leída por el público general.
- Integridad: la información puede ser alterada (cambios, eliminaciones o adiciones) únicamente por el personal autorizado a través de los canales autorizados. Está frecuentemente ligada a escenarios de fraude y eliminación de evidencia.
- Disponibilidad: un sistema de información debe ser disponible y operable (tanto lectura como escritura) para el personal autorizado en el momento en que se requiera, lo que no quiere decir que deba estarlo de manera inmediata. Dicho de otra forma “las solicitudes de autorización de créditos deben ser aprobadas por el sistema en menos de 12 horas” es un requerimiento de rendimiento. “El sistema de solicitudes de autorización de créditos debe recibir solicitudes durante mínimo 23 horas por día” es un requerimiento de disponibilidad.
- No repudio: frecuentemente referido como trazabilidad o incluso con la palabra inglesa *accountability*. Consiste en la necesidad de que las acciones realizadas sobre un sistema tengan un mecanismo que permita llevar registro sobre las transacciones y acciones realizadas sobre el mismo, pero sobre todo, asociar de manera inequívoca un actor con sus acciones. Si bien no suele ser tratado como un atributo de la información, es fundamental en términos de auditoría y atención de incidentes.
- Identificación/Autenticación/Autorización: esta terna es fundamental para promover mecanismos de control de acceso. Con frecuencia se ve como un “meta atributo” de la información, como la fecha de creación de un archivo o su extensión. Los tres pasos persiguen objetivos distintos y complementarios. Tras identificar a un usuario o sistema (v.g. a través de la cédula o un nombre de usuario), es necesario validar que es quien dice ser (v.g. mediante contraseña, token, medida biométrica, etc.) y, finalmente, determinar qué privilegios tiene sobre un recurso dado (v.g. mediante listas de control de acceso o permisos de archivo).

El objetivo es que estas categorías, al igual que STRIDE como patrón de ataque, sirvan de guía para identificar y clasificar los atributos deseables de cada activo de la solución software. Sin embargo, hay que tener en cuenta que existen otros atributos de calidad a considerar en esta etapa como por ejemplo: rendimiento / desempeño, tiempo de recuperación, facilidad de modificación o escalabilidad. Cuesta imaginar escenarios en los que un requerimiento no funcional caiga por fuera de estas categorías.

Por esta etapa sólo resta una última recomendación: evitar consideraciones técnicas detalladas. “El sistema debe tener un sistema de login con usuario y contraseña de mínimo 8 caracteres” es, por mucho, demasiado específico para este nivel. Debe preferirse “el sistema bloqueará el acceso al 100% de usuarios no autorizados”.

6.4.1 ATAM

El método ATAM[1] *(Architecture Tradeoff Analysis Method)* es una técnica de evaluación de arquitecturas de software con relación a sus objetivos de atributos de calidad. El objetivo que persigue esta metodología consiste en llevar todos los atributos de calidad, seguridad, continuidad y cumplimiento incluidos, a un lenguaje común en el que puedan ser comparables y sea posible identificar interacciones entre ellos (v.g. *tradeoff*) y tomar decisiones, acerca de qué tan bien estos objetivos de calidad están alineados con el negocio.

ATAM, busca definir en nueve pasos una manera de armonizar y/o priorizar las diversas voces de los usuarios de la solución, que habrán planteado diferentes preocupaciones en términos muy diversos, como por ejemplo: "los usuarios no deben esperar más de cinco segundos la respuesta a su solicitud". "La aplicación debe funcionar en dispositivos móviles". "La incorporación de una funcionalidad debe tomar máximo un mes". "El sistema debe estar en condiciones de recibir 10000 solicitudes por minuto".

A continuación se presentan los nueve pasos de ATAM:

Paso 1: Presentación de la metodología

ATAM es una metodología útil, pero fundamentalmente desconocida especialmente en las áreas usuarias. Sus condiciones, criterios, procedimientos y métricas son probablemente desconocidos por la mayoría de interlocutores.

Es por esto que ATAM se asegura de enfatizar que el primer paso, antes de abordar el problema, sea la socialización de la metodología. A través de reuniones, manuales u otros mecanismos que se consideren apropiados, todo el grupo involucrado en el proyecto debe tener claro el conjunto de conceptos que se usarán, las etapas por las que hay que atravesar e, idealmente, conocer ejemplos anteriores de ejecución de la metodología que sirvan como referencia.

Paso 2: Presentación de los objetivos del negocio

Este es un proceso ubicuo en cualquier metodología que pretenda aportar valor a una organización, y en este punto del proceso global, es muy posible que ya previamente se hayan delimitado y especificado. Por otra parte, si no es el caso, es el momento de prestarle atención.

Como se ha dicho anteriormente, ninguna organización tiene como objetivo "montar un sistema de información que…" o "prevenir las intrusiones a sistemas informáticos desde…" o "tener una plataforma tecnológica actualizada". Todos estos, entre muchos otros ejemplos imaginables, son medios que permiten alcanzar un fin.

Hasta el momento, se sugería identificar cuáles eran los objetivos de implementar el proceso de negocio que se estaba analizando. En este punto, una variante de esa pregunta puede aportar información valiosa: ¿cuál es el objetivo de implementar ese sistema de información específico?

En términos generales, es de esperar que ambas respuestas apunten hacia la misma dirección. De hecho, sería necesario prestar especial atención si no fuera el caso. ¿Cuál sería la idea de montar el mejor sistema de información de la historia (muy escalable, con disponibilidad casi perfecta, impenetrable, instantáneo en la respuesta a todas las transacciones, o cualquier otro adjetivo muy deseable pero poco realista que se pueda imaginar) si no aporta ningún valor a la organización? Si, por el contrario, ambos conjuntos de objetivos están alineados, la razón de ser de todas las decisiones posteriores estará definida sólidamente y será posible añadir valor con cada paso que se dé.

Paso 3: Presentación de la arquitectura

Con mayor frecuencia de la que se quisiera admitir, las organizaciones empiezan a considerar sus necesidades de seguridad de la información en etapas tardías de la implementación. En algunos casos, es posible incluso que la solución esté más cerca del fin del desarrollo que de su concepción.

[1] http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

Por esta razón, es particularmente común encontrar que en este punto del proceso, ya exista una arquitectura definida (bien porque se definió desde un principio o bien porque se improvisó sobre la marcha, pero en cualquier caso existe un mapa de la solución). Si no es el caso, el insumo en este punto será cualquier avance que se haya hecho al respecto ("por ley hay que tener 99.95% de disponibilidad del servicio de consulta de…", "se espera añadir nuevos módulos en el futuro y la aplicación debe permitir integrarlos desde el principio", "se tiene en mente la implementación de un bus de servicios que orqueste los diferentes elementos del sistema…").

Sea cual sea el camino que se haya seguido hasta este punto en términos de arquitectura, hay una máxima que debe determinar la calidad del material con el que se cuenta: "la calidad de una decisión arquitectural es tan buena (o tan mala) como su justificación". Si la arquitectura descrita y difundida en este punto no supera esta validación, el problema es mucho mayor que el aseguramiento de algunos elementos del sistema. Este es el punto de no retorno: si la arquitectura no se sostiene por sí sola, incorporar cualquier decisión (no necesariamente en términos de seguridad) será costoso y probablemente saldrá mal.

En el escenario ideal en términos de arquitectura, en este punto se presentó un documento de arquitectura con diferentes tipos de vistas arquitecturales, árboles de utilidad, atributos de calidad, entre muchos otros elementos. Si la arquitectura es aceptable pero no cuenta con este nivel de detalle, es recomendable incorporarlo en este punto para asegurar el conocimiento del sistema en detalle y evitar sorpresas más adelante.

Idealmente, el final de este paso llegará con la claridad de cómo la arquitectura se ajusta a los objetivos de la organización y del sistema.

Paso 4: Identificar las aproximaciones arquitectónicas

El siguiente paso es esencialmente un listado. ¿Qué aproximaciones arquitecturales se tomaron? es necesario enumerarlas de manera exhaustiva, pero la metodología establece que durante este paso no se ejecute ninguna tarea de análisis o interpretación de éstas.

Paso 5: Generar árbol de utilidad

Es necesario organizar las necesidades de todos los actores involucrados de manera armónica, y un mecanismo interesante para lograrlo es el conocido como **árbol de utilidad**. Se trata, al menos en términos conceptuales, de un árbol (a menudo, por facilidad de visualización, se representa mediante una tabla) que tiene como raíz la palabra **Utilidad**.

En el segundo nivel, las ramas del árbol empiezan a surgir, pero siguen siendo toscas. Cada rama representa uno de los grandes atributos de calidad que son considerados para la solución: Seguridad, Escalabilidad, Modificabilidad, etc.

En el tercer nivel, las ramas empiezan a ser más finas, en la medida en que se empiezan a perfilar algunos de los rasgos específicos de cada atributo. En la rama de riesgo operacional, normalmente el tercer nivel es la tríada CIA *(Confidentiality - Integrity - Availability)*. En otras ramas, la división suele ser menos estándar (por ejemplo, de la rama Modificabilidad podrían desprenderse términos como "Modularidad"), pero la lógica en todos los casos es similar.

En este punto, el arquitecto y su grupo de trabajo deben tomar una decisión. En muchos casos, dos niveles de ramas son suficiente granularidad para expresar las necesidades del sistema. En otros, algunas ramas adicionales pueden ser necesarias.

En cualquier caso, el punto crucial del árbol son las hojas. Cada hoja contendrá uno de los atributos de calidad específicos en términos de métricas y dos indicadores: la importancia de garantizar el cumplimiento de dicha métrica y la dificultad para hacerlo. Más adelante se profundizará en este artefacto, pero como idea general: si es fundamental y fácil de implementar, debe impulsarse e implementarse. Si es deseable (i.e. no fundamental) y tiene un costo alto de implementación, es probable que deba descartarse.

Paso 6: Presentación de la metodología

La razón fundamental por la que en el paso 4 se hace especial énfasis en no analizar las

aproximaciones arquitecturales identificadas es que, para ese punto, no existe un marco de referencia contra el cual determinar si las aproximaciones son acertadas o no.

En el paso 5, el árbol de utilidad aparece para ofrecer dicho marco. Una vez establecido, es posible continuar. Y así es exactamente como ATAM continúa: determinar cómo se ve afectada cada hoja del árbol (positiva o negativamente) por cada aproximación arquitectural identificada.

La manera de realizar ese análisis desde la perspectiva ATAM es sencilla y rigurosa. Para empezar, es necesario identificar lo que han bautizado como **puntos de sensibilidad**: elementos específicos que -si fueran removidos o alcanzaran ciertos umbrales- comprometerían alguna de las hojas del árbol de utilidad. Usualmente se podrán encerrar dentro de un círculo en alguna vista arquitectural.

El segundo elemento a identificar es conocido como **puntos de tradeoff**. Se trata de puntos en la arquitectura que al favorecer un atributo de calidad, obstaculizan otro. Por ejemplo, la implementación de un módulo criptográfico para todas las comunicaciones de un sistema puede aumentar significativamente la confidencialidad (e incluso la integridad) de éstas. Por otra parte, implicará un aumento de tiempos de procesamiento y transporte, que puede afectar atributos de calidad relacionados con eficiencia.

ATAM define como riesgos arquitecturales los escenarios en los cuales un punto de sensibilidad alcanza el umbral en el que se compromete el cumplimiento de un escenario de calidad (v.g. una velocidad de procesador demasiado baja para cumplir cierto requerimiento de eficiencia).

Esta etapa culmina con la identificación de todos los puntos de sensibilidad, puntos de tradeoff y riesgos arquitecturales que puedan identificarse en la arquitectura.

Paso 7: Lluvia de ideas y priorización

Es importante que todas las voces involucradas en el problema se hagan oír de manera oportuna. En particular, es importante que sean los usuarios y demás stakeholders del proceso los que conozcan las limitaciones de la arquitectura y tomen las decisiones que resulten pertinentes.

De entrada hay que tenerlo claro: no es posible satisfacer todas las demandas ajustándose a todos los requerimientos (en especial, presupuesto y tiempo son bienes escasos). Aceptando esto, el único camino posible es asegurarse que las decisiones son tomadas por las personas involucradas en el proceso y no por personal técnico/ingenieril que no conozca la dimensión de negocio de las alternativas planteadas.

Paso 8: Analizar las aproximaciones arquitectónicas

El objetivo de esta etapa es menos reiterativo de lo que aparenta en un principio (tiene -de hecho- un nombre similar al de la etapa 6). No se trata de repasar nuevamente las aproximaciones arquitecturales identificadas, sino de alinearlas con las opiniones y puntos de vista que se recolectaron durante la etapa anterior.

No hay mucho que añadir en este punto: una arquitectura no es mejor entre más eficiente, modificable o segura sea (de hecho, como se ha repetido a lo largo de este documento, esos adjetivos tienden a carecer de un significado práctico y medible), sino en la medida en que esté diseñada para favorecer los intereses de los *stakeholders*.

Puesto de otra forma: si una arquitectura que no está correctamente alineada con los objetivos del negocio falla y genera una pérdida millonaria, un daño reputacional o incluso provoca la pérdida de vidas, es una mala arquitectura. Si una arquitectura correctamente alineada con los objetivos del negocio tuviera efectos similares, los equivocados serán dichos objetivos.

Paso 9: Presentación de resultados

ATAM no persigue corregir los puntos débiles de una arquitectura, sino evidenciarlos. De ahí que el objetivo final del proceso sea la comunicación clara y explícita de las actividades realizadas, y en particular de los puntos de sensibilidad, tradeoffs y riesgos (en el sentido ATAM) identificados.

ATAM es una fuente poderosa y flexible de información cualitativa. Es capaz de identificar

los puntos de una arquitectura que ameritan atención especial y las razones por las que se puede ver afectada. Sin embargo, nótese que carece del valor cuantitativo necesario para poder sustentar y presentar las decisiones que se tomen de manera efectiva. Precisamente el método DREAD que se presenta a continuación ofrece un mecanismo cuantitativo de evaluación complementario a ATAM.

6.4.2 *Checklist* de arquitectura y diseño

A continuación se presenta un checklist sugerido de consideraciones de arquitectura y diseño (tomado de [Mar10] y [Mar11]) que = se deben tener en cuenta, si aplican, para el sistema en desarrollo, de tal modo que tengamos un arquitectura que contemple diferentes aspectos de calidad y seguridad. Vemos a continuación las consideraciones de despliegue e infraestructura:

- El diseño identifica, entiende y se acomoda a las políticas de seguridad de la compañía.
- Restricciones impuestos por una infraestructura de seguridad (incluyendo servicio disponibles, protocolos y restricciones de firewall) son identificadas.
- El diseño reconoce y se acomoda a los ambientes impuestos por servicios de hosting.
- El nivel de acceso que se tendrá en el ambiente en que se va desplegar es conocido.
- El diseño identifica los requerimientos de despliegue en la infraestructura y la configuración que debe tener la aplicación en ella.
- Estructuras de dominio, servidores de aplicación remota y servidores de base de datos están identificados.
- El diseño identifica requerimientos de cluster.
- El diseño identifica los puntos que hay que configurar para mantenimiento. Características de comunicación segura que proveen la aplicación y la plataforma son conocidas.
- El diseño tiene en cuenta consideraciones para que se pueda hacer web Farming (incluyendo manejo de sesiones, llaves de cifrado específicas para cada servidor, certificados SSL del servidor etc.).
- El diseño identifica la autoridad certificadora que usará el sitio para soportar SSL.
- El diseño tiene en cuenta la escalabilidad y desempeño que se necesitará la aplicación.

Respecto a manejo de sesiones es importante tener en cuenta:

- Se utiliza SSL para proteger *cookies* de autenticación.
- El contenido de las *cookies* está cifrado.
- El tiempo de sesión es limitado.
- El estado de la sesión está protegido contra acceso no autorizado.
- Los identificadores de sesión no se transmiten usando. *query strings*

Respecto a datos sensibles es importante considerar lo siguiente, lo cual es un mínimo que se recomienda expandir de acuerdo a las necesidades de cada proyecto:

- Los secretos (como claves) no deben ser almacenados, a menos de que sea estrictamente necesario, es importante explorar diferentes alternativas para no hacerlo.
- Secretos no se almacenan en el código. Nunca debe tener una clave “quemada” en el código.
- Las conexiones a base de datos, passwords, llaves, u otros secretos no se deben almacenar en texto claro.
- El diseño identifica el método para almacenar secretos de forma segura (por ejemplo las cadenas de conexión pueden estar cifradas con DPAPI).
- Los datos sensibles no son registrados en el log en texto claro.

- El diseño tiene en cuenta la protección de datos sensible que deben viajar por la red.
- La información sensible no es almacenada en cookies persistentes.
- Los datos sensibles no se transmiten con protocolo GET.

6.5 Plan de Pruebas

El proceso de análisis de arquitectura –hecho correctamente- es exhaustivo y logra prevenir vulnerabilidades que, en caso de explotarse, podrían ser enormemente perjudiciales y para las cuales corregir el rumbo más adelante podría resultar terriblemente costoso.

No obstante, la ejecución de los diseños arquitecturales trae consigo su propia carga de vulnerabilidades. Por esta razón es necesario que, además de la definición de pruebas funcionales (unitarias, de carga, de integración, de sistema, etc.), se incorpore y exija la aprobación de pruebas de seguridad – continuidad - cumplimiento.

En este punto, y los sucesivos, hay que tener presente una máxima absoluta: una falsa sensación de seguridad es mucho más perniciosa que no tener seguridad en lo absoluto. Si el sentido, por ejemplo, de un algoritmo simétrico de cifrado de 128 bits no es del todo comprendido, o si no es clara la utilidad de un algoritmo de *hash*, o si no se sabe cuál es la tarea que cumple un firewall, es posible cometer errores muy graves. Las decisiones tomadas sin total conocimiento o con base en "lo que todos usan" han llevado a más de un ataque vergonzoso en organizaciones del mayor prestigio (tanto a nivel nacional como internacional).

Como en la versión funcional, es posible definir una serie de escenarios de prueba que la aplicación deba superar en términos de riesgo operacional. En general, es una práctica sana que permite a los arquitectos transmitir sus lineamientos y a los desarrolladores tener una idea de qué amenazas su código deberá enfrentar.

Vemos entonces, que las pruebas deben hacerse desde dos puntos de vista, el primero y transparente para los desarrolladores en general, es el funcional. El segundo, es desde el punto de vista de lo inusual o comportamientos fuera del funcionamiento normal del sistema, es decir, pruebas basadas en el análisis de riesgo realizado previamente, patrones de ataques, casos de abuso, mal uso y continuidad. Para estas pruebas es fundamental, al igual que haciendo un análisis de riesgo, se piense como un atacante que quiere aprovecharse del sistema, o en situaciones que son causadas por una eventualidad que normalmente no ocurre. Todas las cosas que se vengan a la mente y que se puedan materializar se deben tener en cuenta, dado que pueden ocurrir, o un atacante las puede hacer ocurrir. Nunca se debe pensar "eso lo se yo porque conozco el sistema muy bien, pero a alguien externo no se le ocurriría". Se deben tratar de cubrir todas las posibilidades, y si bien eventualmente son riesgos que no se pueden eliminar en un 100%, tener la claridad de que existen, para posteriormente, mitigarlos en la mayor medida que se pueda haciendo las modificaciones pertinentes o seleccionando los controles adecuados.

El qué tan rápido realizar estas pruebas es un tema que requiere una reflexión cuidadosa: la evidencia reciente de la altísima productividad que se obtiene con metodologías "lean" y "scrum" y "test-driven development", sugiere que estas pruebas se hagan de manera muy temprana –en el mismo escritorio del programador- y de forma automática con tecnologías de "continuous integration".

Una vez definidas las pruebas de calidad, el equipo de desarrollo empezará a elaborar el código encargado. Las métricas usuales de medida de desempeño (LoC: Lines of Code o casos de uso escrito o puntos funcionales) siguen aplicando como es usual, sólo que el programador deberá tener en mente los requerimientos de calidad definidos en el Plan de Pruebas.

Esta fase de desarrollo se elabora de la forma que la organización haya escogido como método de desarrollo de software: cascada, ágil, XP, Rational o scrum. Lo único que cambia es el alcance, (v.g. si se utiliza scrum, la implementación se realizará por sprints) y quedará listo para pasar a la siguiente actividad.

6.6 Análisis estático de código

Es importante tener presente, que desde el punto de vista de seguridad y riesgo operacional, no siempre es suficiente que unos programas pasen los escenarios predefinidos de pruebas. El programador –dependiendo de su formación y experiencia en código seguro- es posible que utilice construcciones y sentencias de código vulnerables que pese a pasar las pruebas "de calidad" predefinidas pueden llevar a vulnerabilidades no evidentes en su código. Esta es la razón por la que la inspección de código fuente es muy recomendada.

La posibilidad de escribir inadvertidamente sentencias riesgosas es un hecho particularmente cierto en lenguajes de programación como C o C++, en el que el programador tiene la responsabilidad de la gestión de memoria. Se dice que programar en el lenguaje C es el análogo a "correr con un par de tijeras en la mano".

Usualmente se tienen dos aproximaciones a la inspección de código estática. Inspecciones automáticas e inspecciones manuales.

Existen herramientas en el mercado que permiten identificar las fallas de seguridad en programación más comunes. Bien utilizadas, son herramientas poderosas que permiten explorar la totalidad del código de un sistema en tiempo breve e incluso –dependiendo de los procesos de fábrica de software definidos- afinar la habilidad del desarrollador para prevenir su reincidencia en futuros programas que se elaboren.

Es posible encontrar herramientas de todas las escalas de precios, desde la gratuita CodePro Analytix® de Google o Klocwork® de la empresa homónima, hasta Fortify® de HP®.

Existen, como no podría ser de otra forma, dos lunares para estas herramientas poderosas. En ambos casos, la limitación se debe a que se tratan de sistemas automatizados y no de seres racionales.

Por una parte, estas herramientas no detectan vulnerabilidades, sino patrones de código que son comúnmente asociadas a vulnerabilidades o malas prácticas. Esto quiere decir que los resultados deberán ser validados manualmente y que el riesgo de falsos positivos y falsos negativos es latente. Si se configura la herramienta para que sea muy sensible y detecte muchos escenarios vulnerables, la carga de validarlos puede ser abrumadora y des-moralizante para los programadores.

Por otra parte, y por la misma razón, estas herramientas son incapaces de identificar si una entrada o salida de datos corresponde a un uso normal de la aplicación o a algo que puede comprometer la seguridad del sistema.

Fundamentalmente, esto se debe a que "la seguridad del sistema" tiene significados radicalmente distintos para cada aplicación y, en consecuencia, no es posible programar a una herramienta irracional para que identifique estos puntos débiles con precisión.

De esta manera, es aconsejable incorporar el uso de este tipo de herramientas, identificando errores comunes y capacitando a los desarrolladores en la identificación y corrección de malas prácticas de seguridad, pero hay que ser prudentes a la hora de confiar ciegamente en los resultados arrojados, toda vez que los falsos positivos pueden hacer que el equipo de desarrollo termine desestimando los resultados.

Por otro lado, existe la posibilidad también de realizar una inspección manual del código en busca de puntos vulnerables. No es sorprendente afirmar en este punto que es imposible analizar

la totalidad del código de una solución informática con total atención y considerando todos los casos posibles. De hecho, resultará contraproducente plantear una tarea de revisión maratónica. Más de dos horas o 200 líneas de código analizadas en una sola sesión de revisión terminará en el sobreseimiento de más errores de los que se identifican.

Por esta razón, es fundamental seleccionar cuáles serán los componentes esenciales de la solución, y por lo tanto críticos que serán sometidos a este nivel de auditoría manual.

Para proceder con la revisión, es fundamental hacer énfasis en el sentido de la palabra: **el objetivo es revisar el código e identificar puntos débiles**. La tarea de corrección no corresponde al revisor, sino al desarrollador original y busca incorporar un esquema mental "de atacante" al esquema mental generalmente inocente del desarrollador.

Así empezará un procedimiento que requiere algunos cambios en la manera en que las pruebas son abordadas. Hay que superar el modelo de "si doy clic en... e introduzco este valor en... los resultados deberían ser..." y empezar a pensar como un potencial atacante: "si quisiera alterar... debería introducir... en el campo... y luego re-intentar...".

El ejercicio requiere afinar el instinto y asegurarse que los evaluadores conozcan y puedan identificar una amplia variedad de vectores de ataque. No con el objetivo de crear una lista exhaustiva de escenarios porque, precisamente, es la limitación que se intenta superar en este punto. El objetivo es evitar que el desconocimiento técnico de lo que se puede hacer sea el límite para el alcance de las pruebas.

Por ejemplo, si se quisiera validar si es posible iniciar sesión a nombre de otro usuario, la mentalidad de un ingeniero de pruebas sugeriría "probar un usuario que no exista... probar un usuario que exista con una contraseña falsa... probar un nombre de usuario excepcionalmente largo..." o escenarios similares. El ingeniero de pruebas con mentalidad de atacante pensaría en inyección SQL, *buffer overflow* o contraseñas por defecto.

Independientemente de la combinación de revisión manual y automática que se seleccione para cada caso, un reto particularmente importante consiste en la incorporación de estas actividades de revisión de código en los procesos actuales de fábrica de software. Nuevamente los métodos lean – scrum sugieren que estas pruebas se realicen de forma muy temprana –en el mismo escritorio del programador- y de forma automática con tecnologías de "continuous integration".

6.7 Ejecución de pruebas NFR

La ejecución de las pruebas de requerimientos no funcionales, al igual que las demás pruebas, es ideal que se hagan lo más pronto posible después de haber escrito el código, ya que es menos costoso corregir un error que sea encontrado tempranamente[CA11]. Lo ideal sería que se realizaran en la máquina del desarrollador, pero como esto no es posible para todas las pruebas, algunas otras se tendrán que hacer en el servidor de integración continua y otras una vez la aplicación esté desplegada.

En un principio es difícil pensar en pruebas de requerimientos no funcionales que se ejecuten en la máquina del desarrollador, pero en realidad hay varias que se pueden lograr. Por ejemplo si es un sistema que se conecta con un motor de base de datos relacional, un anti-requerimiento podría ser la ejecución de sentencias sql desde un formulario de datos. Se puede programar una prueba unitaria, que se corra cada vez que se compile el programa y que inserte sentencias sql. Lo anterior cumple con el principio lean de intentar encontrar defectos lo más pronto posible, siendo aún requerimientos no funcionales. Debemos ejecutar todas las pruebas que podamos de esta forma. Si encontramos que hay unas que no se pueden ejecutar hasta que no se integre con el desarrollo de otros miembros del equipo, se deben realizar las pruebas tan pronto se haga la integración, que como vimos anteriormente debe ser continua, por lo que el servidor de CI debe

tener la capacidad de ejecutar pruebas automatizadas que corran inmediatamente cuando haya una versión nueva debido al *commit* de un desarrollador.

Existen otras pruebas de requerimientos no funcionales que definitivamente no se pueden automatizar para hacer en el servidor de integración continua o en la máquina del desarrollador, como por ejemplo las de usabilidad. Sin embargo, gracias a que todo el tiempo hay nuevas versiones integradas, tan pronto como una característica esté lista, podrá ser, y debe ser, probada por el área usuaria correspondiente, de tal modo que si se encuentra que tiene problemas de usabilidad, se retro-alimente el equipo de desarrollo rápidamente, para que sea tenido en cuenta el defecto y se pueda actuar de manera temprana, y no desarrollar componentes con una usabilidad similar.

6.8 Pruebas de penetración

Una vez la aplicación haya pasado las pruebas unitarias, de aceptación, de comportamiento, de integración y de calidad, es el momento en el cual en principio el sistema podría pasar tranquilamente a producción.

Sin embargo, la situación no es tan sencilla, todos los escenarios de pruebas planteados hasta este momento son escenarios que no tienen en cuenta la situación de vulnerabilidades de la plataforma tecnológica final en la que va a operar la funcionalidad recién desarrollada.

Es decir, falta todavía un último set de pruebas llamadas pruebas de hacking –ético (o también "pen-test", pruebas black-box o pruebas white-box). Con este repertorio de pruebas se busca analizar la situación de operación de la aplicación en producción (i.e. routers, firewalls, IDS, IPS, DBMS, . . .). Estas pruebas recrean el conjunto de herramientas tecnológicas que utilizaría un atacante real.

Afortunadamente en este tipo de pruebas, y más generalmente en lo que se conoce como seguridad de operaciones, el área de seguridad informática tiene muy amplia experiencia. El día a día de seguridad informática tiene que ver precisamente con estas actividades de identificar vulnerabilidades, parchar software, definir reglas en el firewall, etc., que son particularmente útiles en esta actividad de pruebas de despliegue.

6.9 Análisis de superficie de ataque (para sistemas críticos)

La superficie de ataque en el software, la podemos definir de una forma sencilla, como la suma de los puntos donde un usuario no autorizado puede tratar de insertar o extraer datos en un ambiente de software. Ser consciente y controlar la superficie de ataque, permite mitigar riesgos en seguridad dado que hace más difícil explotar vulnerabilidades y reduce el daño causado cuando alguna logra ser explotada. Dichos puntos por los cuales el atacante intenta entrar, los podemos llamar recursos del sistema (pueden ser, como veremos más adelante, datos escritos en un archivo determinado que el sistema leerá en un futuro, un formulario de consulta, un canal ftp etc). No todos los recursos sin embargo, son parte de la superficie de ataque. Únicamente lo son si estos efectivamente pueden ser utilizados para atacar el sistema. Adicionalmente, hay recursos que tienen mayor probabilidad de querer ser atacados, como por ejemplo, un método que esté corriendo con privilegios de administrador, será más deseable para un atacante, dado que si logra explotar una vulnerabilidad de este, logrará tener acceso al sistema como administrador, a diferencia de si es un método ejecutado con privilegios de un usuario corriente. Cada recurso que sea atacable, hace una contribución a la superficie de ataque, siendo esta la suma de todas las contribuciones de cada uno. Una superficie de ataque grande, no significa que un sistema tenga muchas vulnerabilidades, ni tener pocas vulnerabilidades significa tener una superficie de ataque pequeña. Una superficie grande quiere decir que es más probable que un atacante pueda explotar las vulnerabilidades presentes en el sistema con menos esfuerzo y causar más daño (en

muchas ocasiones, existen vulnerabilidades de los sistemas que son desconocidas, que pueden ser incluso debido al mismo lenguaje de programación, por lo tanto siempre debemos asumir que nuestro sistema tiene vulnerabilidades, así no las conozca nadie hasta el momento, pero puede haber un atacante que las descubra). Por lo anterior, podemos decir que un sistema con una superficie de ataque menor, es menos inseguro que otro con una mayor.

Podríamos pensar (y en algunos artículos de seguridad se indica esto) que a cada recurso se le puede asignar un "peso" según las estadísticas de ataques ya conocidas en sistemas populares, como por ejemplo un S.O. como Windows®. Pero esto es complicado para analizar aplicaciones nuevas, por lo tanto se definirá que cada punto de entrada, tiene el mismo valor de contribución a la superficie de ataque. Por lo tanto cuando se añaden nuevos recursos, la superficie de ataque se incrementa, y esto lo debemos tener en cuenta siempre. Además, como se menciono anteriormente, el daño potencial de un ataque, puede variar según los privilegios del usuario con el que se esté corriendo una funcionalidad, por esto debemos siempre aplicar la ley del mínimo privilegio, es decir, cada usuario o función debe correr con el mínimo privilegio posible.

Ciertamente, entre más grande sea la superficie de ataque, mayor es la probabilidad de que un atacante que quiera penetrar el sistema lo logre hacer. A continuación se describe una forma de hacer análisis de superficie de ataque planteada por OWASP, la cual es enfocada para ser usada por desarrolladores, de tal modo que entiendan y gestionen los riesgos de seguridad de una aplicación mientras diseñan y mantienen esta, al igual que por especialistas en seguridad a la hora de hacer un análisis de riesgos. El objetivo del análisis de la superficie de ataque, es proteger la aplicación de un ataque externo. En este no se tienen en cuenta ataques de ingeniería social.

El análisis de superficie de ataque pretende comprender qué partes del sistema necesitan ser revisadas y probadas para encontrar vulnerabilidades de seguridad. En otras palabras, el objetivo del análisis de superficie de ataque es entender las áreas de riesgo en una aplicación, para hacer a los desarrolladores conscientes de que partes de la aplicación están abiertas a ser atacadas para posteriormente encontrar maneras de minimizar estas. Además, pretende hacer notar cuándo y cómo cambia dicha superficie a medida que se mantiene el software y qué impacto tiene esto desde la perspectiva del riesgo.

Si el análisis es hecho por arquitectos de seguridad o penetration testers, como suele hacerse normalmente, los desarrolladores deben entender y monitorear la superficie de ataque a partir de la fase de diseño en el ciclo de vida del software. Puntualmente la superficie de ataque ayuda a:

- Identificar que partes y funcionalidades del sistema necesitan ser revisadas y probadas para encontrar vulnerabilidades.
- Identificar áreas de código de alto riesgo que requieren protección de seguridad en profundidad (La seguridad en profundidad es un término que viene de ambientes militares. En el software significa simplemente poner más de un control que proteja un activo. Por ejemplo si se quiere proteger información sensible, se puede tener un firewall, un IDS y además la información cifrada, de tal modo que si un atacante logra evitar el primer control, se va encontrar con un segundo e incluso un tercero).
- Identificar cuando la superficie de ataque ha cambiado y se debe realizar nuevamente una valoración de amenazas.

Como la superficie de ataque debe describir todos los puntos donde un atacante puede penetrar un sistema, la podemos resumir en la suma de todos los caminos que reciben o envían datos o comandos, el código que protege estos caminos (incluyendo conexiones con recursos y autenticación, autorización, log de actividades y validación de datos). Se deben tener en cuenta tanto los ataques internos como externos, debido a que un atacante puede ser una persona sin ningún privilegio o alguien en el interior de la organización.

Podemos empezar a construir la superficie de ataque en simples notas y dibujos. Se deben

gastar unas pocas horas revisando el diseño y arquitectura desde la perspectiva de un atacante (siguiendo la filosofía lean que se lleva a lo largo de toda esta consultoría, se debe hacer los mas pronto posible en el ciclo de vida del software para evitar desperdicio, por lo tanto se hace tan pronto se tenga el diseño y arquitectura en la aplicación). Una vez revisados debemos anotar los diferentes puntos donde hay entradas y salidas, vemos a continuación algunos ejemplos para esclarecer la idea:

- Campos de formularios de una interfaz de usuario
- Cabeceras http y cookies
- APIs
- Lectura de archivos
- Consultas a bases de datos
- Emails
- Argumentos en tiempo de ejecución como comandos de consola

El número total de puntos de ataque de una aplicación puede ser del orden de miles. Para poder administrar la seguridad de estos, podemos categorizarlos en diferentes tipos, por ejemplo:

- Puntos de autenticación
- Interfaces de administración
- Consultas y funciones de búsqueda
- Formularios CRUD
- Interfaces o APIs de transacciones
- Interfaces de monitoreo y comandos de operaciones
- Interfaces con otros sistemas

Esto es especialmente útil dado que en la vida real no podemos cubrir todo de una manera perfecta, pero si podemos añadir controles que nos aseguren gran parte de cada categoría, de tal modo que si se incorpora un nuevo componente en el sistema, se definirá en qué categoría se encuentra para protegerlo con los controles correspondientes a esta (si no corresponde a ninguna, se debe crear una nueva, con nuevos controles).

Una vez este mapeada la superficie de ataque, se deben identificar las áreas de alto riesgo. Los puntos de acceso remoto, interfaces que se acceden desde internet y especialmente donde el sistema permite acceso público, son donde normalmente se esperarían más ataques. Entre más componentes de este tipo tengamos, más “atacable” es el sistema Aunque existen diferentes herramientas para protegernos, como por ejemplo firewalls o IDS, es importante que los componentes que no están siendo usados se deshabiliten, es decir, siempre se debe mantener únicamente las características necesarias, dado que dejar las que ya no se utilizan, hacen más grande la superficie de ataque sin aportarnos ningún valor. Por ejemplo, si tenemos un sistema de control de versiones online que contiene los fuentes de una aplicación, pero no esta siendo usado, es importante deshabilitarlo, dado que alguien podría intentar obtener los fuentes y eventualmente lograr violar la seguridad para hacerlo. Siempre hay que reducir la superficie de ataque deshabilitando lo que no se necesita. Cuando tengamos la superficie de ataque identificada, cada vez que se haga un cambio al software se deben hacer las siguientes preguntas:

- ¿Que ha cambiado?
- ¿Qué se está haciendo diferente?
- ¿Qué caminos nuevos se le pueden haber abierto a un atacante?

Cuando en un sistema se incluye por primera vez una página Web, esta agranda significativamente la superficie de ataque del sistema, introduciendo toda clase de riesgos nuevos. Ahora, si por ejemplo se añade un campo nuevo a un formulario de la página, o se añade otra página igual, técnicamente la superficie de ataque crece, pero en realidad el análisis de riesgos sigue siendo similar. Esto no quiere decir que cuando añadamos una página hecha con la misma tecnología no debamos tener cuidado, quiere decir que podemos reutilizar los controles de seguridad que ya teníamos para esta, pero cuando se usa una tecnología que no esta dentro de las categorías en la que agrupamos los puntos de ataque que mencionamos anteriormente, debemos hacer un análisis de riesgos nuevo y determinar qué controles utilizar. Hay componentes en los que se debe tener especial atención al hacer un cambio así sea mínimo. A continuación se muestran algunos que deberían tener especial atención y es útil tener en una lista:

- Manejo de sesiones
- Autenticación/Autorización
- Gestión de contraseñas
- Lógica de control de acceso
- Cambio en la definición de un rol
- Añadir nuevos usuarios administradores
- Cifrado de información sensible
- Validación de datos de entrada
- Cambios en la arquitectura

Todo lo anterior permite tener claridad de cómo un atacante puede llegar a ver un sistema, de tal modo que se elijan los controles pertinentes y los que administran el software sean conscientes de deshabilitar las características innecesarias.

6.10 Mantenimiento y Ajustes

Con alguna frecuencia sucede que los cambios normales a una aplicación que incorporan nueva funcionalidad o que mejoran / corrigen funcionalidad existente pueden introducir problemas de riesgo operacional que la versión original no tenía.

Las razones son variadas... Por ejemplo, es frecuente que en tiempo de desarrollo para agilizar la autenticación de los programadores se relaje la política de contraseñas y puede terminar sucediendo que pase a producción una base de datos cuya contraseña de administrador es “admin” (contraseña por defecto que cualquier persona puede conocer leyendo el manual).

La prevención para este tipo de accidentes es relativamente simple: aún en una aplicación en producción cada vez que se realicen cambios, estos cambios deben pasar todo el set de pruebas original de la aplicación. Es importante considerar el set de pruebas de una aplicación en producción como el componente posiblemente más importante de la documentación de la aplicación.

6.11 Fin de la vida del producto

Finalmente, es importante no perder de vista que aún un sistema en el momento mismo de ser dado de baja en producción, requiere tener presentes algunas consideraciones de riesgo operacional y especialmente seguridad de la información.

Son ya bastante conocidos los casos de computadores “dados de baja” y entregados en donación a “Computadores para educar” que contienen en su disco duro información sensible de la organización.

7. Conclusiones

Se presentó un Marco Teórico bastante completo que resume las principales técnicas y metodologías modernas aplicadas a gestión de riesgo operacional aplicada al ciclo de vida del software.

Estas técnicas aquí descritas van desde conceptos de mejoramiento de procesos y ciclo Deming PDCA, hasta conceptos de madurez en fábrica de software y principios de manufactura Japonesa aplicada a desarrollo de software. Fundamentalmente estas propenden por la integración de esfuerzos de gestión de riesgo operacional en el ciclo de vida del software.

Nótese que estos principios aquí descritos son Universales, en el sentido en que aplica a cualquier organización que maneje cualquier tipo de datos, incluyendo información sensible o los así llamados “datos abiertos”.

Sin embargo, es importante tener presente que la aplicación de estos principios a un proceso específico de fábrica de software en una organización específica, está fuertemente determinado por la misma cultura corporativa.

Solo con los procesos de gestión de cambio adecuados, es que el grupo de ideas presentado en este documento puede ser adoptado e institucionalizado.

Nótese sin embargo que la institucionalización y adopción de estas prácticas es sólo el primer paso en el camino de madurez[CMU10b]. El siguiente paso consiste en la definición de métricas de optimización de procesos. Al respecto hay varias ideas de indicadores y métricas de seguridad en [Jaq07], que sin duda constituyen el siguiente paso en la búsqueda de oportunidades de mejora para un ciclo de vida de software moderno.

Bibliografía

Libros

[15411a] ISO/IEC 15408. *Information Technology - Security techniques - Evaluation criteria for IT security - Part 2: Security functional components*. International Organization for Standardization, 2011 (cited on page 59).

[15411b] ISO/IEC 15408. *Information Technology - Security techniques - Evaluation criteria for IT security - Part 3: Security assurance components*. International Organization for Standardization, 2011 (cited on page 59).

[15414] ISO/IEC 15408. *Information Technology - Security techniques - Evaluation criteria for IT security - Part 1: Introduction and general model*. International Organization for Standardization, 2014 (cited on page 59).

[27016] ISO/IEC 2700:2016. *Information technology – Security techniques – Information security management systems – Overview and vocabulary*. International Organization for Standardization, 2016 (cited on page 11).

[al11] Caralli et al. *CERT Resilience Management Model – A Maturity Model for Managing Operational Resilience*. Addison-Wesley Professional, 2011. ISBN: 000-0321712439 (cited on page 19).

[al12] Jakob Freund et al. *Real-life BPMN - Using BPMN 2.0 To Analyze, Improve and Automata Processing your Company*. Camunda, 2012. ISBN: 1480034983 (cited on page 31).

[Bai12] Scott Bain. *Emergent Design: The Evolutionary Nature of Professional Software Development*. Net Objectives Lean-Agile Series, 2012. ISBN: 978-0321889065 (cited on page 44).

[Bec02] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, 2002. ISBN: 078-5342146530 (cited on page 52).

[Cur09] Mike Sullivan Curt Hibbs Steve Jewett. *The art of lean software Development*. O'Reilly, 2009. ISBN: 978-0596517311 (cited on page 42).

[Cyn14] Aaron Roth Cynthia Dwork. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science, 2014. ISBN: DOI: 10.1561/0400000042 (cited on pages 12, 13).

[Els08] Amr Elssamadisy. *Agile adoption patterns – a roadmap to organization success*. Addison-Wesley Professional, 2008. ISBN: 978-0321514523 (cited on page 43).

[Fea04] Michael Feathers. *Working effectively with legacy code*. Prentice Hall Professional Technical Reference, 2004. ISBN: 007-6092025986 (cited on page 54).

[Hux06] Aldous Huxley. *A Brave New World*. Harper Perennial, 2006. ISBN: 0060850523 (cited on page 13).

[Int11] The Standish Group International. *CHAOS MANIFESTO - The Laws of CHAOS and the CHAOS 100 Best PM Practices - year 2010*. The Standish Group, 2011 (cited on page 39).

[Jaq07] Andrew Jaquith. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007. ISBN: 978-0321349989 (cited on page 79).

[Jav06] Richard Wysk José Manuel Torres Javier Santos. *Improving Production with Lean Thinking*. John Wiley and Sons, 2006. ISBN: 978-0471-75486-2 (cited on page 41).

[Mar10] Lakshmikanth Raghavan Mark S. Merkow. *Secure and Resilient Software Development*. Auerbach Publications, 2010. ISBN: 978-1439826966 (cited on page 71).

[Mar11] Lakshmikanth Raghavan Mark S. Merkow. *Secure and Resilient Software: Requirements, Test Cases, and Testing Methods*. Auerbach Publications, 2011. ISBN: 978-1439866214 (cited on page 71).

[McG06] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006. ISBN: 978-0321356703 (cited on page 62).

[Min16] MinTIC. *GUÍA DE DATOS ABIERTOS EN COLOMBIA*. MinTIC, 2016 (cited on page 13).

[Orw61] George Orwell. *1984*. Signet Classic, 1961. ISBN: 0451524934 (cited on page 13).

[She07] Sheffi. *The Resilient Enterprise – Overcoming vulnerability for competitive advantage*. MIT Press, 2007. ISBN: 978-0262693493 (cited on page 17).

[Sut14] Jeff Sutherland. *Scrum – The art of doing twice the work in half the time*. Crown Business, 2014. ISBN: 978-0385346450 (cited on page 44).

[ver95] Achtergrondstudies en verkenningen. *Privacy-enhancing technologies: The path to anonymity*. Information and Privacy Commissioner/Ontario (1995), 1995. ISBN: 9034632024 (cited on page 15).

Artículos

[CA11] B. Chess and B. Arkin. “Software Security in Practice”. In: *Security Privacy, IEEE* 9.2 (Mar. 2011), pages 89–92. ISSN: 1540-7993 (cited on page 74).

[Dhi11] Danny Dhillon. “Developer-Driven Threat Modeling: Lessons Learned in the Trenches”. In: *IEEE Security and Privacy* 9.4 (July 2011), pages 41–47. ISSN: 1540-7993 (cited on page 64).

[FGJ14] Adila Firdaus, Imran Ghani, and Seung Ryul Jeong. “Secure Feature Driven Development (SFDD) Model for Secure Software Development”. In: *Procedia - Social and Behavioral Sciences* 129.0 (2014). 2nd International Conference on Innovation, Management and Technology Research, pages 546–553. ISSN: 1877-0428 (cited on page 64).

[Hus10] Peter Hustinx. “Privacy by design: delivering the promises”. In: *Identity in the Information Society* 3.2 (Aug. 2010), pages 253–255. ISSN: 1876-0678 (cited on page 15).

[IMJ11] Shareeful Islam, Haralambos Mouratidis, and Jan Jürjens. “A framework to support alignment of secure software engineering with legal regulations”. In: *Software and Systems Modeling* 10.3 (2011), pages 369–394. ISSN: 1619-1366 (cited on page 61).

[Zub15] Shoshana Zuboff. “Big other: surveillance capitalism and the prospects of an information civilization”. In: *Journal of Information Technology* 30.1 (Mar. 2015), pages 75–89. ISSN: 1466-4437 (cited on page 13).

Enlaces Web

[Cam05] Kim Cameron. *The Laws of Identity*. https://msdn.microsoft.com/en-us/library/ms996456.aspx. 2005 (cited on page 14).

[Cav11] Ann Cavoukian. *Privacy by Design - The 7 Foundational Principles*. https://www.ipc.on.ca/wp-content/uploads/Resources/7foundationalprinciples.pdf. [En línea; accesado en Dic 2017]. 2011 (cited on page 14).

[CMU10a] CMU/SEI-2010-TR-032. *CMMI® for Acquisition, Version 1.3*. http://www.sei.cmu.edu/reports/10tr032.pdf. [En línea; accesado en Dic 2017]. 2010 (cited on page 38).

[CMU10b] CMU/SEI-2010-TR-033. *CMMI® for Development, Version 1.3*. http://www.sei.cmu.edu/reports/10tr033.pdf. [En línea; accesado en Dic 2017]. 2010 (cited on pages 38, 79).

[CMU10c] CMU/SEI-2010-TR-034. *CMMI® for Services, Version 1.3*. http://www.sei.cmu.edu/reports/10tr034.pdf. [En línea; accesado en Dic 2017]. 2010 (cited on page 38).

[CD07] ORGANISATION FOR ECONOMIC CO-OPERATION and DEVELOPMENT. *OECD Principles and Guidelines for Access to Research Data from Public Funding*. https://www.oecd.org/sti/sci-tech/38500813.pdf. [En línea; accesado en Dic 2017]. 2007 (cited on page 13).

[Rep14] Congreso de la República. *Ley de Transparencia y del Derecho de Acceso a la Información Pública Nacional*. http://www.secretariasenado.gov.co/senado/basedoc/ley_1712_2014.html. [En línea; accesado en Dic 2017]. 2014 (cited on page 13).

[Ris14] Basel Committe - Operational Risk. *Review of the Principles for the Sound Management of Operational Risk*. http://www.bis.org/list/bcbs/tid_28/index.htm. [En línea; accesado en Dic 2017]. 2014 (cited on pages 18, 19, 55).

[Wik17a] Wikipedia. *Privacy by design — Wikipedia, The Free Encyclopedia*. [En línea; accesado en Dic 2017]. 2017. URL: https://en.wikipedia.org/wiki/Privacy_by_design (cited on page 14).

[Wik17b] Wikipedia. *Value sensitive design — Wikipedia, The Free Encyclopedia*. [En línea; accesado en Dic 2017]. 2017. URL: https://en.wikipedia.org/wiki/Value_sensitive_design (cited on page 14).

Índice