

# **Scrum**

## en Ingeniería de Software

Dario Palminio

2017

Palminio, Dario Andrés

Scrum en ingeniería de software - Dario Andrés Palminio.

- 1a ed. - Córdoba : el autor, 2015.

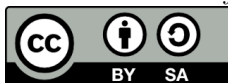
ISBN 978-987-33-8286-4

1. Ingeniería de Software. I. Título.

CDD 005.1

Primera edición, versión 1.0.0, 2015, Córdoba, Argentina.  
Segunda edición, versión 2.0.0, 3 de Abril de 2017, Santiago de Chile.

Los artículos y las ilustraciones de este libro se distribuyen bajo  
una licencia Creative Commons by-sa 4.0



[http://creativecommons.org/licenses/by-sa/4.0/deed.es\\_AR](http://creativecommons.org/licenses/by-sa/4.0/deed.es_AR)

Pueden ser copiados, distribuidos y modificados bajo las  
condiciones de reconocer a los autores y mantener esta licencia  
para las obras derivadas.

Impresión: Autores Editores S.A.S.,  
Bogotá D.C. - Colombia, Buenos Aires - Argentina.

La versión electrónica y el código fuente se publicaron en:  
<https://github.com/dariopalminio/ScrumEnIngenieriaDeSoftware>

Gracias a todos los hombres y mujeres que van a trabajar cada día y llevan con ellos su calor humano; contribuyendo positivamente a su equipo, a su organización y al mundo por ser lo que son, haciendo lo que hacen, cuando lo hacen.



## Prefacio

Quien escribe se propuso ofrecer una guía para el conocimiento e implementación de una manera de trabajar y de gestionar proyectos de Ingeniería de Software llamada Scrum. Se propone explicar al lector el sistema Scrum de un modo integrador desde diferentes perspectivas y fuentes de conocimiento Scrum. De este modo se intenta proporcionar un marco integral que incluya los principios filosóficos, estructura y procesos de Scrum y aspectos complementarios. Se busca ayudar a los equipos en el avance de intentar emplear Scrum, a ejecutarlo correctamente para lograr alcanzar los resultados que aún no se han obtenido. También se pretende brindar un material de apoyo para facilitadores que emprendan el coaching de equipos, equipos de desarrollo de software e interesados en la Ingeniería de Software.



# Índice

<b>1</b>	<b>Introducción</b>	<b>13</b>
1.1	Consideraciones iniciales . . . . .	13
1.1.1	Definición . . . . .	13
1.1.2	Sobre si es una Metodología . . . . .	14
1.1.3	Ámbito de aplicación . . . . .	15
1.1.4	Visión general . . . . .	15
<b>2</b>	<b>Origen</b>	<b>19</b>
2.1	Origen histórico . . . . .	19
2.2	Origen causal . . . . .	21
2.2.1	Problema . . . . .	21
2.2.2	Filosofía criticada . . . . .	22
2.2.3	Metodología criticada . . . . .	25
<b>3</b>	<b>Filosofía Scrum</b>	<b>33</b>
3.1	Mentalidad y modelos mentales . . . . .	34
3.2	Principios . . . . .	35
3.3	Manifiesto Ágil . . . . .	36
3.3.1	Valores del Manifiesto Ágil . . . . .	37
3.3.2	Principios del Manifiesto Ágil . . . . .	38
3.4	Valores de Scrum . . . . .	40
3.5	Principios de Scrum . . . . .	41
3.6	Ideas filosóficas relacionadas . . . . .	45
3.6.1	Emergentismo . . . . .	45
3.6.2	Sinergia . . . . .	45
3.6.3	Ley de Linus . . . . .	46

3.6.4	Sabiduría de multitud . . . . .	47
3.6.5	Arquitectura emergente . . . . .	48
3.6.6	Resiliencia . . . . .	49
3.6.7	Prácticas emergentes . . . . .	50
<b>4</b>	<b>Núcleo del Sistema Scrum</b>	<b>51</b>
4.1	Estructura del Sistema Scrum . . . . .	52
4.2	Sistema de Roles . . . . .	53
4.2.1	Equipo de Desarrollo . . . . .	53
4.2.2	Product Owner . . . . .	55
4.2.3	Scrum Master . . . . .	56
4.3	Proceso Scrum . . . . .	58
4.3.1	Reuniones principales . . . . .	58
4.4	Flujo de artefactos . . . . .	62
4.4.1	Definición de preparado . . . . .	63
4.4.2	Definición de Terminado . . . . .	64
4.5	Reglas y Consideraciones . . . . .	64
<b>5</b>	<b>Gestión de Proyectos</b>	<b>69</b>
5.0.1	Proyecto Scrum . . . . .	69
5.0.2	Planificación . . . . .	69
5.0.3	Triángulo de la Gestión de proyectos . . . . .	71
5.0.4	Planificación de entregables . . . . .	71
5.0.5	Gestión de proyecto orientada a producto . . . . .	73
5.0.6	Gestión de riesgos . . . . .	74
5.0.7	Métricas . . . . .	76
<b>6</b>	<b>Escalamiento</b>	<b>83</b>
6.1	Equipos . . . . .	84
6.1.1	Equipos de características . . . . .	84
6.1.2	Equipos de componentes . . . . .	85
6.1.3	Equipos mixtos . . . . .	87
6.2	Integración . . . . .	87
6.2.1	Scrum de Scrum . . . . .	87
6.2.2	Scrum de Scrum de Scrum . . . . .	88
6.2.3	Comunidades . . . . .	88
6.2.4	DevOps . . . . .	89
6.3	Modelos de escalamiento . . . . .	90



6.3.1	SAFe . . . . .	91
6.3.2	LeSS . . . . .	91
6.3.3	Modelo Nexus . . . . .	91
6.3.4	DAD . . . . .	92
<b>7</b>	<b>Complementos de Scrum</b>	<b>93</b>
7.1	Gamification y Dinámicas De Grupo . . . . .	94
7.2	Técnicas de Justificación de Negocio . . . . .	95
7.2.1	Retorno de la Inversión . . . . .	95
7.2.2	Business Value versus complejidad . . . . .	95
7.3	Técnicas para requerimientos . . . . .	96
7.3.1	Historias de usuarios . . . . .	96
7.3.2	BDD . . . . .	102
7.4	Técnicas de Estimación . . . . .	103
7.4.1	Estimación relativa . . . . .	103
7.4.2	Poker de planeación . . . . .	104
7.5	Técnicas de Comunicación . . . . .	106
7.5.1	Hacer silencio . . . . .	106
7.5.2	Gestión visual . . . . .	106
7.5.3	Tableros Scrum/Kanban . . . . .	107
7.5.4	Tablero de Obstáculos . . . . .	107
7.5.5	Calendario de Obstáculos . . . . .	108
7.5.6	Gráficos de esfuerzo pendiente . . . . .	109
7.5.7	Gráficos de Velocidad . . . . .	111
7.6	Técnicas de Programación . . . . .	112
7.6.1	Técnica Pomodoro . . . . .	112
7.6.2	Programación de a pares . . . . .	113
7.6.3	Revisión de a pares . . . . .	113
7.6.4	TDD . . . . .	114
7.6.5	Integración continua . . . . .	115
7.6.6	Entrega continua . . . . .	115
7.7	Checklists . . . . .	116
7.7.1	Planning Checklist . . . . .	116
7.7.2	Refinement Checklist . . . . .	117
7.7.3	Review Checklist . . . . .	117
7.7.4	Retrospective Checklist . . . . .	119
<b>8</b>	<b>Malas prácticas</b>	<b>121</b>

**9 Conclusión****129****10 Glosario y Acrónimos****131**

# Lista de figuras

1.1	Modelo de dominios de Marco Cynefin . . . . .	16
1.2	Mapa mental sobre Scrum . . . . .	17
2.1	Modelo Cascada de desarrollo . . . . .	26
2.2	Cono de la incertidumbre en contextos inestables .	27
2.3	Metodología de Gestión de Proyectos Clásica PMI	28
2.4	Ciclo de vida en una Gestión de Proyectos tradicional	31
2.5	Metodología Cascada criticada por Scrum . . . . .	32
3.1	Modelo de emergentismo . . . . .	46
3.2	Modelo del espectro del tipo desarrollo de software	49
4.1	Diagrama del Núcleo del Sistema Scrum . . . . .	52
4.2	Diagrama del Sistema de Roles Scrum . . . . .	66
4.3	Diagrama de Flujo de Datos del Proceso Scrum . .	67
4.4	Diagrama de Flujo de Stock de artefactos Scrum .	67
4.5	Diagrama de flujo de estados de un PBI o historia en el flujo de trabajo de desarrollo de software (ejemplo) . . . . .	68
5.1	Proyecto Scrum . . . . .	70
5.2	Triángulo de Gestión de Proyectos Scrum . . . . .	71
5.3	Crecimiento del producto . . . . .	74
5.4	Diagrama de riesgos, problemas e ítems del proyecto (PBIs). . . . .	75
5.5	Estados posibles de un riesgo. . . . .	75

5.6	Medida de satisfacción . . . . .	81
6.1	Esquema de equipos de características . . . . .	85
6.2	Esquema de equipos de componentes . . . . .	86
6.3	Ejemplo de equipos mixtos. . . . .	87
6.4	Ejemplo de comunidades . . . . .	89
6.5	Ejemplo de integración de equipos en DevOps . . . .	90
7.1	Diagramas de Business Value . . . . .	96
7.2	Ejemplo de un tablero Kanban para Scrum. . . . .	107
7.3	Ejemplo de un tablero Scrum. . . . .	108
7.4	Ejemplo de un tablero de Obstáculos. . . . .	108
7.5	Calendario de obstáculos ejemplo. . . . .	109
7.6	Ejemplo de un gráfico Burndown con un sprint de dos semanas y 20 tareas comprometidas. . . . .	109
7.7	Ejemplo de cómo llevar un Burndown con un sprint de 5 días y quedando 3 storypoints por quemar. . . .	110
7.8	Ejemplo de un gráfico Burnup de tareas comprometidas en un sprint de dos semanas. . . . .	111
7.9	Diagrama de Velocidad y Diagrama de Velocidad con Velocidad Promedio. . . . .	112

# Capítulo 1

## Introducción

### 1.1 Consideraciones iniciales

#### 1.1.1 Definición

Scrum es un marco de trabajo (framework) para construir y mantener productos complejos [SBOK, 2013] [Scrum Alliance, 2015]. Scrum funciona como una implementación del ciclo de mejora continua de Deming (PDCA) y como una implementación de los principios ágiles y principios Scrum. Hay que tener en cuenta que Scrum no es exactamente un proceso íntegro, metodología completa o una técnica para construir productos; sino que, es un marco de trabajo dentro del cual se pueden emplear varias técnicas y procesos [Agile Atlas, 2012]. En otras palabras, Scrum es un marco de trabajo que permite organizar a un equipo para que logre cierta cadencia, que es un ritmo sostenible y cíclico de trabajo a través de múltiples iteraciones de trabajo, en el transcurso de un ciclo de vida de un proyecto, en el que el equipo se siente cómodo entregando productos parciales y de valor para el cliente.

### 1.1.2 Sobre si es una Metodología

Hay quienes consideran que Scrum no es una metodología, entre otras cosas porque no especifica exactamente el cómo se hacen las cosas, sino que dice el qué hacer. Sin embargo, hay autores y guías que tratan a Scrum como metodología (por ejemplo la guía SBOK <sup>1</sup>, la ScrumAlliance <sup>2</sup> y Jeff Sutherland<sup>3</sup>). De hecho en el informe original de Ken Schwaber se habla de metodología [Ken Schwaber, 1995]. En consecuencia, se puede encontrar en numerosa bibliografía que el marco de trabajo Scrum puede ser denominado Metodología de Desarrollo Scrum o Metodología de Gestión de Proyectos Scrum. En el primer caso puede deberse a que se puede considerar una metodología como un proceso de desarrollo iterativo e incremental de productos [Ken Schwaber, 1995]. Y en el segundo caso porque se puede considerar que es una alternativa a la gestión clásica de proyectos propuesta por metodologías como la Metodología de Gestión de Proyectos PMI. En este último sentido podemos recordar la definición original de Ken Schwaber: "Scrum es una metodología de gestión, mejora y mantenimiento de un sistema existente o prototipo de producción" [Ken Schwaber, 1995].

Por otro lado, considerando que Scrum define roles, artefactos, actividades, flujo del ciclo de actividades Scrum [Agile Atlas, 2012], reglas y algunas sugerencias de implementación como, además, al definir el flujo del ciclo de Scrum o flujo de trabajo, define parcialmente un cómo, en el cual se incluye una secuencia básica de cosas que hacer; por eso, y sin ser puristas, se puede considerar en forma práctica como una forma de metodología de trabajo y de gestión. O sea que puede funcionar como una metodología a alto nivel o plataforma de trabajo sobre la cual pueden funcionar otras metodologías, más específicas de producción y desarrollo, y otras técnicas y procesos. Por este motivo, puede ser adaptado a diversas empresas y organizaciones que trabajen con metodologías

---

<sup>1</sup>[SBOK, 2013], SCRUMstudy (2015).

<sup>2</sup>"Scrum is the leading agile development methodology", Learn About Scrum, Scrumalliance.org, 2015.

<sup>3</sup>[Jeff Sutherland, 2014b]

diferentes pero compatibles con los lineamientos de Scrum (sus valores y principios) y del Movimiento Ágil (filosofía ágil). Se puede usar Scrum y a su vez utilizar técnicas de otras metodologías para implementar sus actividades y sugerencias. O sea que cuando se usa este marco se hace una aproximación empleando diversas técnicas y, posiblemente, otras metodologías.

### 1.1.3 **Ámbito de aplicación**

Relacionado a su ámbito de aplicación se puede decir que Scrum no es un marco de trabajo orientado a implementarse en cualquier dominio y contexto. Scrum está pensado para proyectos bajo "dominios complejos" [Snowden 2007] donde existe un grado alto de incertidumbre y baja predictibilidad (ver figura 1.1). O sea que es útil en ámbitos con requisitos inciertos y riesgos técnicos altos. Nos permite encontrar prácticas emergentes en dominios complejos, como por ejemplo en la gestión de proyectos de innovación [Martin Alaimo, 2014]. Está orientado a contextos que necesitan niveles altos de creatividad, innovación, interacción y comunicación. Por este motivo, es bastante empleado en la industria de software, ya que en la misma existen contextos específicos de alta complejidad e incertidumbre con necesidad de creatividad e innovación. Pero también se utiliza en otras industrias con dominios de problemas de complejidad semejante. Por ejemplo ha sido empleado en: educación, organizaciones de campañas publicitarias, industria de productos de innovación, empresas de editoriales de libros, etcétera.

### 1.1.4 **Visión general**

En esta metodología se definen los principios y valores a seguir, los roles, relaciones y responsabilidades, los artefactos o entidades manejadas en el proceso de trabajo y un conjunto de reuniones o actividades en un flujo de trabajo. Como resumen podemos ver la imagen de la figura 1.2.

En los siguientes capítulos se explicarán los diferentes aspectos y características de la propuesta de este marco de trabajo y al

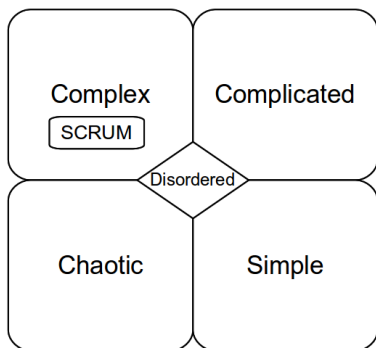


Figura 1.1: Modelo de dominios de Marco Cynefin

final, tras leer el libro, el mapa mental de la figura 1.2 quedará explicado y será fácilmente entendible.



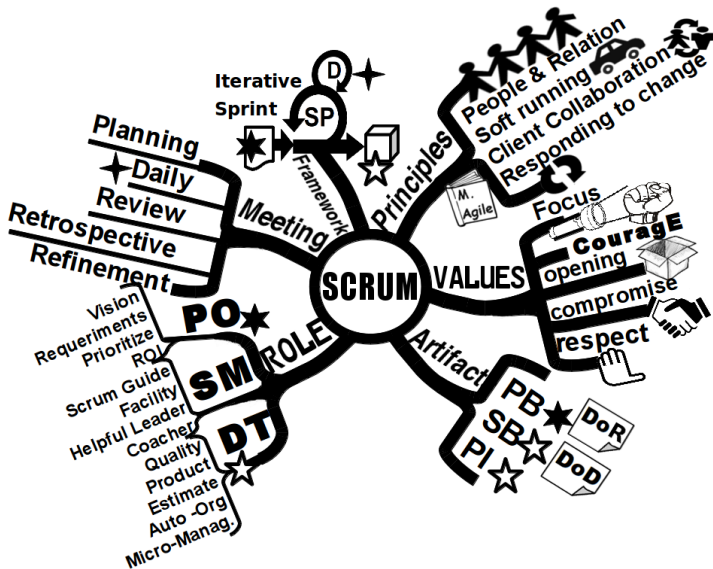


Figura 1.2: Mapa mental sobre Scrum



# Capítulo 2

## Origen

### 2.1 Origen histórico

El origen de Scrum se remonta a la década del 80. La revista gerencial "Harvard Business Review" publica un artículo de Takeuchi y Nonaka denominado: "El nuevo juego para el desarrollo de productos" [Takeuchi/Nonaka, 1986]. En el artículo se describe cómo empresas tales como Honda, Canon y Fuji-Xerox producían nuevos productos a nivel mundial utilizando un enfoque diferente al tradicional de entonces. En ese artículo se introdujo el concepto Scrum, con el término tomado del deporte Rugby, para simbolizar este nuevo enfoque basado en equipos integrales para el desarrollo de productos. Hay autores que denominan a estos conceptos originales de Scrum como "Scrum pragmático" [ScrumManager, 2014].

Una década más tarde, Jeff Sutherland y su equipo en Easel Corporation crearon el proceso de Scrum para ser utilizado en el desarrollo de software, tomando los conceptos del artículo original de Takeuchi y Nonaka. Luego, en 1995, Jeff Sutherland junto con Ken Schwaber publican un informe sobre Scrum en una conferencia de Programación Orientada a Objetos OOPSLA. El informe fue formalizado con el nombre Proceso de Desarrollo SCRUM [Ken Schwaber, 1995]. Hay autores que denominan a este marco de reglas para desarrollo de software como "Scrum

técnico" [ScrumManager, 2014]. Desde esa fecha, Schwaber y Sutherland, han producido y publicado varias especificaciones para Scrum que han servido como guías y material de referencia; como por ejemplo: "Agile software development with Scrum" [Ken Schwaber, 2002] y "The Scrum Guide" [Ken/Jeff, 2013].

Luego, en los 90, Scrum paso a ser reconocidamente parte de las llamadas "Metodologías de Desarrollo de Software de peso liviano", junto a Crystal (1992), Feature Driven Development (1997), Desarrollo de Software Adaptativo (1999) y Extreme Programming (1999). Y luego del Manifiesto Ágil [Beck, 2001], promulgado en 2001 y firmado por Ken y Jeff, pasó a ser reconocido como parte del movimiento ágil y su filosofía. Ken Schwaber fue el primer presidente de la Alianza Ágil fundada tras el Manifiesto Ágil.

Desde que surgió se popularizó en el mundo industrial, principalmente en la industria de software, miles de proyectos en todo el mundo han utilizado Scrum para el desarrollo de productos, tanto en empresas pequeñas (startups) como en multinacionales. Debido a su amplia aplicación surgieron diferentes entidades capacitadoras y certificadoras para difundir Scrum y certificar el conocimiento de quienes rinden los respectivos exámenes necesarios. La Scrum Alliance [Scrum Alliance, 2015] es un ejemplo de este tipo de organizaciones y es considerada la principal organización certificante. La Scrum Alliance fue fundada en 2002 por Ken Schwabe, Mike Cohn y Esther Derby. En 2006, Jeff Sutherland creó su propia compañía llamada Scrum.inc, sin dejar de ofrecer y enseñar a los cursos de Certified Scrum. A su vez, Ken dejó la Alianza Scrum en 2009 para fundar la Scrum.org para mejorar aún más la calidad y la eficacia de Scrum, principalmente a través de la serie Profesional Scrum ("Professional Scrum series").

Hay otras organizaciones capacitadoras y certificadoras que no pertenecen al núcleo Scrum o al Scrum original. Como ScrumStudy, empresa (subsidiaria de VMEdU) que se basa en el libro "SBOK Guide" [SBOK, 2013], pero que está alejada del origen fundacional y de sus autores como Ken Schwaber <sup>1</sup>.

---

<sup>1</sup>[Don Kim, 2016]

## 2.2 Origen causal

Scrum tuvo un origen causal que suele constituir el análisis crítico previo a su explicación. Scrum emerge del intento de resolver un problema y de la crítica a las consideradas causas del mismo. Scrum vino a surgir como una alternativa al modo en que se desarrollaba software y lo hizo como posible solución a los problemas que presentaba la situación en ese entonces (década del 90) de Desarrollo de Software. Por esa razón es que por lo general cuando se habla de Scrum se hace un análisis crítico previo para plantear un cambio debido a un problema. Por un lado, Scrum plantea un cambio paradigmático o filosófico que consiste en un cambio de perspectiva y de mentalidad (mindset). La otra cuestión fundamental es la metodológica en la cual se propone un cambio del proceso de desarrollo de software o proceso industrial de software. Por este motivo, para comenzar a comprender Scrum, hay que comprender cuál es el problema que viene a intentar resolver y qué es lo que se critica.

### 2.2.1 Problema

El problema principal por el que Scrum surgió como alternativa consistió en que la mayoría de los proyectos de desarrollo de software no lograban entregarse en tiempo, dentro de los costos y con las funcionalidades comprometidas. Ya en la primera conferencia organizada por la OTAN (en 1968) sobre desarrollo de software se hablaba de la "Crisis del Software" haciendo mención de los problemas recurrentes en que se veía afectado el desarrollo de software y sus resultados. En ese momento se identificaron entre diversos problemas la baja calidad del software que se desarrollaba, el no cumplimiento de las especificaciones y el código prácticamente inmantenible que dificultaba la gestión y la evolución de los proyectos. Una encuesta de cientos de proyectos de desarrollo de software empresarial indicó que cinco de seis proyectos de software se consideraban no satisfactorios [AntiPatterns, 1998], por otro lado sólo 29 de cada 100 proyectos de IT se terminaban exitosamente y el 71 por ciento de los clientes no estaban satisfechos con los resultados (Standish Groups

Chaos Report 1994 - 2004). En el año 1994 el Standish Group publicó un estudio conocido como el "CHAOS Report" [CHAOS Report, 1994] donde se mostró las siguientes tasas de fracaso en los proyectos de desarrollo de software en general:

- **Proyectos cancelados:** el 31.1 por ciento es cancelado en algún punto durante el desarrollo del mismo.
- **Proyectos insuficientes:** el 52.7 por ciento es entregado con sobrecostos, en forma tardía o con menos funcionalidades de las inicialmente acordadas.

## Causas

Los problemas detectados en los modelos tradicionales se fundamentan principalmente en lo siguiente:

- **Entorno cambiante:** Entorno altamente cambiante propio de la industria [Martin Alaimo, 2014].
- **Dependencia de Procesos rígidos:** el proceso mismo de desarrollo de software donde el resultado depende de la actividad cognitiva de las personas más que de las prácticas y controles de empleados [Martin Alaimo, 2014]. Las metodologías de desarrollo de software resultaron muy pesadas y prohibitivas para responder satisfactoriamente a los cambios de negocio.

### 2.2.2 Filosofía criticada

Como se dijo antes, con Scrum se plantea un cambio paradigmático o filosófico que consiste en un cambio de perspectiva y de mentalidad (mindset). Algunas de las ideas que se critican son las siguientes:

- **Prevalencia de la opinión de expertos:**

La filosofía basada en expertos es una de las criticadas. Normalmente solemos favorecer la opinión de los expertos, pues consideramos que

sólo una persona con experiencia y conocimientos es capaz de emitir juicios verdaderos o correctos en un área o materia en particular [James Surowiecki, 2005]. Pues, es sabido que el juicio de expertos nos puede llevar a malos resultados debido, entre otras cosas, al "mecanismo de autoridad"<sup>2</sup> que hace que respetemos o sigamos las opiniones de los expertos aunque las mismas estén equivocadas.

- **Cultura de Liderazgo:** También se critica la filosofía basada en liderazgo que han generado gestión de control y mando. En el ámbito empresarial y en cierta época, ha prevalecido la filosofía del líder con gestión de control y mando. Podíamos ver una innumerable cantidad de ofertas de cursos de temáticas tales como, por ejemplo, "Masters en Liderazgo" y "Coaching en Liderazgo". Se tenía la idea de que si las organizaciones no eran fuertemente lideradas y los grupos humanos carecían de un líder entonces eran propensas al desastre. En esta perspectiva prevalecía la figura del líder coercitivo y transaccional. El líder coercitivo es el autoritario que se basa en la idea de jerarquía jefe y subordinado, y en la idea de mando y control. El líder transaccional es el que se basa en la motivación de premios y castigos, y también en la idea de negociación constante.
- **Organizaciones centralizadas y jerárquicas:** La idea del líder junto a la idea del control generan, entre otras cosas, la idea de la "jerarquía verticalista en organizaciones centralizadas". Idea que también se rechaza en el marco contextual de Scrum. Entre otras cosas, porque se genera una cultura donde prevalece el mecanismo de autoridad, mencionado anteriormente, y restringe la capacidad de innovación y creatividad. También puede generar sistemas culturales cerrados y rígidos ante situaciones cambiantes que requieren apertura y adaptación. En estas empresas de desarrollo de software, en la que se opera en base a la

---

<sup>2</sup>Mecanismo de autoridad: mecanismo por el cual uno se subordina a la opinión de un líder con autoridad o a un experto por su autoridad. El experimento de Milgram corrobora este fenómeno.

estandarización mecánica, pirámides de mando, estructuras jerárquicas, planificaciones estrictas y herramientas de control con la finalidad de controlar lo que acontece, controlando procesos y manipulando personas según planes, los planes fracasan con probabilidad alta.

- **Cultura de la burocracia:** Las organizaciones centralizadas y jerárquicas pueden generar una cultura de la burocracia en la que se generan cargos y áreas que obstaculizan la ingeniería de requerimientos y en consecuencia al desarrollo de software. Mientras más personas intermedien entre el desarrollador y el cliente, más ruido se puede generar en las comunicaciones, generando así un desfase entre lo que el cliente realmente quiere y lo que se construye. Por ejemplo, si entre el desarrollador y el cliente se encuentran el analista de negocio, el encargado del producto, el ingeniero de aplicaciones y el jefe de ingeniería, es más probable que se forme el teléfono descompuesto en la cadena de comunicaciones y se termine construyendo un producto que no cumple las expectativas del cliente.
- **Evaluación de desempeño individual:** Hace muchos años que una porción importante de los trabajadores, sociólogos y psicólogos viene expresándose en contra de las evaluaciones anuales de desempeño. Sin embargo, esa metodología sigue practicándose en las grandes organizaciones. En esas organizaciones, la cultura competitiva individualista fomenta el éxito individual en vez del colectivo y prevalece la cultura del sistemas de "evaluación de desempeño individual" en vez de grupal<sup>3</sup>. Este tipo de sistema erosiona el trabajo colaborativo y de equipos necesario para trabajar con Agilidad. A pesar de los sistemas de premios y castigos en las organizaciones centralizadas y jerárquicas, los resultados siguen siendo diferentes a lo planificado. Por otro lado, los ciclos de

---

<sup>3</sup>Los métodos tradicionales de gestión de proyectos hacen hincapié en la responsabilidad individual hacia las responsabilidades del proyecto [SBOK, 2013].



retroalimentación para la evaluación de desempeño son ciclos largos que parecen no ser realmente útiles<sup>4</sup>.

### 2.2.3 Metodología criticada

La causa de los problemas y fracasos de los proyectos de software en la situación dada, en ese entonces, de Desarrollo fueron atribuidos principalmente a la Metodología Cascada (Waterfall Methodology) [Ken Schwaber, 1995]. Pero hay que tener en cuenta de qué se habla cuando se critica a la metodología cascada. Pues la metodología cascada no es el modelo cascada y en ocasiones se han atribuido, en un sentido erróneo, las causas de los problemas al modelo cascada. Pues no se puede comparar Scrum con el modelo Cascada porque Scrum no es exactamente un modelo y ofrece soluciones a aspectos que el modelo cascada no ofrece. Aunque sí, el modelo cascada, es parte de lo que se podría considerar una Metodología Cascada.

#### Modelo en Cascada

El Modelo en Cascada (Waterfall Methodology) [Ken Schwaber, 1995] es un Modelo Secuencial de Procesos para la ingeniería de software presentado por Winston Royce en 1970, aunque ya se venía desarrollando desde antes. Para algunos críticos, el Modelo en Cascada, se convirtió en el modelo metodológico más utilizado dentro de la industria en un período de tiempo. Pero hay que considerar que el modelo solo abarca al sistema de producción o sistema de desarrollo (ver "Development Process" en la figura 2.5) de la industria, no al de gestión, y es simplemente un modelo, no una metodología.

---

<sup>4</sup>Las empresas Deloitte y Accenture comenzaron a dismantlar su engorrosa maquinaria de evaluación de desempeño. Pierre Nanterme, CEO de Accenture, dijo al Washington Post: "No estamos seguros de que todo el tiempo empleado en la gestión del rendimiento haya dado grandes resultados", "Una vez al año les digo lo que pienso de ustedes. Eso no tiene sentido. La gente quiere saber... si lo está haciendo bien. Nadie va a esperar un ciclo anual para recibir esa retroalimentación" (Lucy Kellaway, 2015 The Financial Times Ltd.).

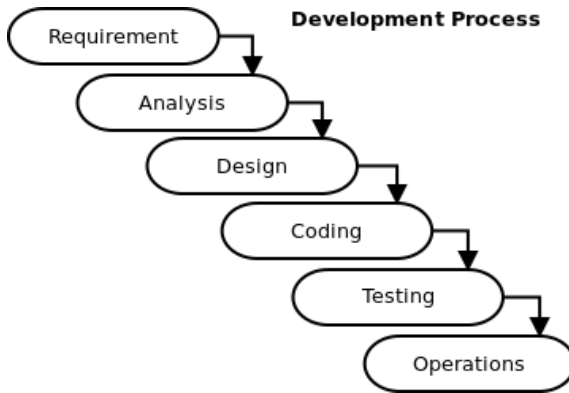


Figura 2.1: Modelo Cascada de desarrollo

En un sentido lato y ortodoxo se puede decir que el Modelo en Cascada refleja un proceso lineal y secuencial de un conjunto de procesos o fases independientes dentro de un proyecto. Las fases son: requerimientos (requerimiento del sistema y requerimientos del software), diseño, codificación, pruebas y operación. Según esto y siempre cuando el proceso de desarrollo de un proyecto conste de solo la secuencia de estas fases sin repetición, la principales críticas como desventajas que se hacen son:

### 1. Previsión:

Si un proyecto tiene una fase de requerimientos única al comienzo, el producto final debe ser anticipado de antemano [Scrum-Institute, 2015] y esto requiere de cierta previsión y certidumbre inicial. La previsión es acorde a una mirada más tradicional y analítica que considera que el diseño se puede planificar en detalle al comienzo del proceso de desarrollo. A este método y práctica se lo llama BDUF o BMUF que significa "Diseño Inicial Grande" o "Gran Modelado al Inicio" y con él se establece la práctica de realización explícita de Diseño Arquitectónico Completo al Inicio [Wiley/Sons, 2002]. Es el mandato técnico de crear modelos integrales de los requisitos para un sistema, el

análisis de esos requisitos, una arquitectura que cumple esos requisitos y, finalmente, un diseño detallado antes de implementar el sistema. Lo que sucede es que cierta previsión y certidumbre inicial y necesaria, en proyectos complejos y cambiantes, es poco factible que suceda. Esto lo muestra el modelo del "cono de incertidumbre"<sup>5</sup> en la industria de software (ver figura 2.2) que indica que en la evolución de la incertidumbre a lo largo de un proyecto el nivel inicial se corresponde a un  $\pm 400$  por ciento y tiende a disminuir a lo largo del proyecto. Por tal motivo preveer el proyecto en su primera parte no es efectivo y tomar decisiones difíciles de cambiar en esa etapa no sería conveniente.

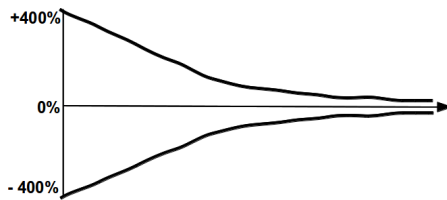


Figura 2.2: Cono de la incertidumbre en contextos inestables

## 2. Requerimientos no necesarios:

Cuando los requerimientos son elicitados al comienzo de un proyecto e implementados posteriormente, podrían nunca ser completamente necesitados por el cliente a partir del final del proyecto [Scrum-Institute, 2015]. O sea que pueden haber requerimientos irrelevantes o innecesariamente implementados, ya sea porque el cliente dejó de tener la necesidad en el transcurso del proyecto o porque la incertidumbre inicial generó una mala elicitación de los mismos.

## 3. Fases separadas:

Cada fase es estrictamente separada [Scrum-Institute, 2015].

---

<sup>5</sup>[McConnell, 2006], [Boehm, 1981], [Martin Alaimo, 2014]

Por ejemplo una vez que se encuentra completa la fase de requerimientos se procede a una firma de aprobación o "sign-off" que congela dichos requerimientos, y es recién aquí cuando se puede iniciar la fase de diseño, fase donde se crea un plano de modelo o "blueprint" del mismo para que, luego, los programadores lo codifiquen, se prosiga con las pruebas y finalmente el despliegue en operación. Pues en entornos altamente cambiante, propio de la industria de software, esta forma secuencial estricta hace del proceso de desarrollo un proceso “pesadas” (estanco y burocrático) y prohibitivo para responder satisfactoriamente a los factores cambiantes de negocio [Martin Alaimo, 2014].

### Gestión de Proyectos Clásica

Bajo el marco Scrum se critica el uso de la gestión de proyectos tradicional o clásica (ver figura 2.3) en proyectos de dominios complejos y con dinámica de requerimientos cambiantes. Se atribuye parte de los fracasos a las siguientes causas:

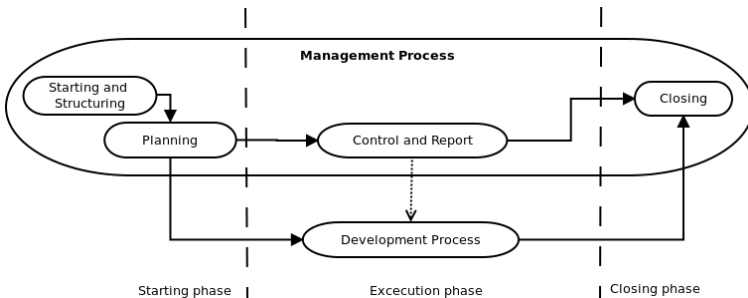


Figura 2.3: Metodología de Gestión de Proyectos Clásica PMI

1. **Planeación predictiva:** La planeación predictiva fue impulsada por el PMI quien ha adoptado el enfoque predictivo y mecanicista para la gestión de proyectos. Planeación predictiva es cuando el alcance del proyecto, el tiempo y el costo requerido para el proyecto se determina

lo antes posible, en la fase de inicio del proyecto, y luego durante la ejecución del proyecto se busca seguir y respetar el plan hasta la fase de cierre (ver figura 2.4). El éxito (o rendimiento) del enfoque predictivo ha sido menos del 50 por ciento (en tiempo, en la fecha y con la funcionalidad deseada) y Scrum se presenta como alternativa para mejorar esta tasa [Ken Schwaber, 2011]. El enfoque predictivo es útil para producción de volumen alto y fabricación de bajo coste. Sus beneficios resultan de reducir la imprevisibilidad del espacio del problema a través de la estandarización y la repetición. La planificación perfecta, la formación y la repetibilidad son las claves. Planificar y luego hacer una y otra vez. La productividad se optimiza a través de procesos de flujo de trabajo perfecto e invariable, y se optimiza el uso de los recursos (personas o máquinas). Pero esta metodología no se ajusta a proyectos donde hay más novedad que repetibilidad y donde las tecnologías, capacidades y creatividad de las personas son cambiantes. En estos casos hay alta probabilidad de que la predictividad fracase [Ken Schwaber, 2011].

2. **Planeación a largo plazo:** Debido a que el proyecto se desarrolla en forma lineal (no iterativo)<sup>6</sup> y que se hace una planificación por adelantado debido a que se considera que los cambios deben ser manejados por un sistema formal de Gestión del Cambio es que se realizan planificaciones a largo plazo. Los proyectos son planificados, en la etapa de inicio de proyecto, para efectuar entregables finales a largo plazo, en el cierre del proyecto. Los cambios que surjan en la fase de ejecución son manejados por un sistema formal de Gestión del Cambio, pero estos cambios suelen tratar de evitarse. No suele ser bien visto la introducción de cambios en la etapa de ejecución y la misma, que involucra la construcción del producto, suele abarcar un período de tiempo grande. Esto hace que se comprometan entregables

---

<sup>6</sup>El modelo lineal de producción y gestión fue propuesta por primera vez por Frederick Taylor en "Principios de Administración Científica", que fue la base de la línea de montaje del modelo T de Ford.

a largo plazo y en forma contractual. Y el compromiso contractual efectuado al inicio del proyecto es difícil de cumplir cuando la realidad del desarrollo no se ajusta a lo planificado debido a la alta tasa de cambios resultado de la incertidumbre y a la variabilidad de las necesidades del cliente en proyectos de software o de dominio complejo. Por otro lado, si se comprueba el retorno de la inversión solo al final del proyecto y el proyecto es prolongado en el tiempo se aumenta la probabilidad de no satisfacer las expectativas de ganancia, no lograr la satisfacción del cliente y estar tarde en la posibilidad de corregir o ajustar para el retorno de la inversión planificado.

### 3. Gestión

**de recursos humanos:** En la planificación tradicional las personas son gestionadas como recursos, hasta cierto punto intercambiables, individuales y especializados que siguen planes. Sin embargo la productividad, la calidad y la creatividad es mucho mayor si la gente que hace el trabajo también lo planea [Ken Schwaber, 2011]. La alta rotación de personal que no deja madurar equipos, el foco en procesos y no en ambientes de trabajo, la evaluación de desempeño individual y no grupal, la priorización de procesos de calidad y herramientas por sobre las personas y relaciones se pueden transformar en un problema.

4. **Gerente de Proyecto:** En el marco de Scrum se considera que el papel del gerente de proyecto es contraproducente en un trabajo complejo y creativo [Ken Schwaber, 2011]. La dirección de un gerente en base a un plan puede limitar la creatividad y la inteligencia del equipo en lugar de incentivarla para resolver mejor los problemas. Si se elimina la figura de gerente de proyecto y se delegan sus actividades de gestión en otros roles, junto con un enfoque de auto-organización, se puede mejorar la productividad y la creatividad.
5. **Gestión Centralizada:** La organización de la gestión tradicional suele ser centralizada. Es centralizada porque

se centra en el gerente de proyectos, en una forma de liderazgo de mando y control y en jerarquías verticalistas. Las estructuras centralizadas, en organizaciones humanas, pueden formar organizaciones mecánicas y rígidas y no permitir la emergencia de la creatividad, de nuevas prácticas y de cambios ágiles adaptativos.

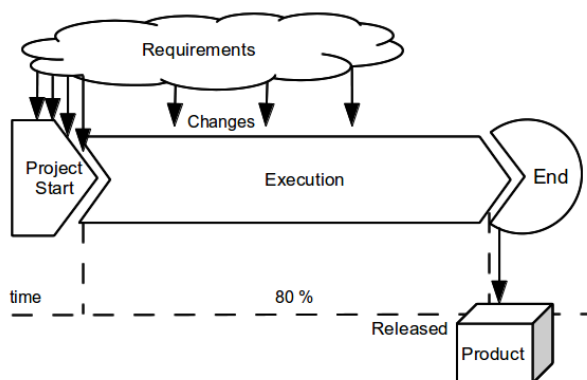


Figura 2.4: Ciclo de vida en una Gestión de Proyectos tradicional

### Metodología en Cascada

La combinación del "Modelo en Cascada" con la "Gestión de Proyectos Clásica" conforma la metodología e idea de desarrollo de proyectos en forma de cascada (ver figura 2.5). La idea se relaciona con una "carrera de relevos secuencial" en la que está incluida la administración del proyecto (proceso de gestión) y la producción del mismo (proceso de desarrollo) en una secuencia prácticamente lineal y en fases secuenciales.

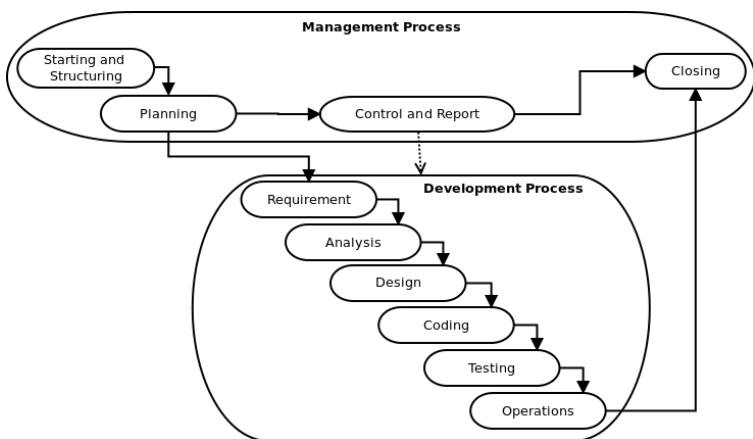


Figura 2.5: Metodología Cascada criticada por Scrum  
(Diagrama integrador del Modelo Cascada  
[Winston Royce, 1970], la Metodología Cascada  
[Ken Schwaber, 1995] y la Metodología de Gestión de Proyectos  
[PMBOK, 1996])



## Capítulo 3

# Filosofía Scrum

Discriminando todas las filosofías generales, como ideologías y concepciones religiosas, podemos decir que hay filosofías relacionadas específicamente al desarrollo de sistemas y de software. Pues, si bien ocurre que ideologías políticas influyen en los desarrolladores, líderes y arquitectos hay ideas más a fines del ámbito de desarrollo de sistemas, ideas que podemos decir que conforman filosofías influenciadoras. Estas influencias son notorias cuando vemos que se toma una decisión de usar una determinada metodología que implementa algunos principios o se siguen determinados principios sin datos empíricos que sostienen su uso ni explicaciones racionales soportadas con evidencia. A veces dichos principios son como sacados de la galera o reflejan expresión de deseos. Tal como sucede cuando se aplican determinadas tecnologías o metodologías como si se tratasen de una moda o de algo aparentemente arbitrario. También esto se puede apreciar cuando escuchamos decir en una reunión de trabajo en equipo que se premiará al individuo que sobresale (filosofía individualista), que “el éxito del grupo está por encima del individual” (filosofía Ubuntu o cooperativa), escuchamos en una reunión técnica que “el software debe ser libre” (filosofía Free Software) o que solo se desarrollara software propietario, que “las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados” [Beck, 2001] (filosofía ágil); o que

“el software es un mundo de objetos” (filosofía del Paradigma Orientado a Objetos). Las consignas o lemas organizacionales, los mantras personales o institucionales, las visiones empresariales y lineamientos institucionales también son, en muchos casos, una expresión de filosofías que condicionan a las personas y al desarrollo de software. Por ejemplo, en particular, se da con las filosofías del pensamiento sistémico, pensamiento crítico, filosofía ágil, software libre, software abierto o software propietario, que son las principales que en el mundo del desarrollo de software influyen a sus actores. Estas filosofías suelen presentar principios a seguir y estos principios guían algunas metodologías (prácticas y métodos) de desarrollo de sistemas y a gran parte de desarrolladores de sistemas. Aquí nos vamos a centrar en la filosofía Scrum y la filosofía Ágil.

### 3.1 Mentalidad y modelos mentales

Las personas que participan en el desarrollo de sistemas, diseños organizacionales o industrias de software tienen experiencias, creencias, principios, vivencias y valores que repercuten y condicionan el modo en que ellas perciben la realidad de su día a día y, en consecuencia, repercuten y condicionan a su actuar, su forma de hacer, su trabajo y el resultado del mismo, sistemas hombre-máquina o software. Estas ideas, en los gerentes y desarrolladores, son modelos mentales que conforman una mentalidad, o lo que se denomina *mindset* (Masa Maeda 2012<sup>1</sup>). O sea que la filosofía que una persona siga o adhiera está relacionada a los modelos mentales de esa persona y forja su mentalidad o *mindset* que en algún sentido lo guía.

---

<sup>1</sup>Masa K Maeda es PhD, Founder and CEO de Valueinnova USA (capacitación Scrum). Es el creador de Serious LeAP, consultor senior del Cutter Consortium en Boston, miembro del comité de dirección del Agile Testing Alliance y maestro en la Universidad de California en Berkeley. Es pionero de Lean y Kanban para trabajo de conocimiento y uno de los formadores del Lean Kanban University. Es una figura líder mundial en Agile y ha generado Serious Games de alto calibre. Previamente hizo I+D para Apple Inc en USA y para Justsystems en Japón. Obtuvo el doctorado y la maestría en Japón.

Los modelos mentales pueden definirse como: “imágenes internas, que están profundamente arraigadas, de cómo funciona el mundo, imágenes que nos limitan a las formas familiares de pensar y actuar”. Los modelos mentales abarcan cuestiones acerca de cómo vemos el mundo y de cómo actuamos en el mundo.

Tener noción de los modelos mentales es importante para entender que hay detrás de las acciones, de las prácticas y de las metodologías usadas. Nos ayuda a comprender que para realizar cambios en las acciones de las personas y cambios en la forma de hacer las cosas es necesario, la mayoría de las veces, cambiar la mentalidad. Sin cambio de mentalidad puede hacerse insostenible en el tiempo un cambio de hábito, práctica o metodología. Por ese motivo, seguir principios en forma mecánica o por obediencia a la autoridad no forma convicción, y seguir una filosofía sin convicción es un primer condicionante de fracaso práctico en su implementación.

## 3.2 Principios

En la industria de sistemas y software los principios son reglas, proposiciones o normas que funcionan como máximas o preceptos que orientan la acción. O sea que un principio es como “una regla general de conducta o comportamiento” [Lawson/Martin, 2008] [SEBoK, 2014]. Un principio es como una recomendación sin el cómo se sigue la recomendación. Cómo hará para seguir la recomendación depende de usted o de quien decida seguir el principio. Por eso son la base, origen y razón fundamental sobre la cual se procede o discurre en materia de sistemas. Se suele usar en el contexto de procesos de desarrollo y metodologías como proposición que da razón, punto de partida o guía, como fundamento de un conjunto de prácticas, una metodología o paradigma de trabajo siendo las metodologías las encargadas de definir el cómo se implementan.

Cada uno sigue algunos principios en su vida como: "trabajo colaborando". También seguimos principios éticos como: “nunca mentir” o “no robar”. A semejanza de los principios éticos tradicionales, ocurre que los principios no necesariamente tienen

fundamento objetivo, racional, empírico o basado en evidencias; pues, en ocasiones se comportan más como principios filosóficos que como principios científicos. Lo cual no quiere decir que no deba buscarse que los principios en ingeniería no sean sacados de la galera o usados de forma irracional, sin justificativo razonable y sin comprobación. Es preferible aplicar principios de comprobada efectividad en su aplicación. El aplicar principios comprobados reduce la cantidad de tiempo necesaria para crear las salidas de planificación de los recursos humanos y mejora la probabilidad de que la planificación sea efectiva [PMBOK, 2004], del mismo modo aplicar principios comprobados en actividades de desarrollo y diseño de sistemas reduce tiempos de investigación para generar la salida deseada y minimiza riesgos, mejorando la probabilidad de que el desarrollo sea efectivo.

Los principios de Scrum son las pautas básicas para aplicar el marco de Scrum y guía a usarse en todos los proyectos Scrum [SBOK, 2013], proyectos en los que se aplica metodología Scrum. Los principios de Scrum se orientan a la gestión de proyectos, desarrollo de productos, trabajo en equipo y el trabajo en base a los principios ágiles. O sea que los principios Scrum tienen correlación con los principios ágiles en forma prácticamente directa [Agile Atlas, 2012]. Y los principios de Scrum están alineados a los valores de Scrum que son: Foco, Coraje, Apertura, Compromiso y Respeto.

A continuación se describen los principios y valores del Manifiesto Ágil y de Scrum.

### 3.3 Manifiesto Ágil

Los principios del desarrollo ágil se encuentran en el Manifiesto por el Desarrollo Ágil de Software o Manifiesto Ágil [Manifiesto Agile 2001], el que expone valores y principios firmado por diecisiete personas convocadas por Kent Beck [Beck, 2001]. Los principios del desarrollo ágil surgieron como principios base y originarios de métodos que estaban surgiendo como alternativa a las metodologías clásica formales (CMMI, SPICE, etc.) a las que, autores como Kent Beck, consideraban excesivamente

pesadas y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas completas y previas al desarrollo [Wiki, 2015].

### 3.3.1 Valores del Manifiesto Ágil

El Manifiesto Ágil propone los siguientes valores:

1. **Individuos e interacciones sobre procesos y herramientas:** hay que priorizar la confianza puesta en los equipos, los individuos dentro de esos equipos y la manera en que éstos interactúan, en vez de seguir rígidamente procesos y herramientas. Pues, son los equipos quienes deben resolver qué hay que hacer, cómo hay que hacerlo y finalmente son ellos quienes lo hacen. Pues los equipos no deberán ser meros autómatas que reciben órdenes jerárquicas de la organización de arriba hacia abajo sino que, en lugar de eso, se espera que de abajo hacia arriba sepan resolver los problemas, ofrecer sus propios métodos de trabajo y ser hasta cierto punto autosuficientes. En este sentido, hay que delegar en ellos la identificación de qué se interpone en el camino de sus metas y la asunción de la responsabilidad de resolver todas las dificultades que se encuentren dentro de su alcance. Se les debe permitir trabajar en conjunto con otras partes de la organización para resolver asuntos que están más allá de su control.
2. **Software funcionando sobre documentación extensiva:** hay que estar orientado al producto o focalizarse en él y, de este modo, requerir un incremento de producto completo y funcionando como resultado final de cada ciclo de trabajo, en vez de tener que cumplir con grandes y engorrosas documentaciones y formalismos burocráticos. Ciertamente, en la construcción del producto, es necesario realizar determinada documentación, pero es el producto concreto o funcionando lo que permite a la organización guiar al proyecto hacia el éxito. Es crucial que los equipos produzcan un incremento de producto en cada ciclo de trabajo.

3. **Colaboración con el cliente sobre negociación contractual:** en vez de tener comunicación pobre debido a restricciones contractuales, el cliente debería ser el punto de contacto principal del equipo, en colaboración de trabajo, con los eventuales usuarios finales del producto y con las partes de la organización que necesitan el producto. Se debería ver al cliente como un miembro del equipo que trabaja colaborativamente con el resto de integrantes para decidir qué debe hacerse y qué no. Con el cliente se debería poder seleccionar el trabajo que debe realizarse a continuación, asegurando que el producto tenga el valor más alto posible en todo momento. Esto es crucial, construir una fuerte colaboración con el cliente en vez de negociar contratos rígidos que obstaculizan el trabajo ágil y generan fricción con el cliente.
4. **Respuesta ante el cambio sobre seguir un plan:** el avance del trabajo o del equipo debería estar representado por un incremento de producto real y que funciona, y no por una correcta correlación y contrastación a un plan. Se debe priorizar la flexibilidad para adaptación a cambios en vez de la rigidez de seguir un plan detallado. Para ello, los equipos deberían inspeccionar lo que sucede de forma abierta y transparente buscando adaptar sus acciones a la realidad. O sea que, la planificación se debería adaptar a la realidad y al equipo y no al revés.

Se puede notar que estos valores son aplicables a cualquier tipo de organización e industria. Pues, solo el segundo valor se refiere particularmente a la industria de software y el mismo se puede reformular de forma más general como sigue: trabajar orientados al producto funcionando más que sobre una amplia y extensa documentación [Sriram Narayan, 2015].

### 3.3.2 Principios del Manifiesto Ágil

Alineados a estos valores, el Manifiesto Ágil propone los siguientes principio:

1. **Entregar valor temprano y continuamente:** Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor (alineado al principio de "entregar lo más rápido posible" de Lean).
2. **Apertura al cambio:** Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. **Entregables frecuentes:** Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible (alineado al principio de "entregar lo más rápido posible" de Lean).
4. **Cooperación con el cliente:** Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto (alineado al principio de "potenciar al equipo" de Lean). O sea que se privilegia una cooperación constante entre miembros del equipo e interesados externos al equipo (como clientes).
5. **Personas motivadas:** Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo (alineado al principio de "entregar lo más rápido posible" de Lean).
6. **Conversación cara a cara:** El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. **Orientación al producto:** El software funcionando es la medida principal de progreso.
8. **Desarrollo sostenible:** Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

9. **Excelencia técnica:** La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. **Simplicidad eficiente:** La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial (alineado al principio de simplicidad). Este estilo de prácticas es parte de algo que, en metodologías ágiles, se conoce como: "hacer todo lo posible por hacer lo menos posible" [Anacleto, 2005]. Por ejemplo en Arquitectura se puede considerar mantener una lo más simple posible. Si la simpleza se traduce en facilidad de uso de la arquitectura, facilidad para entender los conceptos involucrados y documentación necesaria, es muy probable que el nivel de productividad aumente [Anacleto, 2005]. En este sentido la simpleza de la arquitectura es un requerimiento de calidad alineado a la filosofía ágil.
11. **Equipos auto-organizados:** Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados (alineado al principio sistémico de emergencia).
12. **Equipos auto-reflexivos:** A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para la continuación ajustar y perfeccionar su comportamiento en consecuencia.

### 3.4 Valores de Scrum

El corazón de Scrum es el control empírico sostenido por la transparencia, la inspección y la adaptación<sup>2</sup>. Aunque, sin embargo, la Scrum Alliance propuso cinco valores esenciales a seguir<sup>3</sup>:

1. **Foco:** hay que enfocarse en sólo unas pocas cosas a la vez, trabajamos bien juntos y buscando producir un resultado excelente tratando de entregar ítems valiosos en forma pronta.

---

<sup>2</sup>[Ken/Jeff, 2013]

<sup>3</sup>[Scrum Alliance, 2015]



2. **Coraje:** hay que buscar sentirse apoyados y tener más recursos a disposición para promover el coraje para enfrentar desafíos más grandes. Además, para poder lograr cambios significativos en una organización que mantiene una cultura con principios y valores que entran en conflicto con los de Scrum y el Manifiesto Ágil, es necesario tener coraje para impulsar el cambio y plantarse en forma efectiva ante la resistencia al cambio. En este sentido, coraje se refiere al valor que hay que tener para no dejarse dominar ante la idiosincrasia predominante y el “status quo” que atentan contra el pensamiento Scrum.
3. **Apertura:** Hay que tener apertura para expresar cotidianamente cómo nos va, qué problemas encontramos, manifestar las preocupaciones y aceptar las sugerencias de los pares para que éstas puedan ser tomadas en cuenta por nosotros y por los demás. La apertura requiere capacidad de aceptación y tolerancia ante la crítica y la opinión de los demás.
4. **Compromiso:** Se busca lograr compromiso para el éxito gracias a promover el mayor control nuestro sobre lo que hacemos y nuestro destino. Hay mayor probabilidad de lograr compromiso en las personas que deciden sobre lo que hacen.
5. **Respeto:** Buscamos convertirnos en merecedores de respeto a medida que trabajamos juntos, compartiendo éxitos y fracasos, llegando a respetarnos los unos a los otros y ayudándonos mutuamente.

## 3.5 Principios de Scrum

Se suele asociar a los valores del Manifiesto Ágil como principios de Scrum. Por lo que, en primera instancia, los cuatro valores ágiles son los principales principios de Scrum. Pero para no repetirlos en esta sección vamos a nombrar otros seis principios que en diferente bibliografía se suelen atribuir a Scrum. Los mismos son:

1. **Control Empírico:** El proceso empírico de control del proyecto es más efectivo que el control predictivo de largos plazos. Es más efectivo para gestionar la complejidad y obtener el mayor valor posible, basado en inspección y adaptación regular en función de los resultados que se van obteniendo y del propio contexto del proyecto. El proceso empírico permite adaptabilidad a requisitos que emergen del mismo proceso de desarrollo. Con este principio como marco, podemos usar metodologías, prácticas y técnicas, y ser nosotros (o el propio equipo de trabajo) los que a través del empirismo, determinemos la forma más adecuada de hacer las cosas para lograr los objetivos. Son los equipos de desarrollo los que deben hacer lo que sea necesario para entregar el producto esperado y aprender de su propia experiencia mediante exploración y experimentación [UNTREF, 2014]. Es el equipo el que determina qué prácticas y herramientas les dan los mejores resultados, y así mejoran de manera continua. Los buenos equipos trabajarán constantemente en mejorar y aprender de sus experiencia. Además, se debe aprender de la experiencia de los demás leyendo libros y buscando la experiencia de personas que ya hayan venido probando algunas prácticas, o que están experimentando con nuevas potenciales mejores formas de hacer las cosas a través de la inspección y la adaptación.
2. **Auto-organización:** Los equipos auto-organizados pueden auto-gestionarse y de ellos emerge la sabiduría necesaria para la gestión de sus proyectos y actividades, y así lograr la sinergia necesaria para resolver problemas en forma ágil. Esta idea proviene de la concepción de que de un sistema social puede emerger inteligencia de grupo como puede suceder en una bandada de pájaros. Esta es una premisa aceptada en Inteligencia Artificial, pensamiento sistémico y en filosofía emergentista. Se considera que en un sistema con agentes inteligentes, a partir de reglas locales simples puede emerger inteligencia grupal colectiva o de enjambre, pues se considera que la "información local puede conducir a la sabiduría global" [Steven Johnson, 2002]. De aquí

que, de la auto-organización en equipos de trabajo puede surgir inteligencia o también sabiduría según un fenómeno conocido como "sabiduría de multitud" o "wisdom of the crowd" [MIT Press, 2009] en la que la opinión colectiva de un grupo puede ser mejor que la individual de un experto. Este fenómeno, no sólo es útil a la gestión de proyectos, sino también a las actividades de estimación, pues el promedio de muchas estimaciones individuales suele estar mucho más cerca del valor real que la estimación de un experto. En lo referente al diseño se cree, como lo indica el principio 11 (once) del Manifiesto Ágil, que se logran mejores diseños y arquitecturas desde equipos auto-organizados [UNTREF, 2014]. En lo referente al liderazgo se cree que no es necesario un líder jerárquico, autoritario o experto que guíe al equipo sino que es el propio equipo el que genera su liderazgo. Según esta perspectiva se puede prescindir de la figura de líder tradicional o jefe, se puede carecer de líder, lograr muchos líderes o tener un líder con perfil más bien de facilitador (servicial e integrador).

3. **Colaboración:** Se puede entender a la colaboración como la capacidad de reconcebir nuestras propias ideas a la luz de la de las demás [Austin, 2003] para poder lograr ideas colectivas mejores que las ideas individuales [UNTREF, 2014]. Las ideas en colaboración son resultado y mérito del equipo y no de alguno de sus integrantes. La agilidad requiere de colaboración, colaboración interna en el equipo y externa con el cliente. Colaborar con el cliente permite guiar de manera regular los resultados del proyecto de desarrollo. O sea que, la colaboración en el marco de agilidad se orienta directamente a conseguir los objetivos del cliente en un proyecto mediante el trabajo en equipo colaborativo. El trabajo en equipo con colaboración del cliente posibilita su frecuente retroalimentación y mantenerse alineado a su punto de vista y sus expectativas para satisfacer sus necesidades o los requisitos del desarrollo, por el cual el sistema producto se desarrolla con mayor agilidad. La colaboración interna requiere soltura y apertura

para dejar los egos de lado e integrarse en el proceso de pensamiento colectivo, donde los problemas los resuelven todos los del equipo con sus aportes individuales. En el marco de colaboración se dejan de lados los héroes, pues los héroes no ven el gran dragón Malveau 2004 y los héroes se suelen llevar los créditos. La cooperación es la convicción de que nadie llega a la meta si no llegan todos (Virginia Burden).

4. **Priorización por valor:** Se prioriza por valor, es decir que se puede ser más efectivo si se hacen primero las tareas que suman más valor al negocio o a las necesidades del cliente. Se hace necesario prescindir de requisitos de baja prioridad antes que tener que degradar la calidad.
5. **Limitación de tiempos (time-boxing):** El trabajo limitado en periodos de tiempo ayuda en la regularidad en las actividades. Por eso, las iteraciones de trabajo, las actividades y las reuniones deben tener un límite de tiempo y se debe buscar no sobrepasar esos límites.
6. **Desarrollo iterativo:** El desarrollo iterativo permite una construcción gradual en proyectos complejos. En cada iteración el equipo evoluciona el producto (hace una entrega incremental) a partir de los resultados completados en las iteraciones anteriores, añadiendo nuevos objetivos/requisitos o mejorando los que ya fueron completados, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental. El desarrollo iterativo permite gestionar las expectativas del cliente (requisitos desarrollados, velocidad de desarrollo, calidad) de manera regular y lograr reacción, aceptación del mercado y adaptabilidad. Permite que el cliente pueda obtener resultados importantes y útiles ya desde las primeras iteraciones. Facilita la mejora ya que al tener experiencias por períodos de iteración se puede mejorar de las experiencias de iteraciones previas y permite planificar los cambios necesarios para aumentar la productividad y calidad en iteraciones subsiguientes.

## 3.6 Ideas filosóficas relacionadas

### 3.6.1 Emergentismo

Scrum adhiere al principio de auto-organización y está alineado al Manifiesto Ágil y a su filosofía. En la filosofía ágil se toma una idea basada en el emergentismo que la podemos encontrar en el principio: "Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados" [Beck, 2001]. El emergentismo sostiene que el "todo puede ser más que la suma de las partes"<sup>4</sup> que significa que desde un nivel de realidad dado (n) donde componentes se interrelacionan (S1, S2, S3... Sn) pueden emerger sistemas, propiedades o características nuevas en el nivel superior (n+1) que no existen en los componentes individuales, pero que relacionados la generan (ver figura 3.1). En este sentido suele llamarse emergencia al fenómeno en el que algo emerge o es emergente, en el que una totalidad nueva llega a existir, a una cosa nueva que posee una propiedad emergente, a un proceso nuevo que surge de algo, a una estructura que aparece de otras, a un orden que viene de otro, a una novedad cualitativa en la naturaleza que brota, a un sistema que resulta de componentes relacionados o al surgimiento espontáneo de algo nuevo.

En

este sentido la auto-organización sucede cuando componentes, sistemas o personas se organizan sin aparente dirección o mando controlador y generan espontáneamente una forma global de orden o coordinación. Esto se da en una gran variedad de fenómenos físicos, químicos, biológicos, sociales y sistemas cognitivos. A nosotros nos incumbe principalmente lo relacionado a lo social e informático.

### 3.6.2 Sinergia

En lo social se sabe que se pueden lograr equipos de personas con una alta organización y coordinación sin la necesidad de un

---

<sup>4</sup>"The whole is more than the sum of its parts" (Ludwig Von Bertalanffy, General System theory, 1968). Esta idea del sistemismo y el emergentismo también está relacionada al holismo que sostiene la misma afirmación.

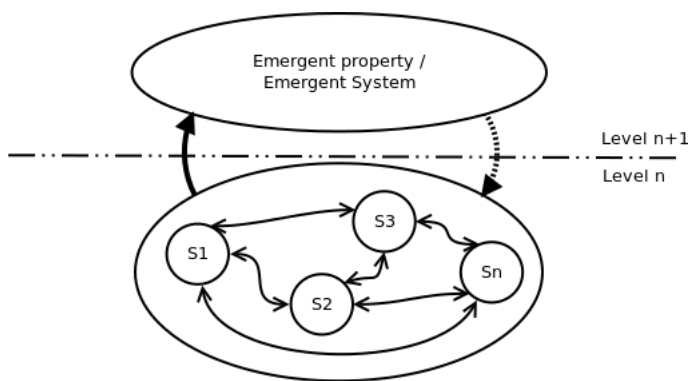


Figura 3.1: Modelo de emergentismo

líder director, sin un líder que ordene y controle, e incluso sin planificación alguna. Pues, no solo organización y coordinación se puede lograr desde equipos auto-organizados, sino que también se puede lograr sinergia. La sinergia es la prestancia extra que puede dar un equipo debido a la acción conjunta de individuos en equipo que es superior a la suma de las individualidades. Otra propiedad emergente que se puede lograr en grupos auto-organizados es la inteligencia colectiva o inteligencia enjambre que incluye a la "Ley de Linus" y a la "Sabiduría de grupo".

### 3.6.3 Ley de Linus

Se pueden formar muchos tipos de inteligencia colectiva en equipos colaborativos. Por ejemplo, lo más común en el desarrollo de software es que problemas extremadamente complejos sólo pueden ser resueltos por un conjunto de personas trabajando conjuntamente en él, a pesar de que muchas veces sea una sola persona la que resuelva un determinado problema, esa persona nunca podría haberlo hecho sola. Es decir que, dada una base suficiente de desarrolladores y testers validadores colaborando, casi cualquier problema puede ser caracterizado rápidamente, y su solución ser obvia [Eric Raymond, 1997]. A esta idea de inteligencia colectiva se la denomina "Ley de Linus".

### 3.6.4 Sabiduría de multitud

La Sabiduría de grupo o de multitudes es una idea, apoyada en evidencias, de que dada "ciertas condiciones" las decisiones o predicciones tomadas colectivamente por un grupo de personas suelen ser más atinadas que las decisiones o predicciones individuales o que las que son tomadas sobre la base del conocimiento de un experto [James Surowiecki, 2005]<sup>5</sup>. Es más, a medida que el grupo es más grande, las decisiones o predicciones son más acertadas.

Claro que para que esto suceda deben cumplirse ciertas condiciones. Para Surowieki (escritor del libro *La Sabiduría de las Multitudes*<sup>6</sup>) las condiciones necesarias son:

1. **Diversidad:** el grupo debe tener diversidad de perfiles para lograr una diversidad de Opiniones. Pues, si todos los del grupo piensan igual o semejante, pertenecen a la misma tribu urbana o subcultura, son de la misma profesión o tienen las mismas características de perfil profesional o psicológico, entonces es menos probable de que logren una sabiduría de grupo.
2. **Independencia:** el grupo no debería estar influenciado y las opiniones de los integrantes tampoco. Si las personas son influenciadas por el grupo se puede caer en lograr el "pensamiento de grupo" y tomar decisiones malas o irracionales. Por otro lado, si el grupo es influenciado por un actor externo al grupo, la decisión grupal es dirigida y, hasta en algún sentido, manipulada.
3. **Agregación:** El sistema de decisión grupal debe ser agregativo, que significa que debe tener la capacidad de sumar o promediar las opiniones individuales. Un sistema de votación no agregativo, por ejemplo por votación de la mayoría sobre una alternativa, puede hacer prevalecer una decisión individual. Un ejemplo de agregación es cuando la

---

<sup>5</sup>"The wisdom of the crowd is the collective opinion of a group of individuals rather than that of a single expert." [MIT Press, 2009]

<sup>6</sup>[James Surowiecki, 2005]

decisión métrica grupal se basa en el promedio de todas la decisiones<sup>7</sup>.

4. **Pericia:** Yo agregaría pericia. Pues si juntamos un montón de filósofos, escritores y médicos a estimar una tarea de desarrollo informático es muy probable que fallen en hacerlo por no tener idoneidad.

### 3.6.5 Arquitectura emergente

Hay dos formas extremas de ver el desarrollo de software incluyendo al diseño. Una es la que sugiere que se pueden prever todas las cientos de miles de cuestiones que emergen cuando se desarrolla el software y, en consecuencia, se trata de limitar las respuestas a las mismas conduciendo un desarrollo planificado, estructurado, dirigido y liderado (el extremo izquierdo del aspecto que se muestra en la figura 3.2) [Neal Ford, 2010]. La otra forma es no anticipar nada, permitiendo que las soluciones software surgan espontáneamente a medida que evoluciona el desarrollo en un proceso no dirigido, no liderado y descentralizado (el extremo derecho del aspecto que se muestra en la figura 3.2). En la primera opción el rol de los líderes y expertos (por ejemplo líderes de proyecto y arquitectos expertos), las reglas globales de dirección, el trabajo de comienzo y la información inicial es fundamental para el desarrollo. En el lado de la segunda opción, la orientada a fenómenos emergentes, el rol de todos los actores, el trabajo en equipo y las reglas locales de trabajo son lo principal para el desarrollo. La emergencia de la arquitectura se da, entre otras cosas, cuando se crea una arquitectura inicial simple y flexible, con decisiones tomadas que no son irreversibles, y con una evolución en el tiempo que considera a los nuevos requisitos y problemas que surjan.

Hay una obra literaria llamada "La catedral y el bazar"<sup>8</sup> escrita por el hacker Eric S. Raymond en 1997 que expone esta idea. La

---

<sup>7</sup>Un ejemplo de Sabiduría de multitud es cuando se le pide a muchas personas que predigan la cantidad de elementos contenidos en un frasco transparente. Es sorprendente como el promedio se acerca considerablemente al número real.

<sup>8</sup>[Eric Raymond, 1997]



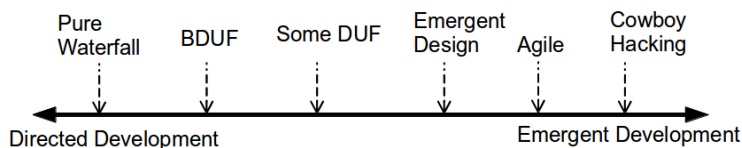


Figura 3.2: Modelo del espectro del tipo desarrollo de software (Espectro que abarca el diseño [Neal Ford, 2010])

misma analiza dos modelos de producción de software: la catedral que representa el modelo de desarrollo más hermético y vertical (el lado izquierdo de la figura 3.2) y por otro lado el bazar, con su dinámica horizontal y bulliciosa (el lado derecho de la figura 3.2).

### 3.6.6 Resiliencia

Generalmente no queremos ser frágiles y vulnerables. Y en trabajo de equipos no queremos equipos frágiles trabajando con procesos o marcos vulnerables que, ante perturbaciones negativas del medio en dominios complejos, tienden a desmoronarse o producir resultados insatisfactorios. Es conveniente todo lo contrario y se puede caer en pensar que es mejor un equipo robusto. Pero sin embargo, mejor que un equipo robusto es un equipo resiliente.

Un aspecto de la filosofía ágil que en muchas ocasiones se hace hincapié, aunque generalmente no se explicita, es el de resiliencia. La resiliencia es la propiedad emergente humana que se expresa como habilidad para surgir de la adversidad, adaptarse, recuperarse y acceder a una vida significativa y productiva<sup>9</sup>. Y es justo esa propiedad la que se intenta lograr emerger de equipos ágiles. Los equipos resilientes son aquellos que "asumen las dificultades como una oportunidad para aprender" y "son flexibles ante los cambios" para lograr respuestas positivas y productivas. Bajo esta perspectiva, un equipo consiste en personas que trabajan bajo presión para hacer todo lo mejor que puedan, manejando los conflictos y teniendo recursos para hacer usos de ellos cuando es necesario. Esta no es una propiedad fácil de lograr en los

---

<sup>9</sup>[OPS OMS, 1998]

equipos, pero se puede condicionar su emergencia logrando que el equipo trabaje con coraje, compromiso, respeto y apertura, basado en relaciones colaborativas y flexibles para tener respuestas positivas ante el cambio. Scrum es el marco de trabajo que permite fallar rápido para aprender de ello y mejorar. Promueve, en su dinámica, el momento de "oportunidad para aprender" y su forma de desarrollo evolutivo permite la "flexibilidad ante los cambios" para no ser frágil, absorbiendo las perturbaciones del entorno y transformándolas en algo positivo.

### 3.6.7 Prácticas emergentes

El marco Scrum se promueve en un dominio de prácticas emergentes. Esto significa que no se puede mecanizar un proceso determinado bajo Scrum para obtener resultados exáctamente predecibles y repetibles en forma estandarizada. Las soluciones no son necesariamente replicables con los mismos resultados, pues sucede que ante los mismos problemas y requerimientos pueden surgir diferentes soluciones. Esto se debe, entre otras cosas, a que las personas no son iguales y los contextos tampoco. Para trabajar bajo Scrum es necesario seguir el núcleo de Scrum, pero implementando diferentes prácticas, técnicas y metodologías, con un grado de experimentación con fallos que intentan ser de bajo impacto. Para lograr una prestancia alta usando Scrum son necesarios niveles altos de creatividad, innovación, interacción y comunicación. En vez de determinar un proceso completo y estructurado, se intenta desarrollar un proceso flexible donde el equipo es quien va generando, sobre la marcha de un proyecto, el proceso propio, de forma tal que el mismo emerge de un contexto relacional e iterativo de inspección y adaptación constante. Son los equipos quienes van encontrando las mejores maneras de resolver los problemas con prácticas emergentes. El conocimiento surge a partir de la acción experimental en prácticas emergentes, rediseñándonos continuamente en busca de una mejor manera, emergente desde lo empírico, de hacer las cosas, sin pretender controlarlas anticipadamente sino, más bien, esculpiéndolas en la práctica constante e iterativa.

## Capítulo 4

# Núcleo del Sistema Scrum

Scrum es un sistema compuesto por la filosofía Scrum (SCRUM Philosophy), como parte del sistema cultural, y tres conjuntos de componentes interrelacionados que forman el "Núcleo del Sistema Scrum"<sup>1</sup> (ver figura 4.1). Los tres conjuntos de componentes son: roles, actividades y artefactos. Los roles conforman un sistema de roles que determina las responsabilidades de los actores integrantes, sus relaciones recíprocas, sus restricciones y la relación con los demás componentes. Las actividades conforman la parte principal del sistema de procesos Scrum (Proceso Scrum) que determina las relaciones de organización entre actividades, roles y artefactos. Y los artefactos conforman el conjunto de almacenes o componentes de trabajo. Todo el sistema busca asegurar una cadencia de trabajo que consta de una regularidad basada en iteraciones de tiempo fijo y ceremonias o actividades regulares que permiten la repetibilidad rítmica (predictibilidad del flujo de trabajo) bajo un grado de prestancia.

A continuación se describirán estos componentes y sus

---

<sup>1</sup>El núcleo de Scrum es determinado esencialmente por los organismos y autores que mantienen el scrum originario, o sea los autores (Ken y Jeff) y scrum.org mediante el libro Scrum Guide 2013[Ken/Jeff, 2013].

relaciones.

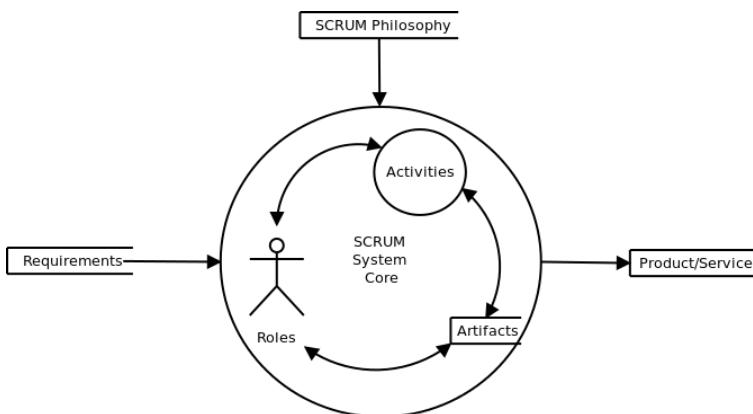


Figura 4.1: Diagrama del Núcleo del Sistema Scrum

## 4.1 Estructura del Sistema Scrum

Scrum está pensado como un sistema de trabajo para equipos con tres roles principales: el Product Owner, el Scrum Master y el Development Team.

Cuando los integrantes del sistema Scrum ejercen los roles mencionados en un flujo de trabajo o proceso Scrum, trabajan sobre tres artefactos esenciales: el Product Backlog (lo que queda por hacer), el Sprint Backlog (lo que se va a hacer) y el Incremento de Producto (lo que logramos hacer). Estos artefactos son tratados en un flujo de trabajo en el cual se construyen productos en forma incremental, en una serie de ciclos cortos de tiempo llamados Sprints.

En cada ciclo Sprint del flujo de trabajo se practican seis actividades Scrum: refinamiento de producto, planificación, reunión diaria o Daily, desarrollo, revisión de producto y retrospectiva. La actividad de refinamiento de producto (Refinement) no suele tener un nombre unificado; pues, se la

suele llamar "Backlog Grooming"<sup>2</sup> (aunque no se aconseja usar la palabra grooming), Mantenimiento de Backlog o Refinamiento (Refinement). Salvo el desarrollo, las actividades se consideran reuniones o ceremonias Scrum. El desarrollo no es una reunión Scrum ya que constituye la actividad de producción del producto o servicio. O sea que es donde se produce el incremento de producto.

## 4.2 Sistema de Roles

El Sistema de Roles (ver figura 4.2), en el núcleo de Scrum, es el conjunto de roles y relaciones parte del sistema Scrum. Como se mencionó anteriormente hay tres roles principales: el Product Owner o dueño del producto, el Scrum Master o facilitador y el Equipo de Desarrollo o Equipo a secas (Scrum Development Team, Miembros del Equipo de Desarrollo o Desarrolladores Scrum). También hay roles secundarios como el de Stakeholder, Vendedores y Cuerpo de Asesoramiento de Scrum. El rol Stakeholder es el más importante de los roles secundarios e incluye a los clientes, usuarios y patrocinadores<sup>3</sup>.

Hay que tener en cuenta que Scrum contempla solo estos tres roles principales como núcleo en el equipo Scrum y cuando se implementa en forma ortodoxa son los únicos roles permitidos. En el caso del Equipo de Desarrollo, cada integrante puede tener diferentes perfiles, características o roles especializados, pero bajo este marco sólo tienen el rol de Desarrollador. Cuando Scrum se integra con otras metodologías o esquemas de roles los desarrolladores pueden cumplir otros roles que funcionan como sub-roles.

A continuación se listan los diferentes roles principales:

### 4.2.1 Equipo de Desarrollo

El Equipo de Desarrollo es parte del Equipo Scrum y son los responsables de construir, o sea desarrollar el producto.

---

<sup>2</sup>No se aconseja usar el término Grooming debido a que según el diccionario Oxford English Dictionary tiene connotaciones sexuales.

<sup>3</sup>[SBOK, 2013]

Para el cumplimiento del rol Equipo de Desarrollo o desarrollador se deben tener en cuenta las siguientes afirmaciones:

- Crea y gestiona el desarrollo de software comprometiéndose a fechas de entregas estimadas por el equipo.
- Es responsable de la calidad técnica del producto.
- Debe buscar su desarrollo profesional para lograr excelencia técnica. Para ello debe, además, mantener el foco en el aprendizaje, la innovación y el mantenimiento de una actitud de mejora continua.
- Debe evitar trabajar en múltiples proyectos teniendo asignación cien por ciento en el proyecto.
- Debe priorizar y promover la comunicación cara a cara.
- Provee estimaciones de ítems de trabajo.
- Es responsable de gestionar su propio trabajo durante el Sprint.
- Buscar encontrar una cadencia sostenible para la entrega de incrementos potencialmente entregable de productos.
- No debe esperar a que le asignen tareas ya que debe seguir la forma "pull" de asignación que consiste en tomar tareas por sí mismo o por consenso de todo el equipo.
- Monitorea el progreso y el éxito con el resto de los roles del Scrum Team.
- No debe ocultar impedimentos ni información.
- No debe hacer tareas de otro rol Scrum. En el caso de una implementación ortodoxa no tiene otro rol que el de desarrollador.
- Debe procurar ser multidisciplinario aprendiendo del conocimientos de los compañeros, aplicando sus habilidades en diferentes tipos de tareas y no especializándose en una sola cosa donde se trabaja en forma estanco (cerrada y hermética).

- Debe procurar la estabilidad del equipo y su cadencia.

### 4.2.2 Product Owner

El "Product Owner" cumple la función de dueño del producto en el proyecto y es el responsable del negocio. O sea que, es el "Responsable del Producto" o Servicio, quien vela por que el equipo tenga una visión clara y una estrategia, de qué es lo que se va a hacer, para la creación de ese producto o servicio.

Para el cumplimiento del rol Product Owner se deben tener en cuenta las siguientes afirmaciones:

- Debe colaborar con el Scrum Master y con el Equipo de Desarrollo.
- Es quien determina el mejor producto a conseguir.
- Es quien provee las "hipótesis requerimientos"<sup>4</sup>, por lo que es responsable de entenderlas, escribirlas y transmitirlas en forma eficaz.
- Es dueño y responsable por el artefacto Product Backlog.
- Es responsable de desarrollar una visión, comunicarla y promoverla.
- Es responsable del retorno de la inversión ROI y de los resultados empresariales, evaluando continuamente el impacto en el negocio.
- Es recomendable que esté asignado a un solo Scrum Team con un porcentaje de asignación mayor o igual a un 70 por ciento.
- Asegura la Colaboración efectiva y la participación de los Stakeholders en el proyecto, administrando sus expectativas y manteniendo una comunicación regular con ellos.

---

<sup>4</sup>Bajo el marco Scrum los requerimientos inicialmente constan de hipótesis a ser evaluadas. Son hipótesis porque son dinámicas, no son requerimientos finales sino que son deseos del cliente a ser refinados hasta convertirse en verdaderamente requerimientos.

- Monitorea el progreso y el éxito con el resto de los roles del Scrum Team.
- No estima ni provee estimaciones.
- No gestiona el presupuesto de todo el proyecto pero puede gestionar el presupuesto del equipo.
- No es jefe ni gerente de proyecto.

### 4.2.3 Scrum Master

El Scrum Master es un "Facilitador" y "Coach del Equipo", que es guardián del marco de trabajo Scrum y responsable de mejorar el flujo de valor hacia el cliente. Que sea guardián del marco de trabajo significa que es quien debe asegurar que se siga el sistema Scrum, concientizar sobre la filosofía seguida bajo el mismo, capacitar y entrenar al equipo de desarrollo y facilitar la resolución de impedimentos relacionados a la implementación de Scrum.

Para el cumplimiento del rol Scrum Master se deben tener en cuenta las siguientes afirmaciones:

- Es el responsable de que se siga Scrum. Brinda apoyo a las prácticas de Scrum.
- Necesita desempeñar su rol con coraje.
- Debe gestionar los riesgos y problemas junto con los demás roles.
- Debe buscar remover obstáculos e impedimentos.
- Es responsable de buscar lograr la mejora continua del proceso o sistema de trabajo.
- Busca conseguir que se haga el trabajo.
- La mejor manera de cumplir el rol es estar tiempo completo en este rol (asignación 100 por ciento).



- Sirve de embajador del equipo ante la organización, obrando en ocasiones como mensajero, representante o interfaz.
- Debe ser guardián del equipo protegiéndolo de perturbaciones externas negativas.
- Monitorea el progreso y el éxito con el resto de los roles del Scrum Team.
- Ayuda al Equipo a encontrar una cadencia sostenible para la entrega de incrementos potencialmente entregable de productos.
- Busca lograr un entorno seguro y de apoyo que genere confianza y respeto mutuo. Ayuda a lidiar con los conflictos personales.
- Busca establecer acuerdos claros y que se cumpla lo pactado.
- Colabora con el proceso de aprendizaje del equipo.
- Considera diferentes formas de trabajar con el equipo Scrum buscando aplicar las más adecuadas según el contexto.
- Busca comprender la esencia de la comunicación en los diálogos del equipo Scrum y ayudar a concretar sus acciones y su entendimiento.
- La presencia en la Daily Scrum no es obligatoria, pero es aconsejable que esté para facilitarla y moderarla cuando sea necesario.
- No debe estimar junto al Equipo de Desarrollo ni hacer tareas de otro rol como, por ejemplo, desarrollar (programar, probar, etc.).
- No es jefe, no es gerente de proyecto, no debe gestionar al Equipo de Desarrollo y no es responsable por la planificación del proyecto.
- No asigna tareas, sino que procura que el equipo se las auto-asigne o las tome por voluntad propia y auto-organización.

## 4.3 Proceso Scrum

En el proceso Scrum o sistema de flujo de actividades Scrum (ver figura 4.3) los miembros del equipo Scrum colaboran para crear una serie de Incrementos de Producto durante iteraciones de intervalos fijos de tiempo denominados Sprints. En cada iteración Sprint, se comienza por una Planificación del Sprint para producir un Backlog del Sprint a partir del Backlog de Producto, es decir un plan para el Sprint. El equipo se auto-organiza para realizar el Desarrollo, mediante reuniones Diarias de Scrum para coordinar y asegurarse de estar produciendo el mejor Incremento de Producto posible en el proceso de desarrollo del producto o servicio. Cada incremento satisface el criterio de aceptación del Product Owner y la Definición de Hecho o "Definition of Done" compartida por el equipo para satisfacer el criterio de tarea terminada. Junto al proceso de desarrollo se hace también un Refinamiento del Backlog ("Refinement") para prepararse para la reunión de planificación del próximo Sprint. Finalizando cada ciclo se termina el Sprint con una reunión de Revisión del Sprint y luego una reunión Retrospectiva del Sprint, revisando el producto y su proceso con una perspectiva crítica y de mejora continua.

### 4.3.1 Reuniones principales

- **Planificación:** es una actividad o una conversación de duración fija al principio de cada Sprint para decidir sobre lo que se terminará y se demostrará en la revisión.

Esta reunión se divide generalmente en tres partes principales: una primera parte es estratégica relacionada al "qué", una segunda parte táctica relacionada al "cómo" y una tercera relacionada al acuerdo de cierre.

- **Planificación relacionada al "Qué":** La primera parte se propone responder a: ¿Qué trabajo será realizado? En esta parte se desarrolla la definición de lo que se necesita hacer, de cuáles hipótesis del Cliente se desean desarrollar. En este proceso se crean los ítems de Product Backlog o PBIs y los Criterio de

Aceptación. Los PBIs son generalmente escritos por el Product Owner y están diseñados para asegurar que las hipótesis de requisitos del Cliente estén claramente representados y puedan ser plenamente comprendidos por todos los Stakeholders y los desarrolladores del Equipo. En la dinámica de la reunión el Product Owner cuenta cuáles son los PBIs disponibles, que pasan el criterio de completitud o DoR (una definición de listo o refinado), para ser desarrollados en el Sprint y los explica para que sean comprendidos por el Equipo. Mientras sucede esto los integrantes del Equipo hacen todas las preguntas que consideren necesarias para comprender los detalles de lo que se desea realizar y puedan, así, entregar una estimación del trabajo a comprometer. Luego de estimar se procede a una negociación entre el Product Owner y el Equipo de cuáles son los PBIs que el Equipo se compromete a desarrollar para transformar en un incremento de producto potencialmente entregable. En este proceso el Scrum Master se encarga de facilitar la ceremonia, moderar y tratar de asegurar de que todos los Stakeholder del proyecto que sean necesarios para aclarar detalles estén presentes o sean contactados para hacer las respectivas aclaraciones. El resultado de este proceso es un conjunto de PBIs estimados y comprometidos inicialmente para ser trabajados en el Sprint.

- **Planificación relacionada al "Cómo":** En la segunda parte de la planificación se propone responder a: ¿Cómo será realizado el trabajo? Esta parte es táctica y por lo tanto más técnica por lo que no es necesaria la presencia del Product Owner, pero debe estar disponible para contestar preguntas y clarificar dudas surgidas sobre la marcha. En esta reunión el Equipo discute cómo implementará los PBIs, diseñando inicialmente, en forma general y abstracta (acuerdo de alto nivel), las soluciones y definiendo tareas

implicadas.

- **Cierre de planificación como "Acuerdo":** Cuando termina la reunión relacionada al "Cómo", el Equipo debe negociar y comprometer finalmente el alcance del Sprint formando un acuerdo de compromiso con el Product Owner. El resultado de este proceso es un conjunto de PBIs que forman el alcance del Sprint, o sea el Sprint Backlog, el objetivo del Sprint y una visión de diseño o arquitectura a alto nivel de lo que se desea implementar junto con un conjunto de tareas planificadas para el Sprint.
- **Scrum Diario:** es una actividad o una reunión diaria obligatoria del Equipo Scrum, en el lugar de trabajo, con una duración fija, que sirve para coordinación y organización mediante una retroalimentación del estado de actividades de cada integrante del Equipo. Permite identificar impedimentos bloqueantes, actualizar artefactos, revisar el Sprint Backlog, ayudar a disminuir riesgos e identificar personas que pueden servir de ayuda a determinadas tareas. En esta reunión los miembros del equipo se reúnen, de pie, para informar de sus progresos en el Sprint y planificar las actividades del día. Para ello proporcionan respuestas a tres preguntas:
  - ¿Qué hice ayer?
  - ¿Qué voy a hacer hoy?
  - Si tengo obstáculos: ¿Qué impedimentos tengo?

Es obligatorio que el Equipo asista a esta reunión y es aconsejable que esté el Scrum Master para facilitarla y servir de moderador. El PO puede asistir también para seguir el avance del trabajo durante la iteración, pero puede participar como oyente o no asistir. Sin embargo, si se quiere lograr un mejor trabajo de equipo y mayor integración con el PO, es aconsejable que siga la dinámica al igual que el Equipo.

- **Refinamiento:** es una actividad o reunión para refinar la lista de PBIs o Product Backlog que consiste en trabajar sobre las hipótesis de requerimientos, definir las y añadirles detalles, granularizarlas, estimarlas y priorizarlas. Es un proceso continuo que hace el Product Owner, pero formalmente como reunión se desarrolla con participación del Equipo en colaboración para examinar y revisar los elementos del Product Backlog. En esta reunión la responsabilidad recae en el Product Owner y el Equipo solo colabora.

En la reunión que se hace con el equipo se desarrollan las siguientes actividades:

- El PO presenta los próximos ítems de backlog al equipo.
  - Se conversa grupalmente aspectos sobre ítems de backlog:
    - \* Conversación, entendimiento y ajuste.
    - \* Búsqueda de posibilidad de granularización.
    - \* Identificación de dependencias.
    - \* Detección de riesgos que pueden hacer que esas historias no se completen e identificación de actividades a realizar para mitigarlos.
  - Estimación.
  - Priorización.
- **Revisión:** es una actividad o una conversación de duración fija al final de cada Sprint para dar retroalimentación sobre el avance del producto. En esta reunión se evalúa el incremento funcional potencialmente entregable construido por el Equipo en el proceso de desarrollo. Para lograr hacer esto el Equipo de Desarrollo junto al Product Owner y los Stakeholder involucrados, revisan los resultados funcionales y operativos (producto utilizable) del Sprint. El objetivo es recibir una retroalimentación de lo construido y aprobar o rechazar los PBIs que pasaron la DoD y son potencialmente entregables, para lo que los Stakeholder prueban el producto construido y proveen su feedback. En este proceso pueden

haber cambios o nuevas hipótesis de requisitos que surjan para agregarse en el Product Backlog.

- **Retrospectiva:** es una actividad o una conversación de duración fija al final de cada Sprint para que el Equipo reflexione y busque mejoras procedimentales. En esta reunión se intentan responder tres preguntas relacionadas al "proceso": ¿qué se hizo mal?; ¿qué se hizo bien? y ¿que se puede mejorar? De esta reunión deberían quedar lecciones aprendidas y un listado de acciones a tomar para mejorar la forma de trabajar. Las acciones a tomar deben ser desarrolladas en el siguiente Sprint y analizadas en la retrospectiva del mismo. En esta reunión es obligatoria la asistencia del Equipo con el SM y la asistencia del PO es optativa, sin embargo, podría asistir si el equipo lo considera apropiado <sup>5</sup>.

## 4.4 Flujo de artefactos

Los artefactos ítems de trabajo fluyen desde que se definen en el Backlog de Producto hasta que se transforman en incremento de producto. Los ítems de trabajo son items de valor para el cliente que comienzan su nacimiento como ítems de backlog o PBIs del Backlog de Producto. Debido a que el dueño del Backlog de Producto es el Product Owner, es él quien crea los PBIs, ya sea por trabajo individual o con la colaboración del Equipo de Desarrollo. Los PBIs son requerimientos que puede escribirse de diferente manera. Es común escribir los PBIs en forma de Historias de Usuario [Cohn, 2004], pero no es un requisito de Scrum. Estos PBIs son refinados por la actividad de Refinamiento de Backlog hecha por el Product Owner y el Equipo de Desarrollo mientras se practica el desarrollo de un Sprint y son priorizados por el ProductOwner. En la reunión de planificación "Planning" se toman PBIs que cumplan el criterio de aceptación o "Definition of Ready" (2 "Selection") para ser incluidos en el "Sprint Backlog" y el Equipo de Desarrollo pueda trabajar en

---

<sup>5</sup>[Martin Alaimo, 2014]

ellos en el Sprint. A medida que el Equipo de Desarrollo termina un ítem "Sprint Backlog" cumpliendo el criterio de aceptación "Definition of Done" (3 "increment") se genera un Incremento de Producto candidato potencialmente entregable (Potentially Shippable Product Increment). Luego en la Revisión se acepta el Incremento de Producto candidato pasando a ser efectivamente un Incremento de Producto listo para ser entregado o desplegado, para "Release" (4 "releasing"). En caso de no ser aprobado (5 "Rejection") pasa nuevamente al Product Backlog para ser tenido en cuenta en el próximo Sprint. Los ítems de "Sprint Backlog" que no se lograron terminar también vuelven (6 "comeback") al "Product Backlog". Esto ocurre formalmente en la revisión [Martin Alaimo, 2014].

#### 4.4.1 Definición de preparado

La definición de preparado, "Definition of Ready" o DoR es un tipo de Definición de Terminado particular (previo al desarrollo) que se corresponde con las condiciones para que un PBI pueda pasar a formar parte de un Sprint Backlog. Si un PBI no cumple con su DoR no puede ser tomado en una planificación para ser ítem de trabajo del Sprint planeado, ya que es un ítem que no está suficientemente preparado para ser comprometido para el desarrollo. O sea que, la definición de preparado permite entonces tener un punto de acuerdo entre el Product Owner y el Equipo, permitiendo conocer cuándo una historia de usuario está realmente lista para evaluar su factibilidad de desarrollo en una reunión de planeamiento y ser llevada a un Sprint.

Un ejemplo de DoR puede ser el que sigue:

1. Historia de usuario definida.
2. Cumple el criterio INVEST (INVEST es tratado posteriormente).
3. Hay un contexto de negocio claro.
4. Criterios de aceptación definidos, claros y comprobables.

5. Identificación de dependencias con otras historias de usuario.
6. Hay una idea clara de cómo se puede demostrar la historia en una review.
7. El equipo tiene el conocimiento para analizarla.

#### 4.4.2 Definición de Terminado

Los ítems de artefactos fluyen por sus diferentes estados (o Stock en el flujo de Stock) a medida que cumplen con determinadas condiciones de cambio de estado. A estas condiciones que deben cumplir para cambiar de estado se las llama definición de terminado y funcionan como válvulas para que los ítems pasen de un Stock a otro Stock. Por ejemplo para que un ítem pase del Stock de Sprint Backlog a Incremento de Producto Potencialmente Entregable debe cumplir una definición de terminado, "Definition of Done" o DoD. Normalmente se suele considerar que el DoD es una lista de actividades que cada elemento de trabajo debe completar para poder ser considerado potencialmente entregable para un cliente dado y conforma un "entendimiento compartido de lo que significa que el trabajo esté completado"<sup>6</sup>.

### 4.5 Reglas y Consideraciones

Además se establecen algunas consideraciones relacionadas a tiempos y tamaños. Se aconseja un tamaño de equipo de desarrollo mayor a 4 miembros y menor a diez ( $7 \pm 2$ ), o sea entre cinco y nueve. Por debajo de este número mínimo se tiene un equipo pobre que puede brindar un producto pobre y por sobre ese número máximo se aumenta la complejidad de gestión y coordinación del equipo disminuyendo el funcionamiento apropiado tras la metodología planteada. También se recomienda una duración de Sprint no mayor a un mes [Ken/Jeff, 2013]. La Reunión de Planificación de Sprint debería tener un máximo de duración de ocho horas para un Sprint de un mes. El Scrum Diario

---

<sup>6</sup>[Ken/Jeff, 2013]



o "daily" es una reunión con un bloque de tiempo de 15 minutos para que el Equipo de Desarrollo sincronice sus actividades y cree un plan para el día. Por ejemplo, una daily mayor de 15 minutos se puede considerar larga y en 15 minutos es complicado que más de 15 personas puedan exponer lo que hicieron, lo que harán y si tienen bloqueos. Por eso se aconseja, como máximo, nueve personas en el equipo de desarrollo. En lo que concierne a la Revisión, el tiempo estipulado es de cuatro horas para Sprints de un mes o dos horas para uno de una quincena. La reunión de Retrospectiva debería estar restringida a un bloque de tiempo de tres horas para Sprints de un mes o a una hora y media para los de una quincena. Las reuniones de Planificación, Revisión y Retrospectiva deberían ser proporcionales a la duración del Sprint.

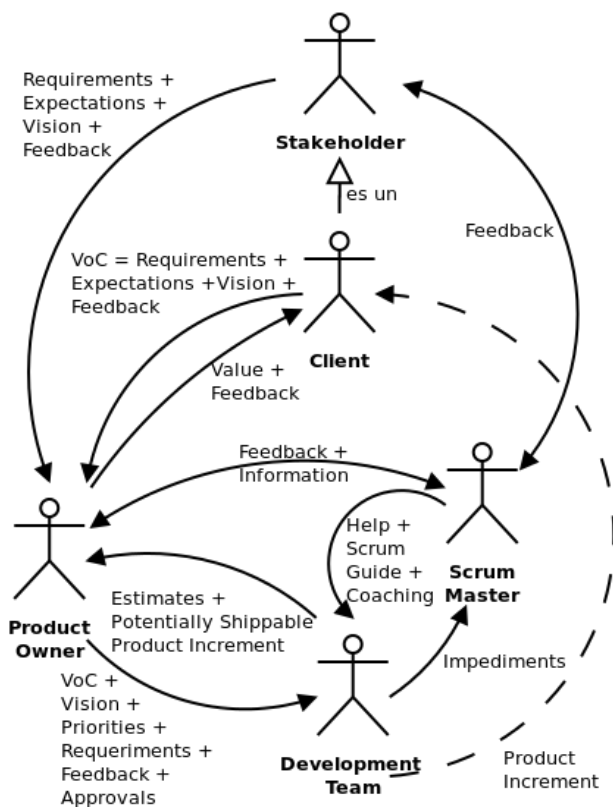


Figura 4.2: Diagrama del Sistema de Roles Scrum

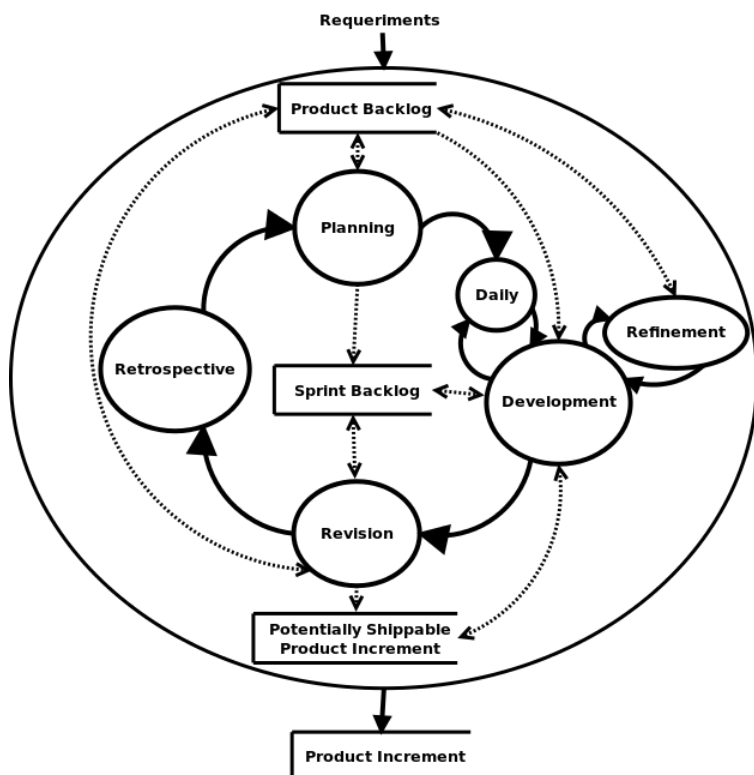


Figura 4.3: Diagrama de Flujo de Datos del Proceso Scrum

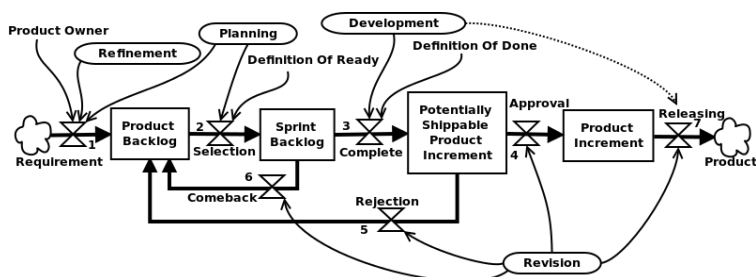


Figura 4.4: Diagrama de Flujo de Stock de artefactos Scrum

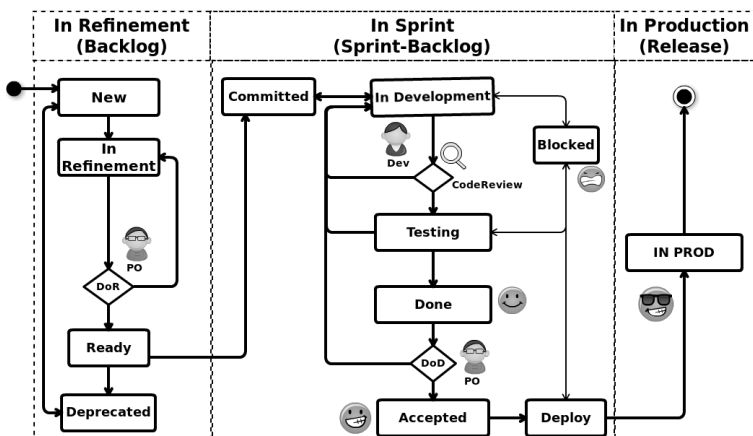


Figura 4.5: Diagrama de flujo de estados de un PBI o historia en el flujo de trabajo de desarrollo de software (ejemplo)

# Capítulo 5

## Gestión de Proyectos

Scrum puede verse como un marco ágil de gestión de proyectos para el desarrollo incremental de productos, valiéndose de equipos autoorganizados.

### 5.0.1 Proyecto Scrum

Un proyecto, en ingeniería de software, es un esfuerzo temporal que se lleva a cabo para crear un sistema, software o resultado único<sup>1</sup>. Los proyectos son organizados, en una empresa u organización, por el proceso de administración de proyectos. Según este proceso, el ciclo de vida de los proyectos se puede dividir en tres fases: inicio, ejecución y cierre (ver figura 2.4).

### 5.0.2 Planificación

En la administración de proyectos es necesario planificar y dicha actividad se suele hacer en la fase de inicio ("Starting phase" o "Project Start"). Pero, a diferencia de la metodología clásica (ver figura 2.4) en que la planificación estaba siempre al inicio y el desarrollo en la fase de ejecución, en el marco Scrum la planificación se distribuye durante todo el ciclo de vida del proyecto y en la fase de ejecución se hace el desarrollo incremental

---

<sup>1</sup>Se parafrasea la definición del PMBOOK [PMBOK, 2004].

de productos (por incrementos de producto) en iteraciones cortas (Sprint) donde cada iteración tiene su respectiva planificación (ver figura 5.1) y su incremento de producto, en caso de haberlo logrado.

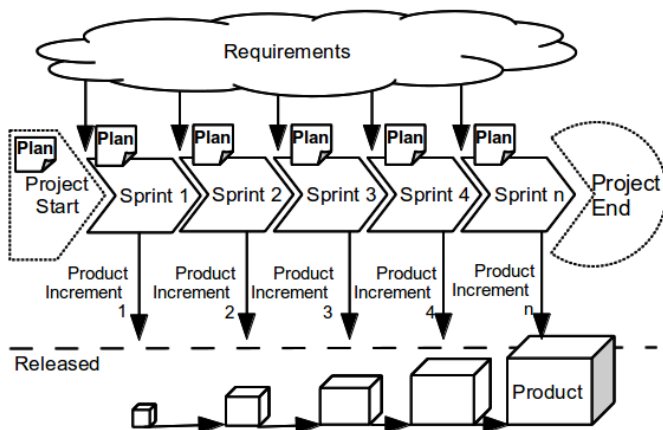


Figura 5.1: Proyecto Scrum

Entonces podemos decir que en Scrum se piensa en muchos planes periódicos (a corto plazo). Los mismo pueden estar en un plan mayor a largo plazo pero de carácter flexible. También se puede realizar un plan global de entregables en base a los incrementos de producto estimado. Pero, desde esta perspectiva, hay que considerar que aunque se trabaje con planificaciones, los planes no son contratos a respetar a rajatabla.

### Inicio de proyecto

En la etapa de inicio de proyecto ("Project Start") se puede hacer una "inception", aunque esta técnica no es parte de Scrum. En una inception se hace un plan versátil alto nivel a corto plazo (un roadmap o release plan) para poder guiar e iniciar el desarrollo bajo el esquema Scrum.

### 5.0.3 Triángulo de la Gestión de proyectos

El marco de Scrum cambia el triángulo clásico de la gestión de proyectos. El compromiso ya no es entre el tiempo, presupuesto y calidad; sino que se basa en el triángulo de: Presupuesto (Costo), Tiempo y funcionalidad (alcance) (ver figura 5.2). Además, tradicionalmente se ha intentado fijar el alcance para negociar y variar el presupuesto y el tiempo. En cambio, desde la agilidad, se intenta mantener fijos el tiempo y el presupuesto mientras se varía el alcance<sup>2</sup>. La pregunta habitual es: ¿cuánto puedes hacer en este tiempo (bajo un costo fijo)?

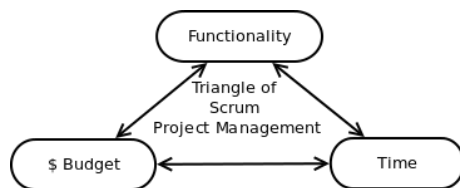


Figura 5.2: Triángulo de Gestión de Proyectos Scrum

### 5.0.4 Planificación de entregables

Con este marco de trabajo tampoco es necesario hacer una entrega final (o "releasing") ya que se pueden hacer entregas paulatinas. En cada sprint se puede entregar valor. Para hacer entregas intermedias planificadas se puede crear un plan de muy alto nivel para múltiples Sprints durante una planificación de lanzamientos (por ejemplo en una inception) llamado "Release Plan". Este plan de entregables o de de lanzamientos es una guía con la que se pretende reflejar las expectativas sobre qué funcionalidad se implementará y cuándo, aproximadamente, se completará <sup>3</sup>. También sirve como una base para monitorear el progreso dentro del proyecto. Pero siempre hay que considerar que no es un plan equivalente a un plan clásico, los ítos de releases no deberían ser compromisos rígidos y contractuales y, además, el desarrollo

<sup>2</sup>[Martin Alaimo, 2014]

<sup>3</sup>[Scrum-Institute, 2015]

del proyecto no debería centrarse en respetar el plan. Por este motivo el plan de lanzamiento no es un plan estático; pues, se cambia durante todo el proyecto cuando nuevos requerimientos o conocimientos están disponible y, por ejemplo, cuando entradas en el Scrum Product Backlog cambian y se re-estiman. Por lo tanto, este plan debe ser revisado y actualizado en intervalos regulares, por ejemplo, después de cada Sprint.

### Release Plan Ágil

Un "Release Plan" ágil es, normalmente, un conjunto de historias de usuario (o épicas) agrupadas por "releases" o versiones del producto que se ponen a disposición de los usuarios <sup>4</sup>. En otras palabras, es una planificación a media distancia como una proyección hacia adelante en una serie de sprints <sup>5</sup>. Esta planificación es algo valiosa de hacer cuando se usa el marco Scrum, pero no es requerido por el "núcleo Scrum" o el "Scrum originario" <sup>6</sup>. Se puede utilizar Scrum con éxito sin necesidad de utilizar "Release Planning".

Hay que remarcar que bajo el marco Scrum, si se hace un Release Plan, debería ser un documento minimalista (buscando el principio de simplicidad), pensado para MVPs (producto de mínimo valor) o entregas frecuentes (respetando el valor de software funcionando), abierto a modificaciones constantes (adaptabilidad y desarrollo evolutivo), consensuado con el equipo (transparencia) y desarrollado por el PO en colaboración con el cliente y con el equipo de desarrollo (priorizando la conversación y no la relación contractual).

Luego hay que tener en cuenta que para crearlo se deben tener disponibles las siguientes cosas:

1. Un Product Backlog priorizado y estimado.

---

<sup>4</sup>Release plan, jmbeas, 2011; Agile Estimating and Planning, Mike Cohn

<sup>5</sup>Release Planning, Retiring the Term but not the Technique, Mike Cohn, 2012.

<sup>6</sup>Gone are Release Planning and the Release Burndown, Ralph Jocham and Henk Jan Huizer in Community Publications, Scrum.org, Saturday, October 01, 2011; Ken Schwaber and Jeff Sutherland Release Updated Scrum Guide, David Bulkin, Infoq.com on Jul 27, 2011



2. La velocidad estimada del Equipo Scrum.
3. Las condiciones de satisfacción (metas para la agenda, el alcance, los recursos)

### 5.0.5 Gestión de proyecto orientada a producto

El verdadero objetivo de Scrum es conseguir un producto mínimo viable o MVP en manos de los futuros clientes cuán rápido sea posible y obtener sus comentarios como feedback temprano<sup>7</sup>. MVP es una estrategia para el aprendizaje de forma iterativa sobre sus clientes para poner a prueba las hipótesis fundamentales del negocio<sup>8</sup>. Es un incremento de producto, subconjunto del 20% de features que representan parte del 80% de valor<sup>9</sup>. Las features del MVP representan conceptualmente al producto final completo y se prueba con un grupo de usuarios "early adopters". Las pruebas deben ser medibles y se hacen sobre las hipótesis fundamentales del producto como negocio.

Por otro lado hay otro concepto que se maneja en el ámbito de la agilidad, el MMP (Minimal Marketable Product). El MMP es el producto con el más pequeño posible conjunto de features y que crea la experiencia del usuario deseada, y por lo tanto puede ser comercializado y vendido, tempranamente, con éxito.

Resumidamente, la gestión de proyecto se orienta al producto y su ciclo de vida. Un producto puede evolucionar mediante una secuencia de releases, de los cuales algunos entregan valor directamente de cara al público. Después de una etapa de creatividad y prototipado (si el producto es nuevo y/o novedoso), tendremos uno o una secuencia de entregables MVPs hasta llegar a un MMP, que será el mínimo producto que se lanza al mercado como tal. De allí en adelante comienza el crecimiento en escalamiento (Growth) e incremento de funcionalidad hasta alcanzar una madurez (maturity), para finalmente constituir un producto final estable o commodity (ver fig. 5.3)<sup>10</sup>.

---

<sup>7</sup>[Jeff Sutherland, 2014b]

<sup>8</sup>[Greg Gehrlich, 2012]

<sup>9</sup>[Jeff Sutherland, 2014b]

<sup>10</sup>[Greg Gehrlich, 2012]

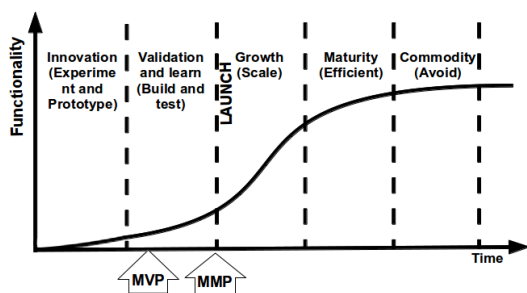


Figura 5.3: Crecimiento del producto

### 5.0.6 Gestión de riesgos

Si bien la gestión de riesgos no es parte de scrum y un facilitador no se encarga de la gestión al modo tradicional como la gestión de riesgos, sí debería velar por mitigar los problemas que surjan en el proyecto y apoyar, en consecuencia, a la gestión de riesgos. En este sentido, no solo se encargaría de ayudar a desbloquear problemas y reducir impedimentos para que el sistema de trabajo fluya, sino que también puede preocuparse de prever issues para anticiparse a los problemas y controlar así, de antemano y en la medida de lo posible, los riesgos asociados a bloqueos del flujo de trabajo que atentan contra los objetivos del proyecto. Lo difícil es lograr una gestión de riesgos ágil en vez de una gestión pesada, difícil y que demande mucho esfuerzo de gestión. En esta vía, es preferible mantener un simple registro de riesgos con información concisa. Los datos principales a registrar de un riesgo, además de su nombre, pueden ser: descripción, probabilidad (probability), impacto (severity), criticidad (criticality), acciones de mitigación, dueño y estado.

Algo simple que se puede lograr trabajando con criticidad es usar tres valores en la probabilidad de ocurrencia y en el impacto (1, 2 y 3). El impacto representa la severidad de la ocurrencia de un problema asociado al riesgo o el tiempo perdido (size of loss). Por otro lado, la criticidad representa la prioridad del riesgo o el grado en que ese riesgo afecta negativamente al

proyecto (exposure). El mismo será resultado del producto entre la probabilidad y el impacto. En consecuencia, la criticidad podría ser: 1, 2, 3, 4, 6, 9.

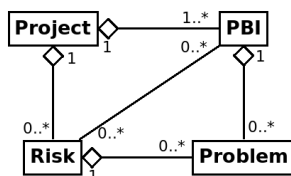


Figura 5.4: Diagrama de riesgos, problemas e ítems del proyecto (PBIs).

Las herramientas gráficas que se pueden usar para dar visibilidad de los riesgos son: a) la “matriz de riesgos”; b) el “histórico de criticidad”; c) o el gráfico de curva de riesgo quemado (Risk Burn-down Chart).

La gestión de riesgos no es independiente de la gestión de impedimento o de problemas (issues o problem). Muchos de los problemas surgidos en el proyecto están asociados a riesgos identificados. Por tal motivo, la gestión de riesgos es útil, porque podemos mitigar los problemas de antemano. Por dicha razón, puede ser valioso llevar un registro de los issues y su relación con los riesgos. Siempre sin perder de vista no caer en el énfasis de la documentación ni en el afán de reportar. Se debe buscar la simplicidad y el foco en el flujo de trabajo sin impedimentos.

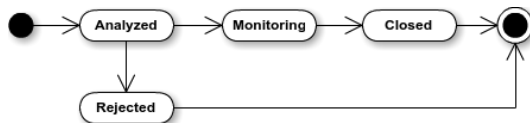


Figura 5.5: Estados posibles de un riesgo.

Las herramientas gráficas útiles para dicha gestión son: “Tablero de Obstáculos” (obstacle board) o el “Calendario de Obstáculos” (Issues calendars).

### 5.0.7 Métricas

Las métricas son una combinación de atributos cuantificables pertinentes que comunican información relevante acerca de la calidad de nuestros productos y nuestra productividad<sup>11</sup>. Para la ingeniería del software, las métricas proporcionan una indicación de la calidad de algún tipo de representación del software basadas en un conjunto de medidas indirectas<sup>12</sup>. O sea que las mismas están relacionadas a unidades de medida e indicadores que ayudan a medir, descubrir y tomar decisiones para mejorar, corregir problemas, hacer estimaciones, controlar calidad, evaluar productividad y hacer control de proyectos. Intentar medir para mejorar nuestra comprensión de entidades particulares es tan poderoso en ciencia, en ingeniería de software como en cualquier disciplina.

Bajo el marco Scrum ayuda al equipo a medir su propio desempeño para poder hacer cambios basados en hechos. También es un soporte a la gestión de proyectos para poder medir el avance del proyecto, revisar la productividad, el desempeño<sup>13</sup> del equipo y poder hacer comparaciones relativas entre evolución de diferentes equipos.

#### Unidades de medida

Para poder usar métricas y medir es necesario unidades de medida. Las más usadas son:

1. **Story Points:** El Story Points o SP es una unidad de medida que indica una cantidad de alcance o trabajo que puede ser entregado o tamaño de producto estimado para entregar. Es una unidad de medida que representa la complejidad o esfuerzo necesario para terminar las tareas de una historia <sup>14</sup>. El SP sirve como estimación de la

---

<sup>11</sup>[INCOSE, 2005]

<sup>12</sup>[Roger S. Pressman, 2002]

<sup>13</sup>Las métricas se pueden usar para medir desempeño, pero no se deben utilizar en forma punitiva ni tampoco en sistemas de evaluación de desempeño de empleados.

<sup>14</sup>[Jipson Thomas, 2015]

complejidad en forma relativa y sumativa que hacen los desarrolladores. También hay que tener en cuenta que es una unidad subjetiva que depende de qué equipo hace la estimación de medida.

2. **Business Value Point:** El Business Value o BV es el valor añadido que la historia aporta al negocio<sup>15</sup>. A semejanza del SP, el Business Value Point o BVP sirve como estimación del valor de negocio en forma relativa y sumativa que hace el PO. También hay que tener en cuenta que es una unidad subjetiva que depende de qué PO o equipo de personas hace la estimación de medida.

## Indicadores

Hay muchos indicadores que nos pueden ser de utilidad, como por ejemplo los siguientes<sup>16</sup>:

1. **Velocity:** La 'velocidad'<sup>17</sup> es el número de unidades de trabajo o puntos de historia SP estimados y aceptados por un equipo en una iteración Sprint. En otras palabras, es el conjunto de puntos de historia totales conseguidos (aceptados) por el equipo al final de cada Sprint<sup>18</sup>.

Esta definición es la clásica y es algo genérica. Hay tres formas más específicas de interpretar y medir la velocidad:

- (a) **Velocidad de trabajo:** cuando la velocidad muestra la cantidad de trabajo o funcionalidad que un equipo entrega (aceptada) en un sprint<sup>19</sup>, incluyendo las de valor indirecto. En este sentido, las historias de usuarios completadas (tomadas del backlog técnico)

---

<sup>15</sup>[Benoit Pointet/Thomas Botton, 2012]

<sup>16</sup>Scott y Jeff [Scott/Jeff, 2013] y la Scrumalliance en un artículo llamado "Velocity, How to Calculate and Use Velocity to Help Your Team and Your Projects", por Catia Oliveira (6 February 2014).

<sup>17</sup>"Sum of original estimates of all accepted work" [Scott/Jeff, 2013].

<sup>18</sup>[Jipson Thomas, 2015]

<sup>19</sup>"Velocidad en la que el equipo pueda completar el trabajo en un Sprint, número de funcionalidades entregadas en un sólo Sprint o Número de Story Points hecho en un determinado Sprint" [SBOK, 2013]

que tienen valor indirecto para el cliente, las correcciones de errores, deuda técnica, migraciones y refactorizaciones sí cuentan en la velocidad. En este caso, la velocidad del equipo da pocas indicaciones sobre el verdadero valor de negocio entregado y más sobre la capacidad que puede producir. Pues la velocidad, en este sentido, no suele tener relación directa con el valor de negocio entregado. Como no se suele aclarar bien la definición de velocidad se suele entender que se trata de esta perspectiva pero, sin embargo, en Scrum clásico se sobreentiende que los equipos deben entregar valor de punta a punta, por lo que la velocidad debería estar ligada al valor como la siguiente definición.

- (b) **Velocity en nueva funcionalidad:** cuando la velocidad muestra solo la cantidad de funcionalidad de valor para el negocio que un equipo entrega (aceptada) en un sprint. En este sentido, las historias de usuarios que no tienen ningún valor para el cliente o incompletas, las correcciones de errores y las refactorizaciones no cuentan en la velocidad<sup>20</sup>. En este caso, la velocidad del equipo da un indicio sobre el valor de negocio entregado por el equipo. En Scrum original o clásico se sobre-entiende que las historias son las que aportan valor, por tal motivo a veces no se aclara explícitamente esto.
- (c) **Value Velocity:** es una forma interesante para medir la productividad (sugerida por James Shore<sup>21</sup>) similar a la velocidad tradicional o velocidad de trabajo, excepto que se basa en estimaciones de valor de negocio (Business Value Point) hechas por el PO antes de la planeación.

2. **Average Velocity:** De la velocity de cada Sprint se calcula la velocidad promedio o "Average Velocity" que es el número

---

<sup>20</sup>[David Koontz, 2014]

<sup>21</sup>[James Shore, 2015]

de unidades de trabajo o SP promedio estimados y aceptados por un equipo en un conjunto de iteraciones Sprint. En un equipo ágil de velocidad estable, la velocidad promedio (por ejemplo de los últimos 4 Sprints) es un indicador adelantado de la velocidad estimada para el próximo Sprint (bajo las mismas condiciones de los Sprints usados para calcularla).

3. **Work Capacity:** La Capacidad es la suma de todos los trabajos reportados durante el Sprint, esten terminados o no<sup>22</sup>. La capacidad es generalmente igual o superior a la Velocity. Aunque la Capacidad puede, en raras ocasiones, caer por debajo de la velocidad. Esto se debe a que la velocidad se calcula en base a las estimaciones originales de trabajo, mientras que la capacidad se calcula en base a la suma de trabajo real reportado<sup>23</sup>. Por lo que en el caso de que esto suceda, lo que indica es que el equipo ha sobre estimado la complejidad de los trabajos solicitados. También existen otras formas de calcular o entender la capacidad. Por ejemplo:
  - (a) **Capacidad en puntos ideales:** La capacidad puede ser una idealización basada en la velocidad promedio, o sea, los puntos de la historia que se pueden considerar gastar en la próxima carrera de velocidad.<sup>24</sup>
  - (b) **Capacidad en horas:** La capacidad puede ser calculada en horas basados en la cantidad de miembros y la cantidad de horas efectivas de trabajo en un Sprint. Por ejemplo en un equipo de 8 miembros, con 6 horas de trabajo efectivo y un Sprint de 10 días, la capacidad en horas es igual a 480 hs ( $8 \times 6 \text{ hs} \times 10$ ).

Cuando se definió el marco de trabajo, como algo mínimo de cosas para que funcione, se dejó lo más simple posible. Debido a ello, el concepto de Velocity sí es partes del marco

---

<sup>22</sup>Scott y Jeff [Scott/Jeff, 2013]

<sup>23</sup>Scott y Jeff [Scott/Jeff, 2013]

<sup>24</sup>[Satish Thatte, 2013]

de trabajo, pero Working capacity no lo es, aunque es ampliamente usado <sup>25</sup>.

4. **Focus Factor:** El factor de foco revela el foco que el equipo ha tenido para entregar valor y su prestancia. El mismo es la relación entre la Velocity y la capacidad de trabajo:  $( \text{Velocity} / \text{Work Capacity} ) \times 100\%$ . La misma debe permanecer en la vecindad de 80 % en promedio para un equipo saludable. Estos puntos de datos por debajo del 80 por ciento indican un equipo que está interrumpido o incapaz de convertir su trabajo estimado en trabajo aceptado mostrando poca previsibilidad. Cuando el valor es alto, cercano al 100, el equipo ha estado bajo la previsión de su capacidad, aunque esto no indica necesariamente que están trabajando bien. Por ejemplo, el equipo puede estar aparentando ser perfecto forzando la coincidencia.
5. **Targeted Value Increase (TVI+)** El TVI+ responde a cuánto cambio ha habido en la velocidad del equipo a través del tiempo desde el primer Sprint. Es la Velocity del Sprint actual dividido la Velocity Original (velocity del primer sprint):  $( \text{Current Sprint's Velocity} / \text{Original Velocity} ) \times 100\%$ . Sirve para medir el aumento de la contribución de valor de un equipo en base a su velocidad origen Sprint a Sprint. Por ejemplo, si el resultado es 200% significa que el equipo ha duplicado su capacidad de resolver con éxito la complejidad requerida.
6. **Business Value Delivered:** Un indicador relevante es el valor de negocio entregado. Se pueden contabilizar los BV entregados en cada Sprint.
7. **Happiness Metric:** Un estudio de Harvard muestra que la felicidad aumenta la producción de cualquier tipo de trabajo, pues "la gente feliz es 12 % más productiva"<sup>26</sup>. Además, un principio de la agilidad es desarrollar en torno

---

<sup>25</sup>Erich Buhler, Agilib.org 2015, Proyecto Mercury 2015 (LAN), [Erich Buhler Coach, 2015].

<sup>26</sup>[Group of U.K. Researchers, 2014]



a individuos motivados. En base a esto podemos intentar medir la felicidad Sprint a Sprint<sup>27</sup>.

La felicidad del equipo es un indicativo de la salud del equipo en relación a su capacidad de entrega de valor. Para medir la felicidad del equipo podemos hacer encuestas o responder las siguientes preguntas:

- (a) ¿Cómo me siento acerca de mi trabajo? Se puede usar una escala de 1 a 5 (ver figura 5.6).



Figura 5.6: Medida de satisfacción

Otra escala para medir felicidad puede ser de siete puntos. Por ejemplo a la pregunta... ¿Cómo calificaría su felicidad en este momento? Se puede responder con 1 que es totalmente triste, 2 es muy triste, 3 es triste, 4 es ni feliz ni triste, 5 es bastante feliz, 6 es muy feliz y 7 es completamente feliz<sup>28</sup>.

- (b) ¿Qué va a hacer que me sienta mejor?

Cuando tenemos la información de cada miembro del equipo, el equipo puede hacer una lluvia de ideas de cómo hacer para mejorar y aumentar la felicidad en el siguiente Sprint y darle curso y seguimiento a las acciones propuestas.

También se puede medir la felicidad de los Stakeholder como indicador de grado de satisfacción que indica qué tan contentos estuvieron con los resultados del Sprint y con la Review. Este relevamiento relacionado a la percepción de valor entregado se puede hacer en la misma Review.

Por último, hay que tener en cuenta que para que las métricas estén bajo un marco Scrum deben respetar los principios de la

<sup>27</sup>[Jeff Sutherland, 2014a]

<sup>28</sup>[Group of U.K. Researchers, 2014]

agilidad. La medida principal es el software funcionando. Las métricas deben ser un medio de comunicación efectiva e impulsar la transparencia. Deben servir al equipo para que se auto-organice en procesos de corrección de desvíos y mejora continua, apoyando la reflexión sobre cómo ser más efectivos. Y, principalmente, se debe buscar siempre la simplicidad. Las métricas que agregan complejidad innecesaria o sobre-información no son acordes a la agilidad. Lo difícil es encontrar el equilibrio entre un Scrum minimalista y hacer ingeniería. Si no medimos no podemos hacer análisis del sistema de trabajo, o sistema productivo, para mejorar con datos empíricos como lo requiere el trabajo ingenieril; pero si medimos excesivamente podemos estar generando desperdicio (burocracia y trabajo no relevante) y ruido informativo (datos que no generan información útil).

## Capítulo 6

# Escalamiento

Scrum es aconsejable para ser óptimo en equipos chicos y proyectos pequeños, con agrupación de personas de múltiples disciplinas en un solo equipo para maximizar el ancho de banda de las comunicaciones, la visibilidad y la confianza. Esto sucede porque cuando los equipos son grandes aumenta el acoplamiento de individuos complejizando las comunicaciones y dificultando la coordinación y el buen desarrollo de las reuniones Scrum. Además se desprende del principio o Ley de Brooks, que dice que cuando se agregan personas a un proyecto o equipo aumentan los canales de comunicación pudiendo generar sobrecarga de comunicación. Por este motivo y desde un punto de vista purista, cuando se quiere implementar Scrum en proyectos grandes que requieren muchas personas, en su forma ortodoxa, no es recomendable. Pero se han encontrado maneras organizativas para aplicar Scrum en estos casos, como así también en grandes organizaciones. A esto se llama "Scaling Scrum" o Escalamiento de Scrum.

En "Scaling Scrum" los desafíos más importantes son: manejar las dependencias e integrar el trabajo en todos los niveles. En general Scrum se escala mediante reproducción de equipos con configuraciones adecuadas y la cohesión y acoplamiento mediante algún sistema de integración ágil, como el uso de "Scrum de Scrum", equipos de Product Owners, equipo de integración, etcétera. Hay varios casos que se pueden investigar, como por

ejemplo las implementaciones de: Google, Spotify, Adobe, Nexus, etc.

## 6.1 Equipos

En "Scaling Scrum" los equipos funcionan como células que, a medida que la organización se expande, se clonan sus estructuras como en reproducción celular. La estructura de las células depende del problema que se quiere resolver. Un equipo puede ser un Scrum-Team formado por diferentes roles, tales como: UX, SM, PO, BA, QA, FE Dev, BE Dev, etcétera. Por ejemplo, un equipo puede estar formado por: PO, SM, UX, 3 FE Dev y 3 BE Dev; otra estructura puede ser: PO, SM, BA, QA, 2 FE Dev y 2 BE Dev. En esta vía tenemos tres lineamientos básicos de configuración: equipos Scrum de características, equipos Scrum de componentes y uno mixto. Y cada una de estas modalidades pueden tener diversas configuraciones de roles.

### 6.1.1 Equipos de características

Abordar el problema de proyectos grandes con "Equipos Scrum de características" consiste en la conformación de equipos totalmente multi-funcionales con un enfoque "Whole Team"<sup>1</sup>, capaces de operar en todos los niveles de la arquitectura del producto con el fin de ofrecer las características centradas en el cliente (ver figura 6.1). O sea que cada equipo trabaja sobre determinadas características de producto (Features) o PBIs desarrollando todos los niveles del sistema a desarrollar (end-to-end). En este sentido, los equipos son homogéneos entre sí pero heterogéneos internamente, con integrantes de habilidades diversas y características profesionales multidisciplinarias. Para lograr esto se debe conformar una organización de aprendizaje donde los equipos practican el aprendizaje continuo, donde aprenden para abarcar los componentes arquitectónicos.

---

<sup>1</sup>En un enfoque Whole Team todos pueden hacer, hasta cierta medida, alguna tarea de otro [Juan Gabardini, 2015].

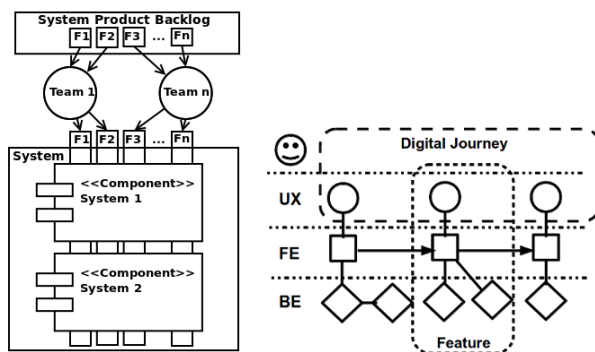


Figura 6.1: Esquema de equipos de características

En esta arquitectura de organización es necesario coordinar el trabajo de los diferentes equipos. Los Scrum Master deben reunirse con regularidad, promoviendo la transformación a través de una lista visible de los impedimentos de organización. Los Scrum Master, además deberán estar familiarizados con biografía relacionada a este problema de escalabilidad como "Scaling Lean and Agile Development" [Larman/Vodde, 2008].

### 6.1.2 Equipos de componentes

Abordar el problema de proyectos grandes con "Equipos Scrum de componentes" (ver figura 6.2) consiste en que cada equipo sólo es responsable de la ejecución de ciertos componentes dedicados en el sistema de los cuales el equipo es dueño de su desarrollo. Desde esta perspectiva se pueden tener equipos dedicados por capas (capa front-end, capa de servicios, capa de persistencia) y por componentes de arquitectura de software (diferentes componentes como librerías, servicios o subsistemas).

Para terminar una PBI o historia de usuario hay en la mayoría de los casos la necesidad de dividir las historias en partes más pequeñas que podrían ser implementadas dentro de un solo componente. Además se genera dependencias entre los diferentes equipos haciendo necesario procesos de integración periódica y

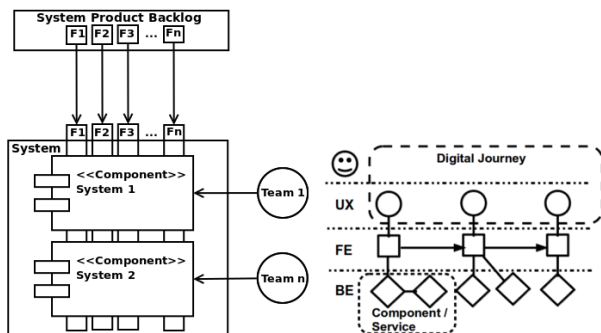


Figura 6.2: Esquema de equipos de componentes

coordinación de equipos. En muchos casos, una sola historia de usuario no se puede implementar dentro de un único sprint y, en su defecto, depende de los resultados de otras historias desarrolladas por otro equipo que aún no están disponibles. A esto se lo llama "pipeline" y debe evitarse en lo posible o gestionarse apropiadamente.

La ventaja de utilizar equipos de componentes es que es más fácil asegurar una determinada arquitectura del sistema. Por ejemplo si se quiere asegurar una Arquitectura SOA o una de Microservicios que está orientada a componentes. Esta idea está, entre otras cosas, basada en la "Ley de Conway"<sup>2</sup> que sugiere que las organizaciones pueden replicar su arquitectura en los productos que ellas producen [Martin Fowler, 2014].

Por otro lado, puede tener como desventaja que las personas se pueden especializar sólo en pequeñas partes del sistema y el conocimiento global sobre el sistema en su conjunto podría perderse [Scrum-Institute, 2015]. En este caso podría tener lugar una optimización local, ya que el equipo a veces puede tomar decisiones que están optimizadas para el componente individual, pero las mejores soluciones desde una perspectiva del sistema total podrían haber sido desestimadas u obviadas.

<sup>2</sup>Conway's Law [Conway, 1968] no es exactamente una ley, sino que es más bien una observación que Conway publicó en 1968.

### 6.1.3 Equipos mixtos

También es posible la configuración de equipos mixtos que son la combinación de equipos de características y de componentes (ver figura 6.3).

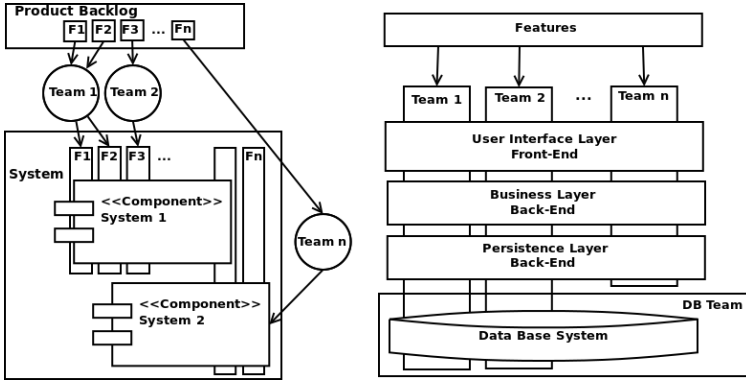


Figura 6.3: Ejemplo de equipos mixtos.

## 6.2 Integración

### 6.2.1 Scrum de Scrum

Scrum de Scrum es una forma de organización y técnica para escalar Scrum a grupos grandes de personas u organizaciones con proyectos grandes, programas o portafolios. Consiste en distinguir un integrante con el rol de Embajador, denominado "Ambassador", por cada Equipo Scrum [Stefanini, 2013]. El Embajador será quien participará en reuniones Daily con Embajadores de otros equipos. A esta reunión de Embajadores se la llama "Scrum de Scrum" o SoS. Habitualmente se usa que el rol de Embajador lo desempeñe el Scrum Master, pero puede ser desempeñado por otro integrante del Equipo de Desarrollo. También puede ser desempeñado por un integrante del Equipo acompañado del Scrum Master.

La reunión SoS se comporta como una Daily donde los Embajadores reportan la situación de su equipo y sus impedimentos. El embajador de cada equipo comenta la respuesta a las siguientes tres preguntas:

- **¿Qué hicimos ayer?**
- **¿Qué vamos a hacer hoy?**
- **Si tenemos obstáculos: ¿Qué impedimentos tenemos?** ¿Qué impedimentos tenemos a nivel de equipo? ¿Si algún otro equipo nos bloquea para algo? ¿Si bloqueamos en algo a algún otro equipo?

La SoS puede ser facilitada y moderada por una persona que puede desempeñar un rol de facilitador, coordinador e integrador inter-equipos. Este rol puede tener diferentes nombres y el SBOK lo denomina Chief Scrum Master o CSM. El CSM apoyará y brindará soporte a los SM de diferentes equipos.

### 6.2.2 Scrum de Scrum de Scrum

En organizaciones donde hay muchos Equipo Scrums trabajando en varias partes de un proyecto o producto grande, puede ser necesario otro nivel más de coordinación, ya que hay equipos que no participan en SoS de otras áreas. Aquí es cuando se aplica la "Scrum of Scrum of Scrum"<sup>3</sup> o SoSoS, que consiste en una reunión frecuente a un nivel superior de la SoS.

### 6.2.3 Comunidades

En una organización grande, independientemente de Scrum, se suelen formar comunidades. Estas comunidades se dan por especialidades técnicas o roles. Las mismas son útiles para colaboración entre equipos, generación de discusiones globales y resolución de problemas transversales. Es aconsejable que se den en forma orgánica. Aunque son particularmente útiles las comunidades de SM y de PO. La de SM es una posibilidad de

---

<sup>3</sup>[SBOK, 2013]



mejora continua del marco de trabajo y la de PO puede funcionar como equipos de PO para tratar temas del negocio transversal y administrar backlog en conjunto.

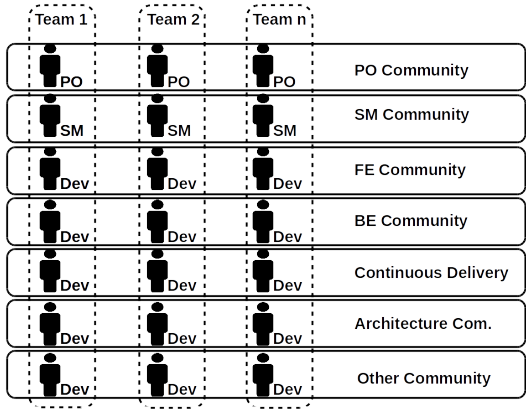


Figura 6.4: Ejemplo de comunidades

6.2.4 DevOps

La integración a escala para que una Software Factory sea ágil y lleve adelante uno de sus principios, el Continuous Delivery, es lo que se denomina DevOps. DevOps es la integración de Ingeniería de Desarrollo de Software con la Ingeniería de Operaciones, que es todo el soporte IT y de plataforma para posibilitar que el proceso de desarrollo sea ágil, haciendo entregas frecuentes de calidad y logrando la colaboración entre el personal de desarrollo y el personal de operaciones a lo largo de todas las etapas del ciclo de vida de producción de software. DevOps es una parte central del agilismo en la entrega de software eficiente, por medio de la integración de equipos, metodologías ágiles, técnicas y stack tecnológico como una sola organización. El objetivo principal de DevOps es minimizar los cuellos de botella en el pipeline de entrega, haciéndolo más eficiente y ágil <sup>4</sup>.

<sup>4</sup>[Sanjeev Sharma and Bernie Coyne, 2015]

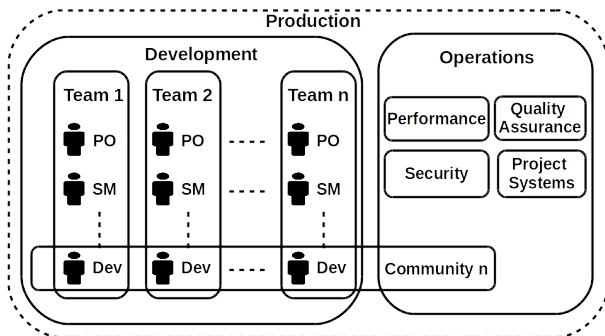


Figura 6.5: Ejemplo de integración de equipos en DevOps

Una manera de integrar Operaciones con Desarrollo es mediante el uso de herramientas compartidas, misma cultura, comunidades en común, procesos ágiles compartidos, comunicación estandarizada, etcétera. Algo crucial en una organización madura es la automatización máxima en procesos de Continuous Integration y Continuous Delivery.

En cuanto a Scrum, el área de operaciones no necesita implementar Scrum, pero sí puede llevar en práctica sus valores, ser ágil (con Lean, Kanban, DAD, etc.) y tener en cuenta los ciclos Scrum de los equipos de desarrollo. DevOps es sólo un paso importante para unirse a la cultura general de la colaboración ágil y que debe involucrar a todas las disciplinas en una organización.

## 6.3 Modelos de escalamiento

Existen diversos modelos, marcos y frameworks para escalamiento como lo son el SAFe, LeSS, modelo Nexus, DAD, Lean Management, Agile Portfolio Management (APM), Recipes for Agile Governance in the Enterprise (RAGE), etc. A continuación una breve reseña de algunos de ellos.

### 6.3.1 SAFe

El framework Scaled Agile Framework o SAFe consiste en una base de conocimientos de patrones integrados modulares, por niveles organizativos, destinados al desarrollo Lean-Agile a escala empresarial. Scrum se integra en SAFe en el primer nivel inferior organizativo, el nivel de equipo. A este nivel le siguen los niveles de programa, flujo de valor y portafolio. SAFe propone elementos opcionales de integración, un nuevo rol llamado Product Manager quien dirige a los POs de los equipos, otro nuevo rol llamado Release Train Engineer quien coordina a los SMs, la coordinación de backlogs por medio de un Backlog a nivel programa y la coordinación entre equipos por medio de eventos conjuntos y refinamientos, eventos trimestrales, System Team, Scrum of Scrums, Team Backlog y Team PO. Para más información hay que remitirse a la guía y definición de prácticas dada por el Sitio web de SAFe.

### 6.3.2 LeSS

Por otro lado tenemos al framework Large-Scale Scrum o LeSS que es un marco de desarrollo de productos que extiende Scrum con reglas de escalado y directrices sin perder los propósitos originales de Scrum. LeSS tiene un conjunto de principios alineados a Scrum y sumado el pensamiento sistémico. Además tiene alrededor de 28 reglas disponibles en 3 libros y propone diferentes mecanismos de coordinación como eventos conjuntos y refinamiento, CoP, SoS, Integración de código, feature teams y Area Product Owner. LeSS también propone subdivisiones de la organización por áreas de valor o Customer Value. Para más información hay que buscar en la guía del sitio web “less.works” y en diferentes libros que tratan el tema, como por ejemplo: Large-Scale Scrum, More with LeSS de Craig Larman y Bas Vodde.

### 6.3.3 Modelo Nexus

Nexus es un marco de trabajo para desarrollar y mantener iniciativas a escala de desarrollo de productos y software basado

en Scrum. Fue creado por Ken Schwaber y Scrum.org. Es un exoesqueleto cuyo corazón es un conjunto de Equipos Scrum combinados para crear un Incremento Integrado, usando una única Lista de Producto Backlog, manteniendo "Listas de Pendientes" por Sprint y apoyados por un "Equipo de Integración Nexus" quien brinda soporte para asegurar que se produzca un Incremento Integrado. Para más información basta leer la "Guía Nexus" publicada en Scrum.org.

### 6.3.4 DAD

El marco de decisiones de procesos "Disciplined Agile Delivery" o DAD, brinda una orientación ligera para ayudar a las organizaciones a escalar agilidad y buscar optimizar sus procesos de tecnología de la información (IT) de una manera sensible al contexto ágil y DevOps. Para ello, muestra cómo las diversas actividades, como la entrega de soluciones, las operaciones, la arquitectura empresarial, la gestión de cartera y otros equipos y áreas trabajan juntas. Las reglas implicadas en este marco hacen que la empresa busque ser consciente y escalable. La referencia primaria para DAD es el libro "Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise", escrito por Scott Ambler y Mark Lines.

## Capítulo 7

# Complementos de Scrum

Scrum no tiene como objetivo dar instrucciones precisas a los equipos sobre la forma concreta en la que deben llevar a cabo su desarrollo. Scrum espera que los equipos hagan lo que sea necesario para ellos para entregar el producto de calidad. Las prácticas y herramientas de desarrollo cambian y mejoran de manera continua y los buenos equipos trabajarán constantemente en pos de obtener el mejor uso de las mismas. Desde esta perspectiva, se puede consensuar en un anillo de complementos útiles que rodea el núcleo de Scrum y que pueden potenciar el trabajo ágil y la práctica de ingeniería. Estos complementos son métodos y técnicas complementarias a Scrum, sugerencias y recomendaciones, lecciones aprendidas y conceptos aledaños. En este capítulo trataremos estos temas.

## 7.1 Gamification y Dinámicas De Grupo

Un facilitador ágil (SM, AM, Agile Coach, etc.) debería facilitar los cambios culturales, la cohesión de equipos, el trabajo colaborativo, la creatividad y la innovación. Para ello, puede traer a la mano, de su cartera de juegos, dinámicas de grupo que son herramientas de Gamification (Game Thinking). Estas dinámicas son actividades en forma de juegos que ayudan a llegar a un resultado grupal de una manera divertida, comprometida, colaborativa, activa, motivadora, visual (Visual Thinking), experimental (Design Thinking) y creativa (Creative Thinking). Hay que considerar que estas dinámicas deben reforzar las actividades de ingeniería y no reemplazarlas.

## 7.2 Técnicas de Justificación de Negocio

### 7.2.1 Retorno de la Inversión

El ROI sirve para estimar el posible valor de Proyecto. Se usa para evaluar los ingresos netos que se esperan obtener a partir de un Proyecto, restando los costos o inversiones estimadas de un Proyecto de su ingreso, y dividiendo esto (beneficio neto) por los costos previstos, con el fin de obtener una tasa de retorno. Su fórmula es:

$$\text{ROI} = (\text{IP} - \text{CP}) / \text{CP}$$

Donde IP es el Ingresos del Proyecto, entendido como beneficios económicos esperados estimados, y CP es el Costo del Proyecto o costo de desarrollo.

### 7.2.2 Business Value versus complejidad

Se puede hacer uso de medir Business Value Points para hacer un seguimiento del valor de negocio entregado mediante diagramas de Business Value Delivered (Ver figura 7.1) o Business Value BurnUp<sup>1</sup>. Además, también es útil para ayudar en la priorización de historias de usuario, pues logrando hacer un diagrama de Business versus Complexity (Ver figura 7.1) podemos tener una visión del peso en complejidad de cada historia (US) o PBI en contraste con el valor de negocio que aporta y, en base a esto, poder hacer priorizaciones en las sesiones de planificación.

---

<sup>1</sup>El Business Value BurnUp es un diagrama de BV entregado acumulado por cada Sprint [Scrum Alliance, 2005]

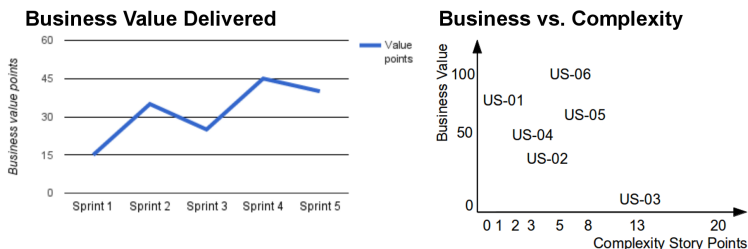


Figura 7.1: Diagramas de Business Value

## 7.3 Técnicas para requerimientos

### 7.3.1 Historias de usuarios

Para Scrum las hipótesis de requerimientos se representan en ítems de Backlog de producto PBI. A estos PBIs se los suele denominar "Story" o "Historias de Usuarios"<sup>2</sup>. El Product Backlog incluye incisos o Stories que aportan valor al cliente y que suelen ser descripción de requisitos funcionales. Aunque también puede incluir entradas para exploración de características, necesidades del cliente u opciones técnicas, requerimientos no funcionales, el trabajo necesario para lanzar el producto, y otros incisos, así como la configuración del entorno o arreglar defectos<sup>3</sup>. O sea que puede proporcionar valor en forma indirecta mediante el aumento de la calidad o la reducción de los incidentes en el largo plazo<sup>4</sup>.

Las historias representan un concepto verificable y simple, que el Product Owner quiere implementar en el producto. Según el concepto general de metodología Ágil, la historia se define como una "promesa de una conversación" o una "descripción de una característica"<sup>5</sup>. Según esta perspectiva, la historias de usuario representa una "funcionalidad de aplicación" para un usuario y que brinda un beneficio (ROL + FUNCIONALIDAD + BENEFICIO). Las mismas se escriben siempre con lenguaje de

<sup>2</sup>Las "User Story" son una técnica de eXtremme Programming (XP)

<sup>3</sup>[Scrum-Institute, 2015]

<sup>4</sup>[Scrum-Institute, 2015]

<sup>5</sup>[UNTREF, 2014], [Dan North, 2015]



negocio y respetando la siguiente pauta de estructura:

**COMO <ROL> QUIERO <FUNCIONALIDAD> PARA QUE <BENEFICIO>**

Por ejemplo:

- **COMO** ScrumMaster
- **QUIERO** que las reuniones no se extiendan más de un tiempo consensuado
- **PARA QUE** no se pierda el foco.

La historia de usuario no es exactamente un requerimiento, pero puede considerarse como el título de una hipótesis de requerimiento como recordatorio de algo relevante a conversar con el usuario o cliente. En consecuencia lo relevante es su evolución como conversación <sup>6</sup>.

La historia de usuario describe lo que el usuario quiere hacer desde la perspectiva de una interacción con un proceso de negocio, describe el objetivo del usuario en términos de necesidad de obtener algo que se hace en el negocio <sup>7</sup>.

Según Dan North y en el marco de Behavior-Driven Development (BDD), una Story es más un requerimiento, pues tiene que ser una descripción de un requisito y su beneficio para el negocio, y un conjunto de criterios con los que todos estamos de acuerdo de qué es o lo que se hace, "lo que el cliente necesita" <sup>8</sup>.

### Criterio de demarcación

Las historias de usuario bien escritas son esenciales para el desarrollo ágil y la ingeniería de software. Existen ciertas características a tener en cuenta a la hora de escribir una historia de usuario y determinar cuál es una bien escrita o una válida. El

---

<sup>6</sup>[UNTREF, 2014]

<sup>7</sup>[Scott Bellware, 2008]

<sup>8</sup>[Dan North, 2015]

criterio de demarcación aceptado en metodología ágil en general es el criterio INVEST<sup>9</sup> y que consta de seis características a cumplir<sup>10</sup>:

1. **Independiente:** una historia debe ser atómica. Es decir que se debe buscar asegurar la cohesión funcional y el bajo acoplamiento entre historias distintas. La fuerte dependencia entre diferentes historias hace que sea más difícil planificar, priorizar y estimar.
2. **Negociable:** debe ser conversable y en consecuencia negociable (es viva). Las historias "no son requerimientos contractuales"<sup>11</sup> ni requisitos rígidos, y su detalle y precisión debe poder evolucionar en el tiempo en procesos de refinamiento y conversaciones en una ingeniería de requerimientos evolutiva.
3. **Valuable:** debe generar un valor al cliente (usuario o comprador). Una Story sin una declaración de la motivación del usuario a menudo se piensa que es no accionable; es decir, la historia no se puede estimar o implementar fácilmente<sup>12</sup>. Lo ideal, además de la declaración explícita del beneficio, es que haya una forma de medir el valor que aporta al cliente.
4. **Estimable:** se debe poder estimar. La historia debe brindar la suficiente información que de la certidumbre necesaria para que los desarrolladores puedan estimarla y permitir, de este modo, que se pueda priorizar y planificar.
5. **Pequeña:** debe ser lo suficientemente pequeña para entrar en un sprint o iteración. El tamaño pequeño de las historias ayuda a reducir el tamaño del lote, incrementar el flujo de valor hacia el cliente y asegurar que cada miembro del equipo de desarrollo pueda hacer una contribución valiosa cada día. Por ejemplo, en cada Sprint se podrían completar entre cuatro y ocho historias pequeñas. A medida que una

---

<sup>9</sup>INVEST es un acrónimo creado por Bill Wake.

<sup>10</sup>[UNTREF, 2014], [Scrum Alliance, 2015]

<sup>11</sup>"Las user stories no son obligaciones contractuales"[Cohn, 2004]

<sup>12</sup>[Scott Bellware, 2008]

historia es más grande va a tener más errores asociados a la estimación y alcance, y aumentará la probabilidad de terminar en un Sprint fallido.

6. **Probable:** debe poder permitir la contrastabilidad. La contrastabilidad del enunciado de la historia es la propiedad de ésta de ser capaz de ser sometida a una prueba empírica y repetible a fin de evaluar su adecuación o no a los hechos a los que se refiere o resultados esperados. Esta característica es crucial para hacer ingeniería, sin la contrastabilidad no se hace ingeniería. Un ejemplo de historia no probable sería: "Como usuario quiero un software hermoso y fácil de usar".

### ¿Qué NO es una Historia de Usuario?

Si no cumple con el criterio INVEST es bastante probable que no sea una Story. Por ejemplo una tarea puramente técnica (que le interesa sólo al desarrollador), un refactoring técnico o un Spike no cumplen con INVEST y en su generalidad no son Story. Veamos los siguientes ejemplos:

- **Technical Story:** "Cada historia tiene que ser valorada por los usuarios. Pero eso sería un error."<sup>13</sup> Pues en realidad, una historia de usuario describe la funcionalidad que será valiosa para un usuario o comprador de un sistema o software <sup>14</sup>. Hay historias que especifican aspectos técnicos que no son valiosos para el usuario sino más bien para el cliente o comprador (la valoración es transitiva y no directa). Historias técnicas pueden ser valorados por los compradores que contemplan la compra del producto, pero no serían valorados por los usuarios reales. Pero esto no es lo que podemos llamar una "Technical Story" sino que es una Story con aspectos técnicos que tiene valor para el cliente. Para Scott Ambler, en Agile Modeling, una Story es una definición de muy alto nivel de un requerimiento que puede ser funcional o no, por ejemplo de un requerimiento

---

<sup>13</sup>[Cohn, 2004]

<sup>14</sup>[Cohn, 2004]

técnico <sup>15</sup>, algo así: "**COMO** usuario **QUIERO** que las transcripciones estén disponibles en línea a través de un navegador estándar **PARA QUE** me permita acceder desde cualquier lugar"<sup>16</sup>.

Cuando se habla de "Technical Story" se referencia a aquella historia técnica que sólo es valorada por los desarrolladores. Estas historias son las que se quieren evitar y considerar como Technical Story que podría ser una tarea técnica, tarea de investigación o tarea por deuda técnica. Este tipo de historias se centran en la tecnología y las ventajas para los programadores. Es muy posible que las ideas detrás de estas historias sean buenas, pero en cambio deben ser escritas para que los beneficios a los clientes o usuarios sean evidentes. Esto permitirá, al cliente, priorizar inteligentemente estas historias en el calendario de desarrollo <sup>17</sup>.

Para la Agile Alliance una Story no se corresponde en general a un "componente técnico" o de la "interfaz de usuario", pues a pesar de que a veces puede ser un atajo útil hablar de por ejemplo "la historia de diálogo de búsqueda", pantallas, cuadros de diálogo y botones, no son historias de usuario <sup>18</sup>.

- **Spike:** No es una User Story ya que es una tarea de investigación y/o experimentación en formato time-boxing, por lo que no cumple con ser probable ni estimable (no cumple con dos criterios INVEST); puede ser considerada una historia de investigación<sup>19</sup>, como por ejemplo: **COMO** desarrollador **QUIERO** investigar sobre **PERFORMANCE PARA QUE** luego se pueda mejorar la prestancia de la aplicación.
- **Refactoring Task:** Es una tarea técnica de reingeniería, mejora técnica (Improvement) o deuda técnica (Technical

---

<sup>15</sup>[Scott Ambler, 2015]

<sup>16</sup>[Scott Ambler, 2015]

<sup>17</sup>[Cohn, 2004]

<sup>18</sup>[Scrum Alliance, 2015]

<sup>19</sup>[Cohn, 2004]

Debt), que no es valuable o su valoración es a futuro o en relación a un atributo de calidad que puede no estar directamente solicitado por el cliente o no impacta en beneficio inmediato para el cliente, ya que podría estar resolviendo una "deuda técnica"<sup>20</sup>. Esto no quiere decir que no pueda haber una "story de refactoring" que puede proporcionar valor en forma indirecta mediante el "aumento de la calidad o la reducción de los incidentes en el largo plazo". Un ejemplo es: **COMO** desarrollador **QUIERO** hacer un refactory del servicio X **PARA QUE** se pueda mejorar su arquitectura, lograr código más elegante, más sencillo y asegurar la mantenibilidad de la aplicación.

- **Bug Fixing:** La corrección de errores no es una característica nueva de producto, puede deberse a deuda técnica, no ser de valor inmediato para el Dueño de Producto y ser considerada historia técnica. Como estrategia, la misma, puede ser tomada fuera de la carrera (de historias de usuario) del Sprint, es decir, el equipo puede mantener un factor de enfoque en historias de usuario para asegurar tiempo para corregir errores <sup>21</sup>. Hay que tener en cuenta que la corrección de uno o varios errores puede ser tratada simplemente como cualquier otra historia<sup>22</sup> (aunque no sea exactamente una historia de usuario).

## Componentes de una Historia de Usuario

Las historias de usuarios se componen de tres elementos comunmente denominados las tres 'C'<sup>23</sup>:

---

<sup>20</sup>Deuda técnica es la incurrida involuntariamente debido a trabajos de baja calidad o deuda incurrida intencionalmente (McConnell, Ward Cunningham) y que consta en arquitectura no escalable, defectos en el código, documentación inservible o desactualizada, problemas para migrar o actualizar funcionalidades o problemas por falta de mantenimiento.

<sup>21</sup>[Henrik Kniberg, 2007]

<sup>22</sup>[Cohn, 2004]

<sup>23</sup>Las 3 Cs de Ron Jeffries.

1. **Carta o tarjeta (Card):** Las historias se deben poder escribir en una tarjeta de papel pequeña. Por ese motivo la redacción de las mismas debe ser clara, precisa y concisa para caber en una tarjeta de papel.
2. **Conversación:** Las historias deben tener y generar conversaciones cara a cara entre el Product Owner y el Equipo de Desarrollo.
3. **Confirmación:** Deben estar suficientemente explicada para que el Equipo de Desarrollo qué es lo que se desea construir y qué es lo que se espera como resultado de dicha implementación. La confirmación es el "Criterio de Aceptación" que los desarrolladores deben tener en cuenta para que la misma sea aceptada por el cliente.

### Criterio de Aceptación

Las historias de usuario tienen límites específicos para las características del producto, proceso o servicio definido en la hipótesis de requerimiento que determinan los resultados esperados para que la historia sea aceptada por el cliente. En otras palabras, el criterio de aceptación es el conjunto de afirmaciones útiles para que el cliente valide la historia.

La estructura de un criterio de aceptación presentado como escenario puede ser como la siguiente:

- **Scenario 1:** Título
  - **Given** [contexto] And [otro contexto]...
  - **When** [evento]
  - **Then** [resultado] And [otro resultado]...

### 7.3.2 BDD

Una manera de ayudar al desarrollo ágil a partir de requerimientos es usar el desarrollo conducido por comportamiento (Behavior Driven Development o BDD), el cual es un proceso de desarrollo por el que se busca comenzar a partir de un lenguaje común que

une la parte técnica y la de negocio, y desde ese lenguaje común se parte hacia el Testing y, posteriormente, al desarrollo. El proceso consta básicamente en escribir las historias de usuario, escribir los escenarios (o criterios de aceptación en forma de escenarios) y automatizar las pruebas de aceptación de los escenarios.

## 7.4 Técnicas de Estimación

### 7.4.1 Estimación relativa

Los puntos de historia o "Story Points" son una medida relativa de estimación de un ítem de trabajo o una historia de usuario para poder medir su tamaño y, además, es útil para medir la velocidad de un equipo. Se usa para hacer estimaciones relativas, que quiere decir que las historias se comparan entre sí, buscando mantener una relación proporcional entre ellas según sus puntos de historia. O sea que si se piensa a nivel de esfuerzo, una historia que tiene 3 puntos de historia requerirá tres veces más de esfuerzo que una que sea de un punto historia. La estimación es relativa porque es subjetiva al equipo que estima. Si diferentes equipos toman a los puntos de historia como medida de complejidad de una historia, pueden llegar a valores distintos porque considerarán la complejidad acorde a su percepción de complejidad que puede ser diferente. Hay equipos que pueden tomar al SP como expectativa de esfuerzo en un día ideal de trabajo (día sin impedimentos) y la expectativa de esfuerzo resultará relativa al equipo que la forma<sup>24</sup>.

Hay que tener en cuenta que "bajo ningún punto de vista la estimación (en story points) es un compromiso"<sup>25</sup> y que los puntos de historia no son medidas objetivas convertibles en forma precisa a horas. Justamente se usan puntos de historia para no estimar en horas hombre.

---

<sup>24</sup>[Cohn, 2004]

<sup>25</sup>[UNTREF, 2014]

### 7.4.2 Poker de planeación

El Poker de planeación o "Planning Poker"<sup>26</sup> es una técnica de estimación y planificación ágil que se basa el consenso por sabiduría de grupo.

La dinámica consiste en que el Product Owner inicia leyendo una historia de usuario ágil o describe una característica para los desarrolladores estimadores. Cada estimador posee una baraja de "Cartas de Poker de Planificación". Los valores representan el número de puntos de la historia, día ideal, u otras unidades en las que el equipo estima. Los estimadores discutir la función, haciendo preguntas al Product Owner, según sea necesario. Cuando la función se ha discutido plenamente, cada estimador selecciona de forma privada una tarjeta a representar a su estimación. Todas las tarjetas se revelaron a continuación, al mismo tiempo. Si todos los estimadores seleccionan el mismo valor, que se convierte en la estimación. Si no, los estimadores discuten sus estimaciones. Los estimadores de valores altos y bajos deben compartir sus razones. Tras un nuevo debate, cada estimador vuelve a seleccionar una tarjeta de estimación, y todas las tarjetas se revelan de nuevo al mismo tiempo. El proceso se repite hasta que se logre el consenso, o hasta que los estimadores deciden que la estimación ágil y planificación de un ítem en particular debe ser diferida hasta que la información adicional se puede adquirir.

#### Cartas de Poker de Planificación

Las Cartas de Poker de Planificación que más se usan son las de Fibonacci y las de talles de remeras:

1. **Talles de remeras:** Las cartas de talles contempla los valores XS (muy pequeño), S (Pequeño), M (Mediano), L

---

<sup>26</sup>El "Planning Poker" proviene de un paper presentado por James Greening llamado "cómo evitar análisis parálisis en la planificación de liberaciones"[James Grenning, 2002] donde se basa en el método Wideband Delphi para realizar la estimación de requerimientos de forma colaborativa. Luego el método fue popularizado por Mike Cohn con su libro "Agile Estimating and Planning"[Cohn, 2005]



- (grande), XL (muy grande) y XXL (extra grande). En su forma más simple se puede usar: S, M y L. Estos valores sirven para estimar épicas, tamaños de historias o trabajo a alto nivel como el tamaño de funcionalidades.
2. **Fibonacci:** Las cartas Fibonacci modificada contempla los valores 0, 1, 2, 3, 5, 8, 13, 20, 40 y 100, que se basa en la secuencia fibonacci que se suele recomendar. Estos valores son útiles para estimar complejidad de historias o trabajo a nivel medio. Realizando una estimación relativa, estos valores corresponden a puntos de historia con los que se puede pesar un ítem de trabajo.

## 7.5 Técnicas de Comunicación

### 7.5.1 Hacer silencio

Una técnica para lograr silencio en un grupo es pedir silencio cuando se levanta la mano, los que ven la mano levantada tienen que callarse y levantar a su vez la mano. Cuando cualquiera ve a alguien con la mano levantada se debe callar y a su vez levantar la mano. Este comportamiento se termina propagando rápidamente en todos los integrantes hasta que se logra el silencio. Es una técnica muy útil en la reuniones cuando las personas se dispersan comunicándose en forma caótica o cuando alguna actividad de conversación libre se extiende del tiempo necesario.

### 7.5.2 Gestión visual

La gestión visual es la utilización de elementos y técnicas visuales, como complemento de Scrum, para la organización del trabajo y la irradiación visual del trabajo. Esto es útil para soporte del equipo y para brindar transparencia hacia interesados externos al equipo. Es aconsejable que los irradiadores de información presenten las siguientes características:

1. **Ubicación ostensible:** estar ubicado en el lugar de trabajo en paredes o paneles visibles.
2. **Soporte físico:** estar hechos de material tangible como papel en vez de usar software, salvo el caso de monitores grandes. Por ejemplo en forma de afiches (flip-chart), carteles o pizarras visibles.
3. **Resumen autoexplicativo:** contener información importante autoexplicativa y didáctica.

Ejemplos de irradiadores de información son: tablero de obstáculos o impedimentos, los gráficos de esfuerzo pendiente burndown y burnup, el gráfico de velocidad, indicadores de estados del build (usando semáforos), métricas de errores, riesgos activos, etcétera.

7.5.3    Tableros Scrum/Kanban

El tablero Scrum/Kanban o "Scrum Kanban Board" es una técnica que consiste en utilizar un tablero Kanban (ver figura 7.2), como irradiador de información, para el manejo del ciclo de estados de las tareas y de los impedimentos. El tablero Kanban debe ser visible por todo el equipo y por lo tanto transparente para todos los involucrados. Por ejemplo, durante una Daily Scrum todo el equipo es capaz de ver qué tareas se resuelven, cuáles no se han abordado todavía y qué impedimentos existen.

Ejemplo de un Scrum kanban board

Ver figura 7.2.











PB	SB	TO DO	ON GOING	DONE	BLOCKED	OK
						
						
						
						

Figura 7.2: Ejemplo de un tablero Kanban para Scrum.

Ejemplo de un Scrum board

Un Scrum board, Kamban Board o taskboard físico (Ver figura 7.3) es una tabla donde se colocan Post-it representando tareas.

7.5.4    Tablero de Obstáculos

El Tablero de Obstáculos (ver figura 7.4) es una herramienta visual útil para hacer seguimiento de los obstáculos o bloqueos del equipo que surgen en el trabajo del Sprint. El dueño del Tablero de Obstáculos es el Scrum Master, quien se encarga de buscar











STORY	TO DO	ON GOING	DONE
	 		
	  		

Figura 7.3: Ejemplo de un tablero Scrum.

remover todos los obstáculos que puedan paralizar o ralentizar el trabajo del equipo o desviarlo de su foco esencial. El tablero también le es útil al equipo para tener la visibilidad del estado de los obstáculos (ver figura 7.4).

OBSTACLE	
NEW	ESCALATE
ASSIGNED	BACK TO TEAM
CLOSED	

Figura 7.4: Ejemplo de un tablero de Obstáculos.

7.5.5 Calendario de Obstáculos

El calendario de obstáculos es una herramienta visual útil para mostrar los problemas surgidos en todos los sprints hasta el momento (ver figura 7.5.).

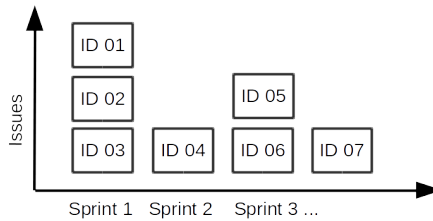


Figura 7.5: Calendario de obstáculos ejemplo.

## 7.5.6 Gráficos de esfuerzo pendiente

### Gráfico Burndown

Un gráfico de trabajo pendiente o "Burndown charts"<sup>27</sup> (figura 7.6) es uno que muestra el trabajo restante ("Remaining Scope") o que queda por hacer versus tiempo. Muestra la velocidad a la que se están completando los PBIs o historias reflejando el avance y permitiendo extrapolar si el Equipo podrá completar el trabajo en el tiempo restante.

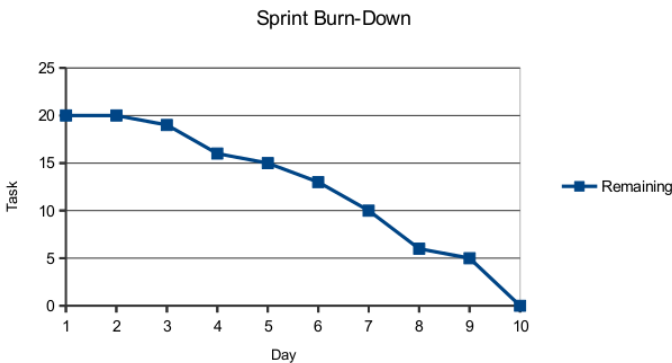


Figura 7.6: Ejemplo de un gráfico Burndown con un sprint de dos semanas y 20 tareas comprometidas.

<sup>27</sup>El Burndown muestra la cantidad de trabajo que queda. [SBOK, 2013]

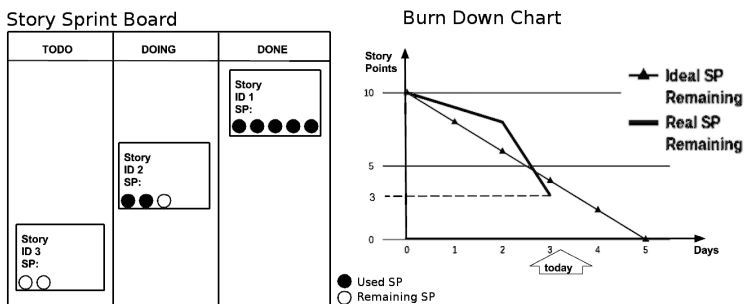


Figura 7.7: Ejemplo de cómo llevar un Burndown con un sprint de 5 días y quedando 3 storypoints por quemar.

Se pueden utilizar dos gráficos de esfuerzo pendiente:

1. **Burndown de Sprint:** Días u horas pendientes para completar las tareas de la iteración (sprint burndown chart), realizado a partir de la lista de tareas o historias de la iteración. Normalmente se utiliza para saber cuánto falta para terminar las historias comprometidas en un Sprint. Es un diagrama de dos ejes: en el eje X el tiempo en días de duración del sprint, en el eje Y la cantidad de trabajo comprometido con el cliente durante el sprint en las unidades que se hayan acordado, story points o tareas (ver figuras 7.6 y 7.7).
2. **Burndown de Proyecto:** Días pendientes para completar los requisitos del producto o proyecto (product burndown chart), realizado a partir de la lista de requisitos priorizada (Product Backlog).

## Gráfico Burn-Up

El gráfico de trabajo realizado o Burn-up<sup>28</sup> (figura 7.8) es muy similar al Burndown, con la diferencia de que se parte del cero, y

<sup>28</sup>El Burnup muestra el trabajo realizado como parte de la Sprint. [SBOK, 2013]

se va marcando la cantidad de trabajo completado en el Sprint. En este gráfico la curva va hacia arriba acercándose a una línea que representa el alcance comprometido. En este tipo de gráfico es más fácil visualizar los cambios de alcance.

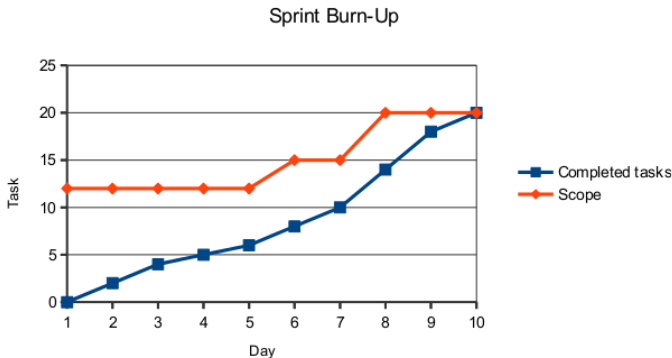


Figura 7.8: Ejemplo de un gráfico Burnup de tareas comprometidas en un sprint de dos semanas.

### 7.5.7 Gráficos de Velocidad

#### Velocity Chart

El gráfico de velocidad muestra todas las unidades estimadas en el plan y aceptadas (US Story Points) para un período de iteraciones realizadas<sup>29</sup>. Un gráfico de la velocidad típico de un equipo de proyecto ágil podría ser como el siguiente gráfico de la izquierda (ver 7.9):

También se puede incluir la velocidad promedio (ver gráfico de la derecha 7.9).

---

<sup>29</sup>[Scrum Alliance, 2014]

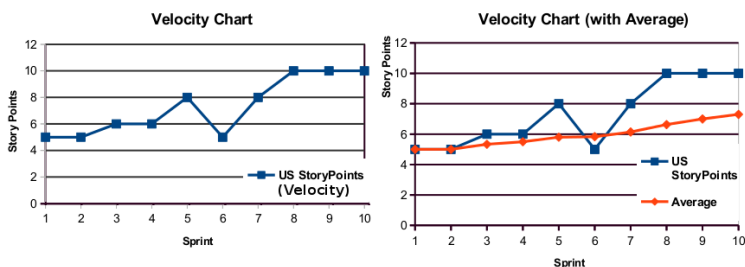


Figura 7.9: Diagrama de Velocidad y Diagrama de Velocidad con Velocidad Promedio.

## 7.6 Técnicas de Programación

Hay diferentes técnicas que pueden ser utilizadas en el proceso de desarrollo de software bajo el marco de Scrum. A continuación se describen algunas principales que sirven de soporte a metodologías ágiles:

### 7.6.1 Técnica Pomodoro

Se usa 'Pomodoro'<sup>30</sup> como una técnica de gestión del tiempo y se basa en la hipótesis de que las pausas frecuentes en el trabajo pueden mejorar la agilidad mental. Por este motivo, la técnica consiste en dividir el tiempo dedicado a un trabajo en intervalos (time-boxing) de 25 minutos (llamados 'pomodoros') separados por descansos. Los primero tres descansos son de 5 minutos y el descanso después del cuarto pomodoro es de 15 minutos, para luego repetir la secuencia en forma cíclica. Durante el intervalo pomodoro el trabajo debe ser focalizado, por lo que no se deben admitir distracciones o trabajos ajenos al trabajo principal. Esta es una técnica útil en el desarrollo ágil y en la programación en pareja (pair programming) además de otros contextos de trabajo.

<sup>30</sup>La Técnica Pomodoro es un método para la administración del tiempo desarrollado por Francesco Cirillo a fines de los años 1980[Cirillo Francesco, 1980].



### 7.6.2 Programación de a pares

La programación de a pares, “pairing” o “pair programming” consiste en que dos programadores participen conjuntamente en un esfuerzo combinado de desarrollo en un puesto de trabajo. La misma se puede extender al desarrollo de a pares (análisis, diseño, codificación, pruebas, diseño de interfaces de usuario, diseño gráfico, etc.).

Existen muchas razones por las que esta técnica es recomendable y las principales pueden ser las siguientes:

1. **Calidad:** Mejorar la calidad logrando reducir defectos por "revisión de a pares" mientras se programa.
2. **Aprendizaje:** Lograr el aprendizaje en equipo distribuyendo y nivelando el conocimiento. Pues, focalizarse en el aprendizaje es crucial para organizaciones scrum de equipos por características, donde todos deben manejar un conocimiento amplio y multidisciplinario.
3. **Propiedad colectiva:** Mejorar los diseños y las estimaciones debido a que todos tienen un conocimiento de la estructura del producto, un empoderamiento del mismo (propiedad colectiva del código) y de cómo puede impactar un cambio en parte de él.
4. **Resolución de problemas:** Permitir superar problemas difíciles de forma más simple, más rápido o al menos efectiva cuando trabajan juntos. La idea es que dos personas piensen mejor que una en resolver un problema.

### 7.6.3 Revisión de a pares

La revisión de a pares o "peer review" es una práctica intrínseca de la actividad científica en un sistema de evaluación del trabajo científico, donde los miembros de la comunidad revisan los trabajos de sus pares<sup>31</sup>. Por este motivo, implementar este tipo de

---

<sup>31</sup>Formalmente el proceso de revisión por pares del trabajo científico fue iniciado en 1753 por la “Royal Society of London”.

prácticas en el desarrollo de software hace que sea una actividad ingenieril. El beneficio es lograr una mejor calidad y una reducción de defectos. Según Boehm "el 60 por ciento de los defectos de código se pueden eliminar durante las revisiones de pares"<sup>32</sup>. Así como en ciencia se revisan los trabajos de investigación antes de ser publicados, en ingeniería de software se revisa el código antes de ser desplegado.

La práctica consiste en que una vez que el programador desarrolló un trabajo de codificación, solicite ante un colega compañero o un conjunto de ellos la inspección del código. Si se encuentran errores u observaciones, el desarrollador deberá mejorarlo y hasta que no tenga el visto afirmativo el código, no será desplegado o integrado al producto. Hay que tener en cuenta que para que la revisión por pares sea realmente productiva, debe ser ejecutada con el máximo de rigor, aunque implique retrasar despliegues de producto y no cumplir con los plazos comprometidos por el equipo. Cuando se aplica Scrum, el máximo de rigor incluye, además de la calidad del trabajo, prestar particular importancia al cumplimiento de la definición de terminado (DoD).

#### 7.6.4 TDD

El "Test Driven Development" (TDD) o "desarrollo guiado por pruebas"<sup>33</sup> es una práctica, técnica de desarrollo ágil y técnica de programación que hace foco en el comportamiento especificado de unidades de software<sup>34</sup> como disparador para el desarrollo de pruebas. O sea, basándose en especificaciones, esta práctica tiene como táctica escribir primero las pruebas (código de prueba) y después el código (código de producto) para luego realizar ciclos de "refactorización" [Kent Beck, 2010]. Con este enfoque se pretende, entre otras cosas, que el diseño sea el código mismo (código de producto) y que las pruebas concretas (código de prueba) sirvan de documentación. Esta técnica se complementa con la automatización de pruebas. En todo ese conjunto de pruebas concretas, las pruebas de unidad se usan para validar métodos

---

<sup>32</sup>[Boehm, 2001]

<sup>33</sup>[Jurado, 2010]

<sup>34</sup>[Wikipedia 2014]

individuales y conjuntos de métodos. En consecuencia, en este tipo de desarrollos, se logran sistemas con un gran conjunto de pruebas (muchas líneas de código en pruebas).

Si tenemos que asignar un lema a este enfoque es "la prueba en primer lugar" y su afirmación filosófica principal sería que "el diseño emerge del refactoring originado por pruebas".

### 7.6.5 Integración continua

La integración continua (Continuous Integration<sup>35</sup>) es una práctica de ingeniería de software y modelo informático que consiste en hacer integraciones automáticas de un proyecto, compilación y ejecución de pruebas, lo más a menudo posible para así poder detectar fallos cuanto antes.

### 7.6.6 Entrega continua

El Continuous Delivery es un enfoque de ingeniería de software en el que los equipos producen software en ciclos cortos, garantizando releases de forma fiable en cualquier momento del desarrollo. Este modelo suele ser parte incluida en la cultura DevOps y, a su vez, suele incluir la práctica de Continuous Deployment para desplegar, en forma automática, código a producción.

Trabajar bajo el marco ágil y no trabajar con Continuous Integration ni Continuous Delivery puede conducir a un trabajo poco ingenieril y menos eficiente.

---

<sup>35</sup>Continuous Integration fue propuesto inicialmente por Martin Fowler.

## 7.7 Checklists

Las Listas de Control o Check Lists son útiles para controlar actividades repetitivas como las ceremonias y lograr un mínimo de control de calidad procedimental. Puede ser de utilidad tener un listado de validación para el inicio de cada ceremonia y otro para el fin. A continuación se exponen algunos ejemplos propuestos por el autor:

### 7.7.1 Planning Checklist

- Checklist previo: Antes de la planning se deben corroborar los siguientes incisos:
  - Invitaciones hechas.
  - Corroborar asistencias.
  - Ubicación (lugar para reunirse, disponibilidad de sala).
  - Recursos materiales necesarios (papel, marcadores, Post-it, reloj-cronómetro, cartas de poker-planning, etc.).
  - Calendario del Sprint a planificar (tener en cuenta feriados, días festivos, capacitaciones, vacaciones, etc.).
  - Tener en cuenta la capacidad del equipo (datos históricos, velocity y capacity).
  - El PO tiene un conjunto de historias priorizadas que pasan el DoR para proponer.
  - El PO o el equipo revisaron el Backlog Técnico para proponer tareas técnicas (bugs, mejoras y deuda técnica).
  - El PO tiene el objetivo del Sprint para proponer.
  - El SM debe tener clara la dinámica y programa de la reunión para explicarle al equipo o que el equipo la sepa.
- Checklist final: Luego de finalizar la planning se deben corroborar los siguientes incisos:

- Se obtuvo un conjunto de ítems de trabajo para el Sprint Backlog.
- Las user stories comprometidas pasaron el DoR.
- Se tiene un objetivo de Sprint.
- Se hizo el compromiso formal con el PO.
- El equipo participó y tiene claros los incisos anteriores.
- Si es necesario asentar datos para histórico o métricas (cómo tiempo real insumido, riesgos, etc.).

### 7.7.2 Refinement Checklist

- Checklist previo: Antes del refinamiento se deben corroborar los siguientes incisos:
  - Invitaciones hechas.
  - Corroborar asistencias.
  - Ubicación (lugar y/o disponibilidad de sala).
  - Recursos materiales necesarios (papel, marcadores, Post-it, reloj-cronómetro, cartas de estimación de tamaño, etc.).
  - El PO tiene un conjunto de historias priorizadas y completas para proponer.
- Checklist final: Luego de finalizar el refinamiento se deben corroborar los siguientes incisos:
  - Se tiene el backlog actualizado con las historias tratadas.
  - Si es necesario asentar datos para histórico o métricas (cómo tiempo real insumido, riesgos, etc.).

### 7.7.3 Review Checklist

- Checklist previo: Antes de la review se deben corroborar los siguientes incisos:

- Invitaciones hechas.
  - Corroborar asistencias de invitados.
  - Catering si es necesario (café, galletas, caramelos, etc.).
  - Ubicación (lugar y/o disponibilidad de sala).
  - Recursos materiales necesarios (proyector, cables, Post-it, marcadores, etc.).
  - Tener las historias disponibles para presentar.
  - Las historias deben ser evaluadas anterior a la review por el PO.
  - Validar cumplimiento del “Definition of Done”.
  - Presentación (diapositivas) y/o folletería (o afiches) si es necesario.
  - Tener un programa (Agenda y una dinámica).
  - Asegurar relevamiento de feedback.
  - Asegurar ensayo (práctica previa, recolección de evidencias, simulación, etc.).
  - El PO le comentó a todo el equipo quienes asistirán como Stakeholders.
  - Planificación y medición de curso de acción:
    - \* Plan A: Curso normal (software funcionando y disponible).
    - \* Plan B: Con imprevistos (plan de contingencia).
    - \* Plan C: Cancelación (plan de cancelación).
  - Disponibilidad de accesibilidad remota (herramienta de video conferencia o comunicación remota).
- Checklist final: Luego de finalizar la review se deben corroborar los siguientes incisos:
    - Tener claro y asentado qué historias fueron aceptadas y cuáles no (el criterio de aceptación del Sprint Review es que las historias de Usuario sean aceptadas o rechazadas durante la misma).

- Tener relevamiento o registro del feedback.
- Si es necesario asentar datos para histórico o métricas (cómo métrica de satisfacción de stakeholder, tiempo real insumido, cantidad de Stakeholders, cantidad de feedback, etc.).

#### 7.7.4 Retrospective Checklist

- Checklist previo: Antes de la retrospectiva se deben corroborar los siguientes incisos:
  - Invitaciones hechas.
  - Corroborar asistencias.
  - Ubicación (lugar y/o disponibilidad de sala).
  - Recursos materiales necesarios (papel, marcadores, Post-it, reloj-cronómetro, etc.).
  - Se tiene una dinámica a seguir (dinámica clásica, dinámica de la estrella de mar, etc.).
  - ¿Se tienen datos históricos o acciones de retrospectivas pasadas?
- Checklist final: Luego de finalizar la retrospectiva se deben corroborar los siguientes incisos:
  - Se tienen identificadas acciones para mejorar.
  - Si es necesario asentar datos para histórico o métricas (cómo tiempo real insumido, riesgos, etc.).





## Capítulo 8

# Malas prácticas

Velar por la correcta utilización de Scrum es más que el cuidado de un conjunto de reglas a seguir. Existen muchas causas diversas para que se fracase en su implementación, además de la de no seguir sus reglas. A esas acciones causantes, resultado de prácticas que perjudican su buen funcionamiento, las podemos llamar malas prácticas y constituirán pautas que se aconseja evitar. A continuación se listan algunas de ellas:

### 1. Aplicar mal la metodología

Una mala práctica es aplicar mal o en forma incompleta una metodología o técnica determinada. Por ejemplo, se critica a la metodología de cascada (Waterfall) o desarrollo en cascada porque se dice que no funciona, sin embargo lo que suele suceder es que no se aplica realmente (Masa Maeda <sup>1</sup>). El problema no es que no sirva o no funcione, sino que no lo hacemos bien (Masa Maeda 2012). Pues, en el artículo original de 1970 en el que Royce Winston expone el

---

<sup>1</sup>Masa K Maeda es PhD, fundador y CEO de Valueinnova USA (capacitación Scrum). Es el creador de Serious LeAP, consultor senior del Cutter Consortium en Boston, miembro del comité de dirección del Agile Testing Alliance y maestro en la Universidad de California en Berkeley. Es pionero de Lean y Kanban para trabajo de conocimiento y uno de los formadores del Lean Kanban University. Es una figura líder mundial en Agile y ha generado Serious Games de alto calibre.

desarrollo en cascada se habla de ciclos y de “fases sucesivas de desarrollo iterativo” [Winston Royce, 1970] ofreciendo unos consejos a seguir, cosa que no se suele hacer y se da por hecho que cascada no sirve. Algo parecido ocurre con Scrum.

Hay quienes dicen: "Scrum fracasó". Pero, sin embargo, lo que suele suceder es que se dice que se implementa Scrum aunque, en la práctica, no se cumplen sus recomendaciones o se hacen hibridaciones<sup>2</sup> con otras metodologías que dan como resultado algo que no es Scrum. Creyendo hacer Scrum, muchas no han logrado superarlo [Gantthead-James, 2010].

Respecto a este tema en una entrevista que le hicieron a Jeff Sutherland (uno de los creadores de Scrum) él dijo: La mayoría de las empresas implementan Scrum a medias. Por ejemplo, cualquier Scrum sin producto de trabajo al final de un Sprint es un Scrum fracasado y el 80 por ciento del Scrum escalado en Silicon Valley se encuentra en esta categoría, pues son "ágiles sólo de nombre". Cuando una empresa modifica o implementa sólo parcialmente Scrum puede estar ocultando u oscureciendo alguna disfuncionalidad que restringe su competencia en cuanto a gestión y desarrollo de producto (Ken Schwaber 2006).

## 2. Aplicar mal un principio

En ocasiones en que se aplica mal una metodología se puede deber a mal interpretar sus principios rectores, a generalizarlos o a caer en un una especie de fundamentalismo. Por ejemplo, si consideramos el "trabajo empírico" como algo fundamental, pero basándonos en ello pretendemos que todo trabajo se base en la experiencia directa de los desarrolladores (tipo prueba error) sin recurrir a experiencias previas, a memoria histórica, a procesos organizacionales o a conocimiento teórico. De este modo se puede caer en ser un equipo de trabajo reactivo, ciego de las consecuencias a largo plazo de sus acciones e ineficiente. No

---

<sup>2</sup>Metodología híbrida o mezcla con Scrum.

es necesario reinventar la rueda cada vez que se nos presenta la necesidad de usar una y a veces eso es lo que ocurre cuando se abusa del trabajo basado en la prueba y el error. Por otro lado, cuando nuestros actos tienen consecuencias que trascienden el horizonte de aprendizaje (nuestra experiencia cercana), se vuelve imposible aprender de la experiencia directa [Peter Senge, 1990]. En ocasiones, por poner otro ejemplo, se hace énfasis en el coraje. Pero se puede caer en ser heroico, cuando como dice una frase: "el programador heroico a menudo no ve el gran dragón" (Software Architect Bootcamp). Pues, fomentar el coraje no es fomentar necesariamente las acciones individuales heroicas en vez de buscar sentirse apoyados, trabajar colaborativamente y tener más recursos a disposición para promover el coraje para enfrentar desafíos más grandes.

### 3. No hacer ingeniería por aplicar una metodología

Metodologías como Scrum no establecen las prácticas específicas de ingeniería, por lo que se puede aplicar la metodología sin hacer ingeniería. En este caso los Scrum Master son responsables de promover un mayor rigor en la aplicación de las prácticas ingenieriles y de la definición de "terminado" (DoD) acorde al marco de ingeniería [Gantthead-James, 2010]. A veces sucede que la agilidad se convierte en un culto, despojando las prácticas reales de ingeniería de software por profesionales ágiles que no tienen una comprensión de la ingeniería de software, y de este modo simplemente se convierte en un conjunto de rituales sin sentido que, en su mayoría, son impedimentos y distracciones a la creación de software con éxito<sup>3</sup>.

Desde esta perspectiva, hay que tener en cuenta que el uso de post-its y gráficos bosquejos que parecen infantiles no debería reemplazar el uso de herramientas conceptuales de diagramación como son: Unified Modeling Language, Architecture Description Language, Business Process Modeling Notation, Conceptual Diagram or ConceptDraw,

---

<sup>3</sup>[Victor Hugo Certuche, 2016] [Mike Hadlow, 2014]

Causal Loop Diagram, Entity Relationship Diagram, Flow Charts (para control de flujo), Data Flow Diagram, Structure Chart, Stock and Flow Diagrams, Structured Systems Analysis and Design Method, Map Mind Diagram, etc. El uso de dinámicas y conversaciones tampoco debería sustituir el Análisis de Sistemas, Investigación Operativa y las prácticas profesionales<sup>4</sup> de Ingeniería de software; y la simplicidad no debería desplazar el uso de herramientas de software ni la eliminación de métricas fundamentales. La Ingeniería del Software intenta dar un marco de trabajo en el que se aplican práctica del conocimiento científico en el diseño y construcción de software con mayor calidad.

#### 4. No se cambia de mentalidad

A veces se implementa y usa una nueva metodología pero no se cambia de forma de pensar. Es que se puede considerar que la simple adopción de una herramienta de trabajo, sin una transformación personal que la acompañe, no sirve de nada, por ejemplo adoptar Scrum sin una transformación personal [Martin Alaimo/Kleer, 2014].

La gran mayoría de metodologías tienen detrás principios y maneras de pensar. Pues hay que entender que no se trata solo de fórmulas, sino también de formas de razonar. No se puede pretender trabajar en un equipo con alguna metodología ágil que implementa auto-organización si no se cree en la auto-organización. Hay personas que creen en el liderazgo centralizado y autoritario, por lo que no cambian su perspectiva y, en vez de adaptarse a la nueva manera de trabajar, terminan queriendo adaptar la manera de trabajar a su idea original, por ejemplo a la conducción centralizada y autoritaria. Esta actitud termina por generar malas prácticas que socavan el buen funcionamiento de una metodología determinada.

#### 5. No se cambia realmente la manera de trabajar

Existen aquellos casos en donde no se cambia realmente

---

<sup>4</sup>[SWEBOKv3, 2014]

la manera de trabajar y se aplica un “Scrum cosmético”<sup>5</sup>. Esto quiere decir que se agregan las ceremonias y los roles necesarios, pero realmente no se aplica Scrum ni se abandona la forma de trabajar que se venía haciendo hasta el momento.

## 6. Disociación entre producción y resto de la organización

La disociación entre producción y resto de la organización se da cuando se implementa agilidad, como Scrum, solo en el área de producción para hacer organizar el proceso de desarrollo, pero la gestión estratégica o las capas de gestión de alto nivel de la compañía desconocen la filosofía ágil o Scrum y gestionan los portafolios, programas y proyectos con las metodologías criticadas en este marco de trabajo. Algo semejante sucede con los vendedores de la organización. Pues, si los mismos venden productos especificados de antemano y en base a esas ventas se realizan compromisos contractuales rígidos y se exige que el área de producción cumpla con esos compromisos, por más que el área de producción intente usar Scrum, se puede caer en los mismos problemas que con Scrum se critica e incumplir con el o los proyectos. El uso de Scrum para lograr una organización de gestión y de desarrollo de un producto optimizado, es un proceso de cambio que debe ser dirigido o acompañado por las altas esferas de la compañía y que requiere que todos en la organización hagan cambios en sintonía (Ken Schwaber 2006).

## 7. Disociación entre Desarrollo y Operaciones

Si Desarrollo es ágil pero Operaciones no, tendremos un gran impedimento para lograr un DevOps eficiente y entrega continua <sup>6</sup>. Operaciones pueden generar bloqueos y cuellos de botella debido a: procesos pesados que

---

<sup>5</sup>Un modelo de madurez para equipos ágiles, Angel Medinilla, Jan 15 2015.

<sup>6</sup>Disociación entre el desarrollo y el resto de la organización se la conoce como Water-SCRUM-fall; y que significa, desde el punto de vista de DevOps, que mientras los equipos de desarrollo pueden haber adoptado prácticas ágiles, los equipos de operaciones no [Sanjeev Sharma and Bernie Coyne, 2015].

generan burocracia, falta de personal para dar soporte, personal con poca idoneidad, cultura waterfall, mala comunicación con los equipos de desarrollo, desincronización de mantenimientos o tareas técnicas y desarrollo que pueden provocar bloqueos generales, escaso o falta de soporte IT, infraestructura con mal funcionamiento, etcétera.

## 8. Equipos con integrantes aislados

Una mala costumbre en el trabajo en equipos es el trabajo en forma aislada de los integrantes, aún estando en la misma oficina. Ejemplos pueden ser las reuniones de daily hechas por alguna herramienta informática de conferencia (como Google Hangouts, Skype, etc.), el trabajo de integrantes ermitaños sentados solos y con auriculares o el trabajo de equipos donde todos sus integrantes son remotos. Esto debe evitarse priorizando el trabajo codo a codo, cara a cara.

## 9. Las reuniones como fin

Hay que tener en cuenta que las reuniones son un medio y no un fin. A veces se cae en una cultura de reuniones y minutas, como si se tratara de un "vicio organizacional"<sup>7</sup>. Por otro lado, en las organizaciones donde prima la confianza no es necesario asentar toda reunión en minutas. Las minutas deberían ser recordatorios y no acuerdos contractuales.

## 10. Excesivo foco en la entrega

El foco en la entrega, en la industria de software, es un anti-patrón que se ha arrastrado por años y que se consideraba como la forma de trabajo eficiente<sup>8</sup>. El foco en la entrega se refiere a cuando los equipos se centran en entregar, el éxito se mide en la cantidad de características que se entrega y en consecuencia hay una presión de cumplir con determinadas entregas pactadas o con determinadas características comprometidas. Cuando aumenta el foco en la entrega ocurre que disminuye el aprendizaje del equipo, la creatividad, la innovación y posiblemente el

---

<sup>7</sup>[UNTREF, 2014]

<sup>8</sup>[Erich Buhler, 2015]

posicionamiento. También ocurre que se gatillan sistemas de control mediante la solicitud de informes, reportes o presión social. Todo esto lleva un ciclo vicioso de trabajo estresante.

Para vencer el foco en la entrega hay que equilibrar con el aprendizaje. La agilidad impulsa que el enfoque debe ser el aprendizaje, que todos hayan aprendido nuevas y mejores formas de hacer las cosas, que el conocimiento se distribuya libremente y que los requisitos dejen de ser requerimientos y sean hipótesis a convalidar o invalidar, lo que requerirá conocimiento y maduración continua<sup>9</sup>.

#### 11. Estandarización en base a medidas subjetivas

En algunas organizaciones se incurre en una simplificación mecánica, fuera del marco ágil, cuando se insiste en comparar a los equipos usando SP, medir sus velocidades como indicativo de productividad y estandarizar niveles de madurez basados en ella. Desconocer que los SP de historia son una unidad de medida relativa y subjetiva para expresar una estimación del esfuerzo, incertidumbre y/o complejidad, no reconocer que es tan relativa que su tamaño varía en el tiempo según la subjetividad de quien los determinan y que los SP de un equipo pueden ser totalmente distintos a los de otro; es un indicio de no entender la agilidad. A veces el deseo y la necesidad de control y maximización de producción nubla la concepción de que la industria de software se basa en el trabajo intelectual y creativo, sobre un producto de contenido prácticamente intangible, como es el software. Si bien es necesario medir, controlar y planificar, siempre hay que tener en cuenta que la industria de software no es una manufactura de trabajo repetitivo, mecánico y en serie (producción en cadena). Por tal motivo hay que prestar particular importancia a la forma en que se mide la productividad y eficiencia; y en cómo se comparan equipos.

#### 12. Se hace lo que el Mesías dice

La mala práctica de “se hace lo que el Mesías dice” se refiere

---

<sup>9</sup>[Erich Buhler, 2015]

al caso en que un líder, un conjunto de líderes o una parte de la organización se comportan como un rey monarca que da saltos de fe hacia un consejero, cual si fuera un Mesías. Un mesías pueden ser algún Agile Coach consultor o una empresa de consultoría Ágil. Pues, que alguien sea Agile Coach no significa que tiene la bala de plata, que entienda de sistemas o que tiene la solución al problema o cambio organizacional que estamos necesitando. A veces sucede que los Agile Coaches son defensores de lo que saben (Scrum, Lean, Kanban, SAFe, LeSS, Crystal u otra metodología o técnica) y venden soluciones empaquetadas que no son necesariamente la solución óptima. Cada organización es un mundo y tiene sus particularidades que se deben analizar. Ninguna metodología resuelve todos los problemas y es necesario integrar diferentes, según el contexto y estado organizacional, según la propia experiencia organizacional y con herramientas de ingeniería de sistemas.



## Capítulo 9

# Conclusión

Aprecio que hayas dedicado tiempo a leer este libro y te agradezco por ello. Me motivó a escribirlo el poder ayudarte brindándote el resultado de mi trabajo de investigación sobre Scrum, mi experiencia en proyectos ágiles usando Scrum, mi participación en transformaciones organizacionales hacia la agilidad y de un diálogo constante con profesionales compañeros.

Para finalizar, después de lo expuesto, quiero concluir que Scrum no es un dogma, ni la solución a los problemas organizacionales. Como mostré, Scrum es un sistema de trabajo flexible para trabajar en contextos de incertidumbre y cambios frecuentes, principalmente en la Industria de Software. Y, si bien se puede aplicar a diversas organizaciones, no significa que resuelva los problemas de cualquiera. Cada empresa con problemas o con intenciones de pasar por transformaciones organizacionales, en busca de ser adaptable y eficiente, tiene sus propias disfuncionalidades que debe analizar y buscar solucionar en su contexto particular. Por otro lado, expuse algunas alternativas de escalamiento y herramientas complementarias para mostrar que, por sí solo, el “núcleo del Sistema Scrum” es insuficiente en organizaciones grandes. Pues, además son necesarias técnicas de XP, Lean, Kanban, System Thinking, Visual Thinking, Game Thinking, etcétera; y sobre todo, aplicar Ingeniería de Software en el desarrollo e Ingeniería de Sistemas en

la organización.

El talento de un SM se demuestra facilitando la mejora de las personas, herramientas y procesos. La habilidad de un PO se muestra en la búsqueda por lograr, con su equipo, el mejor producto posible para el negocio y en forma temprana. La excelencia en los desarrolladores es entregar, en forma temprana y frecuente, el producto de mejor calidad posible. La magia de Scrum consiste en no acostumbrarse ni conformarse y siempre avanzar mejorando. Haz tu propia experiencia con Scrum.

# Capítulo 10

## Glosario y Acrónimos

**Agile:** Agile, agilismo, ágil o agilidad se refiere -en general- al lineamiento con los valores ágiles expresados en el manifiesto ágil, cimientos que son cualidades que consideramos valiosas o deseables tener en cuenta.

**AM:** El rol Manager Ágil (Agile Manager) puede ser un Agile Project Manager o Administrador Ágil que media entre el equipo y Managers del resto de la organización. Pueden haber tenido experiencia gestionando equipos y proyectos, experiencia como SM, tener certificaciones Certified ScrumMaster (CSM), PMI-ACP u otras. Es un rol ágil, no de Scrum.

**Artifact:** Artefacto o almacén de incisos de trabajo.

**BDUF:** Gran diseño al inicio (Big Design Up Front).

**BA:** Analista de Negocio (Business Analyst). Un BA es quien realiza tareas de análisis de negocio que se describen en BABOK Guide. El BA es responsables de descubrir, analizar y sintetizar la información de una variedad de fuentes dentro de una empresa, incluyendo herramientas, procesos, documentación, y los stakeholders. Es responsable del relevamiento de las necesidades reales

de los stakeholders, lo cual con frecuencia involucra la investigar y aclarar sus deseos expresados con el fin de determinar los problemas y las causas subyacentes en el dominio del negocio. Ellos juegan un papel importante en la adaptación de las soluciones diseñadas y entregadas según las necesidades de los stakeholders. Las actividades que realizan incluyen: comprensión de las metas de la empresa y sus problemas, análisis de necesidades y soluciones, análisis de la organización (estructura, política y operaciones), la elaboración de estrategias, impulsión del cambio, y facilitación de la colaboración de los stakeholders.

**BE:** Software de bajo nivel o detrás de la interfaz de usuario (Back-End). Su desarrollo especializado sule darse por BE developer (BE Dev).

**DevOps:** Integración de development (desarrollo) y operations (operaciones), que se refiere a una cultura o movimiento que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales en las tecnologías de la información (IT).

**DoD:** Definición de terminado (Definition of Done). Significa que el trabajo sobre el ítem está completo y está listo para ser subido a operaciones o listo a entregarse.

**DoR:** Definición de preparado o completitud de refinamiento (Definition of ready). Significa que el refinamiento sobre el ítem de trabajo está completado y está listo para ser abordado en la planning.

**DT:** Equipo de desarrollo (Development Team). Se refiere al equipo de desarrolladores o a cualquier miembro desarrollador del equipo Scrum.

**DUF:** Diseño al inicio (Design Up Front).

**FE:** Software de Interfaz de Usuario (Front-End). Su desarrollo especializado sule darse por FE developer (FE Dev), desarrolladores de interfaz gráfica (Dev UI) y diseñadores UI.

**Features:** Representan interacciones y acciones del usuario con el sistema. Son funcionalidades que entregan valor de cara al usuario.

**IT:** Tecnología de la información (Information Technology) es la aplicación de ordenadores y equipos informáticos, con frecuencia utilizado en el contexto de la industria de software como el área encargada de brindar soporte de infraestructura y plataformas para el desarrollo de software.

**MMP:** Producto comerciable mínimo (Minimal Marketable Product).

**MVP:** Producto viable mínimo (Minimal Viable Product).

**PB:** Backlog de producto (Product Backlog).

**PBI:** Ítem de Product Backlog Item (story, technical story, technical debt, spike, etc.).

**PI:** Incremento de producto (Product Increment).

**PMI:** Instituto de Gestión de Proyectos (Project Management Institute).

**QA:** Rol específico para hacer pruebas de software y asegurar la calidad (Quality Assurance Tester).

**Risk:** Riesgo es la posibilidad de un problema o incertidumbre relacionada con un proyecto o PBI de un proyecto, que podría alterar significativamente el resultado del mismo de una manera potencialmente negativa. No tiene ningún impacto actual en el proyecto, pero podría tener un impacto potencial en el futuro.

**ROI:** Retorno de la inversión (Return On Investment).

**SCRUM:** Es un marco de trabajo. El término fue tomado prestado del rugby. En rugby es un juego en el que, por lo general, tres miembros de cada línea se unen opuestos unos a otros con un grupo de dos y un grupo de tres jugadores detrás de ellos, lo que hace un grupo de ocho personas, tres,

dos, tres formados en cada lado; el balón se deja entre la línea divisoria de ambos grupos, los jugadores están abrazados y tomados de la cintura de un compañero de equipo y los del frente hombro a hombro con el oponente, y se trata hacer fuerza grupalmente para desplazar al grupo rival y patear la pelota hacia atrás para que un compañero de equipo la tome.

**SB:** Backlog de iteración o Sprint Backlog. Es la pila de trabajo comprometido a desarrollar dentro de un sprint.

**Scaling Scrum:** Es cualquier implementación de Scrum donde múltiples Equipos Scrum construyen un producto, múltiples productos relacionados o un conjunto de características de un producto en uno o más Sprints (NEXUS, Scrum Profesional a Escala, Lucho Salazar, Versión 3.0.0, Agiles 2016 en Quito, 6-8 Octubre, 2016).

**SM:** Facilitador o ScrumMaster. Suele ser deseable que esté certificado como Certified ScrumMaster (CSM) o equivalente.

**SOA:** Arquitectura Orientada a Servicios (Service-Oriented Architecture).

**Spt:** Iteración (Sprint).

**SP:** Story Point o puntos de historia.

**UX:** Se refiere a la experiencia de usuario de un sistema (User Experience) o al rol encargado de analizar y diseñar dicha experiencia (UXer).

# Bibliografía

- [Agile Atlas, 2012] Agile Atlas (2012). *Agile Atlas. Scrum, una descripción.* by Scrum Alliance. Scrum Alliance Core Scrum 2012-2013.
- [Anacleto, 2005] Anacleto (2005). *El rol de la arquitectura de software en las metodologías ágiles.* por Lic. Valerio Adrián Anacleto. Epidata Consulting S.R.L.- Buenos Aires, Argentina. Diciembre de 2005.
- [AntiPatterns, 1998] AntiPatterns (1998). *AntiPatterns Refactoring Software, Architectures, and Projects in Crisis.* By William J. Brown Raphael C. Malveau Hays W. McCormick III Thomas J. Mowbray John Wiley and Sons. Inc. Publisher: Robert Ipsen, 1998.
- [Austin, 2003] Austin (2003). *Artful Making: What manager need to know about how artist work.* By Austin Robert D., Devin, Lee. Prentice Hall 2003.
- [Beck, 2001] Beck (2001). *Agile Manifesto By Beck, Kent.* URL: [www.agilemanifesto.org](http://www.agilemanifesto.org), 2001, como estaba en Octubre de 2012.
- [Benoit Pointet/Thomas Botton, 2012] Benoit Pointet/Thomas Botton (2012). *Article: Key Dimensions of User Stories.* By Benoit Pointet and Thomas Botton. Scrum Alliance. 12 January 2012.
- [Boehm, 1981] Boehm (1981). *Software Engineering Economics* By Boehm, B. Prentice-Hall.

- [Boehm, 2001] Boehm (2001). *Software Defect Reduction Top 10 list* By Boehm, Barry, Basili, Victor. IEEE Computery, January 2001.
- [CHAOS Report, 1994] CHAOS Report (1994). *The CHAOS Report*. By Standish Group. Standish Group.
- [Cirillo Francesco, 1980] Cirillo Francesco (1980). *Paperback: The Pomodoro Technique*. By Cirillo Francesco. ISBN-10: 1445219948, November 14, 2009.
- [Cohn, 2004] Cohn (2004). *User Stories Applied: For Agile Software Development*. By Mike Cohn. Addison Wesley.
- [Cohn, 2005] Cohn (2005). *Agile Estimation and Planning*. By Mike Cohn. Prentice Hall.
- [Conway, 1968] Conway (1968). *Article: How Do Committees Invent?* By Melvin E. Conway. Copyright 1968, F. D. Thompson Publications, Inc. Reprinted by permission of Datamation magazine, where it appeared April.
- [Dan North, 2015] Dan North (2015). *Whats is a story*. By Dan North. dannorth.net, article 2015.
- [David Koontz, 2014] David Koontz (2014). *Article: Metrics for a Scrum Team*. By David Koontz. Copyright Scrum Alliance Inc., agileatlas.org, February 13, 2014.
- [Don Kim, 2016] Don Kim (2016). *Article: The SBOK? Looks like anyone can create a PM standard these days!* by Don Kim. Agility and Project Leadership Blog.
- [Eric Raymond, 1997] Eric Raymond (1997). *The Cathedral and the Bazaar*. By Eric S. Raymond. O'Reilly.
- [Erich Buhler, 2015] Erich Buhler (2015). *Article: El Anti-patrón del foco en la entrega y Scrum*. By Erich Buhler. Ágil Ibérica, 2015.
- [Erich Buhler Coach, 2015] Erich Buhler Coach (2015). *Diseños sociales exitosos para organizaciones ágiles y digital I.T..* By Erich Buhler. Ágil Ibérica, Septiembre 2015, v1.0.



- [Ganttthead-James, 2010] Ganttthead-James (2010). *Seven Obstacles to Enterprise Agility*. By Ganttthead, James. ganttthead.com.
- [Greg Gehrich, 2012] Greg Gehrich (2012). *Build It Like A Startup: Lean Product Innovation* by Greg Gehrich. RSF Publishing 2012.
- [Group of U.K. Researchers, 2014] Group of U.K. Researchers (2014). *Happiness and Productivity*. By Andrew J. Oswald, Eugenio Proto, and Daniel Sgroi. University of Warwick, UK, and IZA Bonn, Germany. University of Warwick, UK JOLE 3rd Version: 10 February 2014. Published in the Journal of Labor Economics.
- [Henrik Kniberg, 2007] Henrik Kniberg (2007). *Scrum y XP desde las trincheras. Cómo hacemos Scrum*. by Henrik Kniberg. InfoQ.com y proyectalis.com.
- [INCOSE, 2005] INCOSE (2005). *Metrics Guidebook for Integrated Systems and Product Development*. By International Council of Systems Engineering, Wilbur, A., G. Towers, T. Sherman, D. Yasukawa and S. Shreve. INCOSE 2005.
- [James Grenning, 2002] James Grenning (2002). *Article: Planning Poker or How to avoid analysis paralysis while release planning*. By James Grenning. Copyright April 2002 All Rights Reserved james@grenning.net.
- [James Shore, 2015] James Shore (2015). *Article: Value Velocity, A Better Productivity Metric?* By James Shore. jamesshore.com, The Art of Agile SM James Shore, 2002-2015.
- [James Surowiecki, 2005] James Surowiecki (2005). *The Wisdom of Crowds*. By James Surowiecki. Anchorsbooks, August 16, 2005.
- [Jeff Sutherland, 2014a] Jeff Sutherland (2014a). *Article: Q&A with Jeff Sutherland on Scrum: The Art of Doing Twice the Work in Half the Time*. By Jeff Sutherland. infoq.com. Posted by Ben Linders on Nov 05, 2014.

- [Jeff Sutherland, 2014b] Jeff Sutherland (2014b). *Scrum: The Art of Doing Twice the Work in Half the Time by Jeff Sutherland*. Jeff Sutherland and Scrum Inc., 2014.
- [Jipson Thomas, 2015] Jipson Thomas (2015). *Article: How Story Point Estimation Can Help Managers*. By Jipson Thomas. ScrumAlliance.org, 21 August 2015.
- [Juan Gabardini, 2015] Juan Gabardini (2015). *Artítuco: Ya no estamos en testing Kansas. Publicado por Juan Gabardini, 28 de diciembre de 2015*. Desarrollo y testing ágil de software. Artículo publicado en "¿Qué pasó con los testers?", Kleer, Constanza Molinari, 5 enero de 2016.
- [Jurado, 2010] Jurado (2010). *Diseño Ágil con TDD. Por Carlos Blé Jurado y colaboradores. Prologo de José Manuel Beas*. Www.iExpertos.com, Primera Edición, 2010.
- [Ken Schwaber, 1995] Ken Schwaber (1995). *SCRUM Development Process by Ken Schwaber*. Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications OOPSLA.
- [Ken Schwaber, 2002] Ken Schwaber (2002). *Agile software development with Scrum By Ken Schwaber*. Prentice Hall. ISBN 0-13-067634-9.
- [Ken Schwaber, 2011] Ken Schwaber (2011). *Agility and PMI By Ken Schwaber*. KenSchwaber.wordpress.com.
- [Ken/Jeff, 2013] Ken/Jeff (2013). *La Guía de Scrum. La Guía Definitiva de Scrum: Las Reglas del Juego. por Ken Schwaber y Jeff Sutherland*. Offered for license under the Attribution Share-Alike license of Creative Common.
- [Kent Beck, 2010] Kent Beck (2010). *Test-Driven Development By Example. By Kent Beck*. Three Rivers Institute. Addison-Wesley Professional, 2003.

- [Larman/Vodde, 2008] Larman/Vodde (2008). *Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. By Craig Larman, Bas Vodde. Paperback.
- [Lawson/Martin, 2008] Lawson/Martin (2008). *On the Use of Concepts and Principles for Improving Systems Engineering Practice*. By Lawson, H. and J. Martin. INCOSE International Symposium 2008, 15-19 June 2008, The Netherlands.
- [Martin Alaimo, 2014] Martin Alaimo (2014). *Proyectos Ágiles con Scrum. Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos*. Por Martin Alaimo. Kleer (Agile Coaching and training).
- [Martin Alaimo/Kleer, 2014] Martin Alaimo/Kleer (2014). *Equipos más productivos: Personas e interacciones por sobre procesos y herramientas*. Por Martin Diego Alaimo. Kleer (Agile Coaching and training), Buenos Aires. ISBN 978-987-45158-7-2.
- [Martin Fowler, 2014] Martin Fowler (2014). *Article: Microservices*. By Martin Fowler, James Lewis. Martin fowler blog. Article 25 March 2014.
- [McConnell, 2006] McConnell (2006). *Software Estimation: Demystifying the Back Art* By McConnell, S. Microsoft Press.
- [Mike Hadlow, 2014] Mike Hadlow (2014). *Article: Coconut Headphones: Why Agile Has Failed* by Mike Hadlow by Mike Hadlow. DZone - Agile Zone, Mar. 19, 2014.
- [MIT Press, 2009] MIT Press (2009). *The Wisdom of Crowds in the Recollection of Order Information*. By Steyvers, M., Lee, M.D., Miller, B., and Hemmer, P. (2009). In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams and A. Culotta (Eds.) *Advances in Neural Information Processing Systems*, MIT Press, 2009.
- [Neal Ford, 2010] Neal Ford (2010). *Arquitectura evolutiva y diseño emergente: Investigación sobre arquitectura y diseño*.

*By Neal Ford, Application Architect.* IBM, ThoughtWorks Inc., DeveloperWorks, 05-04-2010.

- [OPS OMS, 1998] OPS OMS (1998). *Manual de identificación y promoción de la resiliencia en niños y adolescentes.* Por Dra. Mabel Munist, Lic. Hilda Santos, Dra. María Angélica Kotliarenco, Dr. Elbio Néstor Suárez Ojeda, Lic. Francisca Infante, Dra. Edith Grotberg. Organización Panamericana de la Salud. Organización Mundial de la Salud. Fundación W.K. Kellogg. Autoridad Sueca para el Desarrollo Internacional (ASDI).
- [Peter Senge, 1990] Peter Senge (1990). *La quinta disciplina. El arte y la práctica de la organización abierta al aprendizaje.* By Peter M. Senge. Editorial Granica, 2003. Edición original en inglés, 1990.
- [PMBOK, 1996] PMBOK (1996). *A Guide to the Project Management Body of Knowledge (PMBOK Guide).* By PMI and William R. Duncan Director of Standards. PMI Standards Committee.
- [PMBOK, 2004] PMBOK (2004). *Guía de los Fundamentos de la Dirección de Proyectos Tercera Edición (Guía del PMBOK).* Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299 EE.UU.
- [Roger S. Pressman, 2002] Roger S. Pressman (2002). *Ingeniería de Software, Un enfoque práctico.* Por Roger S. Pressman. Adaptado por Darrel Ince. McGRAW-HIL 2002.
- [Sanjeev Sharma and Bernie Coyne, 2015] Sanjeev Sharma and Bernie Coyne (2015). *DevOps for dummies, a Wiley Brand by Sanjeev Sharma and Bernie Coyne.* 2nd IBM Limited Ed.
- [Satish Thatte, 2013] Satish Thatte (2013). *Article: Agile Capacity Calculation.* By Satish Thatte. VersionOne.com. January 29, 2013.
- [SBOK, 2013] SBOK (2013). *Una guía para el conocimiento de Scrum (Guía SBOK) - 2013 Edición.* Título original: *A Guide*

*to the SCRUM BODY OF KNOWLEDGE (SBOK GUIDE) 2013 Edition.* SCRUMstudy, una marca de VMEdU, Inc.

- [Scott Ambler, 2015] Scott Ambler (2015). *User Stories: An Agile Introduction.* By Scott Ambler. Scott Ambler and Associates, Agile Modeling, 2015.
- [Scott Bellware, 2008] Scott Bellware (2008). *Behavior-Driven Development.* By Scott Bellware. Code Magazine (June 2008) Retrieved 12 August 2012.
- [Scott/Jeff, 2013] Scott/Jeff (2013). *Scrum Metrics for Hyperproductive Teams: How They Fly like Fighter Aircraft.* By Scott Downey, Jeff Sutherland. IEEE HICSS 46th Hawaii International Conference on System Sciences, Maui, Hawaii, 2013.
- [Scrum Alliance, 2005] Scrum Alliance (2005). *Article: Reporting Scrum Project Progress to Executive Management through Metrics.* By Brent Barton, Ken Schwaber, Dan Rawsthorne Contributors: Francois Beauregard, Bill McMichael, Jean McAuliffe, Victor Szalvay. Scrum Alliance.
- [Scrum Alliance, 2014] Scrum Alliance (2014). *Article: Velocity, How to Calculate and Use Velocity to Help Your Team and Your Projects.* By Catia Oliveira KING. Scrum Alliance. 6 February 2014.
- [Scrum Alliance, 2015] Scrum Alliance (2015). *Scrum Alliance* ([scrumalliance.org](http://scrumalliance.org)). Scrum-Alliance.
- [Scrum-Institute, 2015] Scrum-Institute (2015). *Scrum revealed: the only book can simply learn scrum!* by International Scrum Institute. [scrum-institute.org](http://scrum-institute.org) o ISI.
- [ScrumManager, 2014] ScrumManager (2014). *Gestión de proyectos Scrum Manager (Scrum Manager I y II)* By Juan Palacio. De la edición: Scrum Manager (Creative Commons Attribution Non-Commercial 3.0).

- [SEBoK, 2014] SEBoK (2014). *Guide to the Systems Engineering Body of Knowledge (SEBoK) v. 1.3. By Body of Knowledge and Curriculum to Advance Systems Engineering (BKCASE) project*. Sebokwiki.org, released 30 May 2014.
- [Sriram Narayan, 2015] Sriram Narayan (2015). *Agile IT Organization Design: For Digital Transformation and Continuous Delivery*. por Sriram Narayan. Addison-Wesley Professional. Release Date: June 2015. ISBN: 9780133903690.
- [Stefanini, 2013] Stefanini (2013). *Scrum of Scrums: Running Agile on Large Projects*. By Leandro Faria Stefanini. Scrum Alliance, 5 June 2013.
- [Steven Johnson, 2002] Steven Johnson (2002). *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. By Steven Johnson. Prentice Hall 2002.
- [SWEBOKv3, 2014] SWEBOKv3 (2014). *SWEBOK Guide V3.0, Guide to the Software Engineering Body of Knowledge Version 3.0*. Editors Pierre Bourque, École de technologie supérieure (ÉTS) Richard E. (Dick) Fairley, Software and Systems Engineering Associates (S2EA). JIEEE Computer Society, 2014.
- [Takeuchi/Nonaka, 1986] Takeuchi/Nonaka (1986). *The New New Product Development Game*. by Hirotaka Takeuchi, Ikujiro Nonaka. Harvard Business Review.
- [UNTREF, 2014] UNTREF (2014). *Construcción de software: una mirada ágil*. Por Nicolás Paez, Diego Fontdevila, Pablo Suárez, Carlos Fontela, Marcio Degiovannini, Alejandro Molina. Universidad Nacional de Tres de Febrero (UNTREF).
- [Victor Hugo Certuche, 2016] Victor Hugo Certuche (2016). *Article: Agile: Is Agile Really dead?* by Victor Hugo Certuche. Victor Hugo Certuche, Apr 28, 2016.
- [Wiki, 2015] Wiki (2015). *Wikipedia, 2015*. Online es.wikipedia.org.

- [Wiley/Sons, 2002] Wiley/Sons (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. By John Wiley and Sons. Paperback – March 21, 2002. ISBN: 0471202827.
- [Winston Royce, 1970] Winston Royce (1970). *Managing the Development of large Software Systems by Dr. Winston W. Royce*. Proceedings of IEEE WESCON 26 (August).

Los artículos y las ilustraciones de este libro se distribuyen bajo una licencia Creative Commons by-sa 4.0



[http://creativecommons.org/licenses/by-sa/4.0/deed.es\\_AR](http://creativecommons.org/licenses/by-sa/4.0/deed.es_AR)

Pueden ser copiados, distribuidos y modificados bajo las condiciones de reconocer a los autores y mantener esta licencia para las obras derivadas.

Impresión: Autores Editores S.A.S.,  
Bogotá D.C. - Colombia, Buenos Aires - Argentina.

La versión electrónica y el código fuente se publicaron en:  
<https://github.com/dariopalminio/ScrumEnIngenieriaDeSoftware>