

Scrum

en Ingeniería de Software

Dario Palminio

2019

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 3.9.12, 12 de Diciembre de 2019, 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

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

Me propuse ofrecer una guía para el conocimiento e implementación de una manera de trabajar y de gestionar proyectos de Ingeniería de Software llamada Scrum. Intento explicar al lector el sistema Scrum de un modo integrador, desde diferentes perspectivas y fuentes de conocimiento Scrum. De este modo trato de proporcionar un marco integral que incluya los principios del enfoque, estructura, procesos de Scrum y aspectos complementarios. Busco ayudar a los equipos en el avance de intentar emplear Scrum, a ejecutarlo correctamente para lograr alcanzar los resultados que aún no han obtenido. También pretendo 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.

Quiero resaltar que todo lo escrito aquí no es solo teoría basada en otros libros, sino que es producto de mi experiencia como Scrum Master, Ingeniero de Sistemas y Agile Coach en diferentes equipos y proyectos. Todas las práctica descritas aquí las he experimentado, en campo, y muchas opiniones y comentarios provienen de diferentes interacciones en proyectos reales en transformaciones digitales y adopciones de agilidad desarrolladas en distintas empresas.

Índice

1 Introducción	19
1.1 Consideraciones iniciales	19
1.1.1 Definición	19
1.1.2 Sobre si es una Metodología	20
1.1.3 Ámbito de aplicación	21
1.1.4 Visión general	22
2 Origen	25
2.1 Origen histórico	25
2.1.1 Organismos certificantes	26
2.2 Origen causal	27
2.2.1 Problema	27
2.2.2 Mentalidad criticada	29
2.2.3 Metodología criticada	31
3 Enfoque Scrum	41
3.1 Mentalidad y modelos mentales	42
3.2 Principios	43
3.3 Manifiesto Ágil	45
3.3.1 Valores del Manifiesto Ágil	45
3.3.2 Principios del Manifiesto Ágil	47
3.3.3 Corazón de la agilidad	49
3.4 Valores de Scrum	50
3.5 Principios de Scrum	52
3.6 Ideas relacionadas	56
3.6.1 Emergentismo	57

3.6.2	Sinergia	58
3.6.3	Ley de Linus	58
3.6.4	Sabiduría de multitud	59
3.6.5	Arquitectura emergente	61
3.6.6	Resiliencia	62
3.6.7	Prácticas emergentes	63
3.6.8	Autoregulación en equipo	63
3.6.9	El flujo	64
4	Núcleo del sistema Scrum	67
4.1	Estructura del sistema Scrum	68
4.2	Sistema de roles	69
4.2.1	Equipo de desarrollo	71
4.2.2	Product Owner	73
4.2.3	Scrum Master	75
4.3	Proceso Scrum	81
4.3.1	Reuniones principales	82
4.4	Flujo de artefactos	88
4.4.1	Definición de terminado	89
4.5	Reglas y consideraciones	90
5	Gestión de proyectos	93
5.0.1	Proyecto Scrum	93
5.0.2	Inicio	94
5.0.3	Planificación y ciclo de vida	96
5.0.4	Plan de ruta	97
5.0.5	Planificación de entregables	98
5.0.6	Triángulo de la gestión de proyectos	100
5.0.7	Priorización trade-off	101
5.0.8	Gestión orientada a producto	102
5.0.9	Gestión de riesgos	106
6	Métricas	109
6.0.1	Métricas de equipo	110
6.0.2	Métricas del proceso de trabajo	112
6.0.3	Métricas de calidad de producto	122
6.0.4	Métricas de resultados de negocio	124
6.0.5	Dashboard	130

6.0.6	Métricas ágiles	133
7	Escalamiento	135
7.1	Formación de equipos	136
7.1.1	Equipos de características	137
7.1.2	Equipos de componentes	138
7.1.3	Equipos mixtos	140
7.2	Integración	140
7.2.1	Scrum de Scrum	140
7.2.2	Scrum de Scrum de Scrum	141
7.2.3	Comunidades	142
7.2.4	Integración UX	142
7.2.5	Scrum y DevOps	144
7.3	Modelos de escalamiento	146
7.3.1	Serum@Scale	146
7.3.2	Spotify	147
7.3.3	SAFe	149
7.3.4	LeSS	149
7.3.5	Modelo Nexus	150
7.3.6	DAD	150
7.4	Equipo de mejora continua	151
8	Complementos de Scrum	153
8.1	Scrum extendido	155
8.1.1	Definición de preparado	157
8.2	Dinámicas de grupo	158
8.2.1	Coordinar el silencio	158
8.2.2	Gamification y dinámicas de grupo	158
8.2.3	Constitución de un equipo Ágil	158
8.3	Técnicas de justificación de negocio	161
8.3.1	Retorno de la inversión	161
8.3.2	Valor de negocio versus complejidad	161
8.4	Técnicas para requerimientos	162
8.4.1	Historias de usuarios	162
8.4.2	Slicing	171
8.5	Técnicas de estimación	172
8.5.1	Estimación relativa	173
8.5.2	Poker de planeación	175

8.6	Técnicas de monitoreo visual	177
8.6.1	Gestión visual	177
8.6.2	Tableros Scrum/Kanban	177
8.6.3	Tablero de obstáculos	182
8.6.4	Calendario de obstáculos	183
8.6.5	Gráficos de esfuerzo pendiente	183
8.6.6	Gráficos de velocidad	187
8.7	Técnicas de programación	188
8.7.1	Técnica Pomodoro	188
8.7.2	Calidad integrada	188
8.7.3	Programación de a pares	189
8.7.4	Revisión de a pares	190
8.7.5	TDD	190
8.7.6	BDD	191
8.7.7	ATDD	192
8.7.8	Integración continua	193
8.7.9	Entrega y despliegue continuo	193
8.8	Metodologías amigas	195
8.8.1	Scrum con XP	195
8.8.2	Scrum con Lean	196
8.8.3	Ampliar Scrum con Kanban	197
8.8.4	Ampliar Kanban con Scrum, Scrumban . .	200
8.9	Checklists	203
8.9.1	Planning Checklist	203
8.9.2	Refinement Checklist	204
8.9.3	Review Checklist	204
8.9.4	Retrospective Checklist	206
9	Ingeniería de Software Ágil	207
9.1	Proceso de desarrollo	207
9.2	Testing continuo	209
9.2.1	Tamaños de Testing	212
9.3	Prácticas técnicas	214
9.4	DevOps	216
9.4.1	Plataforma	216
9.4.2	Infraestructura	218
9.4.3	Pipeline automatizado	223
9.4.4	Activadores de características	226

9.5 En síntesis	228
10 Malas y buenas prácticas	231
10.1 Malas prácticas	231
10.2 Recomendaciones	239
11 Conclusión	241
12 Glosario y Acrónimos	245

Listado de figuras

1.1	Scrum en el Modelo Cynefin	22
1.2	Mapa mental sobre Scrum	23
2.1	Modelo Cascada de desarrollo	32
2.2	Cono de la incertidumbre en contextos inestables .	34
2.3	Marco de Gestión de Proyectos Clásica PMI	35
2.4	Ciclo de vida en una Gestión de Proyectos tradicional	36
2.5	Ejemplo de entrega de valor en proyectos tradicionales en cascada que fracasan en entregar valor en la fecha límite y entregan todo de una vez	37
2.6	Metodología Cascada criticada por Scrum	40
3.1	Mentalidad y versus prácticas	43
3.2	Valores ágiles	45
3.3	Emblema ágil	50
3.4	Los 3 pilares de Scrum (Scrum.org)	53
3.5	Emblema ágil para Scrum	56
3.6	Diagrama de ideas relacionadas al enfoque	56
3.7	Modelo de emergentismo	57
3.8	Inteligencia enjambre	60
3.9	Modelo del espectro del tipo desarrollo de software	61
3.10	Gráfico de “equipo tradicional de mando y control” y “equipo ágil autoregulado”	65
3.11	Sistema fluyente	65
3.12	Flujo de desarrollo	66

4.1	Diagrama del Núcleo del Sistema Scrum	68
4.2	Flujo de Scrum básico según la Guía de Scrum de Scrum.org de 2017	69
4.3	Diagrama del Sistema de Roles Scrum	71
4.4	Aspectos de un SM según Scrum.org	76
4.5	Mapa mental resumen didáctico del foco según las responsabilidades de los roles.	80
4.6	Diagrama de Flujo de Datos del Proceso Scrum .	82
4.7	Evento de planeamiento	83
4.8	Daily	85
4.9	Revisión	87
4.10	Retrospectiva	88
4.11	Diagrama de Flujo de Stock de artefactos Scrum .	89
4.12	Ejemplo de una DoD	90
4.13	Tiempos máximos Time-box segun Scrum Guide 2017.	91
4.14	Tamaño de equipo óptimo según Putnam.	92
4.15	Diagrama de flujo de estados de un PBI o historia en el flujo de trabajo de desarrollo de software (ejemplo)	92
5.1	Ciclo de vida de un proyecto Scrum.	94
5.2	Proyecto Scrum	96
5.3	Niveles de planificación	97
5.4	Roadmap ejemplo	98
5.5	Release plan en Sprints ejemplo	100
5.6	Triángulo de Gestión de Proyectos Scrum	101
5.7	El MVP	103
5.8	Evolución de una bicicleta	104
5.9	Ajuste de mercado, MVP y MMP	104
5.10	Crecimiento del producto	105
5.11	Llevar trabajo a los equipos de larga duración . .	106
5.12	Diagrama de riesgos, problemas e ítems del proyecto (PBIs).	107
5.13	Estados posibles de un riesgo.	108
6.1	Modelo de 4 categoría de métricas o KPIs.	110
6.2	Medida de satisfacción	111

6.3	Gráfico de velocidad, capacidad y velocidad promedio	116
6.4	Datos del gráfico de velocidad, capacidad y velocidad promedio	116
6.5	Gráfico de factor de foco	117
6.6	Gráfico de efectividad	118
6.7	Datos consolidados	118
6.8	Esquema del Delivery Frequency	119
6.9	Gráfico de Stories Inventory	120
6.10	Datos consolidados del gráfico Stories Inventory . .	120
6.11	DevOps pipeline y sus métricas	121
6.12	Gráfico de deuda técnica acumulada versus resuelta	122
6.13	Métricas vs. KPIs	125
6.14	Embudo de conversión (conversion funnel)	127
6.15	Modelo de KPIs de impacto de negocio.	129
6.16	KPIs del dashboard del gobierno de australia . .	130
6.17	Dashboard C-SAT ejemplo	131
6.18	Televisor con KPI relevantes para un equipo. . . .	131
6.19	Monitoreo de Dashboards	132
6.20	Un equipo (Sierra India en LATAM Airlines) con su Dashboard visible	132
6.21	Métricas en Lean Startup	133
7.1	Equipo Scrum orientado por misión	136
7.2	Esquema de equipos de características	138
7.3	Esquema de equipos de componentes	139
7.4	Ejemplo de equipos mixtos.	140
7.5	Ejemplo de comunidades	142
7.6	Ejemplo de coordinación entre flujos de tarea UX e historia	144
7.7	Ejemplo de integración de equipos en DevOps . .	145
7.8	DevOps=Dev+Tester+Ops	145
7.9	Modelo Scrum@Scale	147
7.10	Modelo organizacional Spotify	148
8.1	Flujo de Extended Scrum	155
8.2	Ejemplo de una DoR	157
8.3	Lienzo de Constitución de Equipo Ágil	159
8.4	Diagramas de Business Value	161

8.5 Escritura de historias ejemplo	163
8.6 Clasificación de historias	165
8.7 Historia de usuario como ejemplo simple	169
8.8 Modelo conceptual del backlog	170
8.9 Desglose del backlog	171
8.10 Actividad de Slicing	172
8.11 Estimación	172
8.12 Estimación relativa	174
8.13 Ejemplo de un tablero Kanban para Scrum.	178
8.14 Ejemplo de un tablero Kanban para historias.	179
8.15 Ejemplo de un tablero Kanban físico que usamos con un equipo (Sierra India) en LATAM Airlines. .	179
8.16 Ejemplo de un tablero Kanban para Scrum sofisticado.	180
8.17 Esquema de un Sprint backlog.	181
8.18 Ejeplo de tableros físicos en LATAM Airlines 2017.	181
8.19 Ejemplo de un tablero integrado de historia y tareas.	182
8.20 Ejemplo de un tablero de Obstáculos.	183
8.21 Calendario de obstáculos ejemplo.	183
8.22 Ejemplo de un gráfico Burndown con un sprint de dos semanas y 20 tareas comprometidas.	184
8.23 Ejemplo de cómo llevar un Burndown con un sprint de 5 días y quedando 3 storypoints por quemar. .	185
8.24 Ejemplo de un gráfico Burnup de tareas comprometidas en un sprint de dos semanas.	186
8.25 Ejemplo de cómo llevar un Burnup con un sprint de 5 días y 10 puntos quemados al cuarto día. . . .	186
8.26 Diagrama de Velocidad y Diagrama de Velocidad con Velocidad Promedio.	187
8.27 Esquema pomodoro	188
8.28 Procesos de Test Driven Development	191
8.29 Esquema del proceso básico de ATDD	193
8.30 Continuous Delivery vs. Continuous Deployment .	194
8.31 Scrum más XP	196
8.32 Scrum más Kanban	199
8.33 Tablero Scrum-Kanban ejemplo para un equipo de desarrollo	200

8.34 Tablero Scrumban y CFD ejemplo para un equipo de desarrollo	202
9.1 Ciclo de ingeniería de Software	208
9.2 Ingeniería de Software Ágil	209
9.3 Modelo en V de calidad y testing	210
9.4 Ciclo de vida de una historia y prácticas	211
9.5 Prácticas básicas en continuous testing	211
9.6 Tamaños de testing en el pipeline del Continuous Delivery	213
9.7 Tamaños de testing y tipos en el pipeline del Continuous Delivery	213
9.8 Flujo de prácticas técnicas	215
9.9 Flujo de producto vs. de delivery	215
9.10 Plataforma e infraestructura	216
9.11 Ambientes del flujo de desarrollo	219
9.12 Infraestructura	221
9.13 Ejemplo de un CI-CD pipeline para una App Java con arquitectura de Microservicios	224
9.14 Script ejemplo de un pipeline	225
9.15 Esquema de un CI-CD pipeline tipo small-medium-large test	226
9.16 Esquema de feature toggle	227
11.1 Agile Core	241
11.2 Resumen de la Guía de Scrum 2017	243

Capítulo 1

Introducción

1.1 Consideraciones iniciales

El marco de trabajo que se presenta en este libro parte de una cultura de solución de problemas adaptativa que apoya el desarrollo de productos y servicios, posibilitando la búsqueda de mejores formas de aprender y crear valor en ambientes laborales más cálidos y humanos. Al inicio del libro se relata el “qué” y el “porqué” de este marco, para luego pasar a contar el “cómo”, a un alto nivel y, desde esta perspectiva, ofrecer diferentes herramientas y prácticas que sirvan para ayudar a implementarlo y complementarlo con éxito.

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 SBOK¹, la ScrumAlliance² y Jeff Sutherland³). 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 en marcos de trabajo como la guía 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 prototípico de producción" [Ken Schwaber, 1995].

Por otro lado, considerando que Scrum define roles, artefactos, actividades, flujo del ciclo de actividades Scrum⁴, 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

¹[SBOK, 2013], SCRUMstudy (2015).

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

³[Jeff Sutherland, 2014b]

⁴[Agile Atlas, 2012]

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 diferentes pero compatibles con los lineamientos de Scrum (sus valores y principios) y del Movimiento Ágil (enfoque 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. Por eso, es recomendable que se lo denomine "marco de trabajo".

1.1.3 Ámbito de aplicación

Relacionado a su ámbito de aplicación, podemos analizar nuestro equipo o nuestra organización bajo el modelo Cynefin⁵. Con Cynefin podemos evaluar los múltiples factores en nuestro entorno y nuestra experiencia y determinar bajo qué dominio nos encontramos. Pues, Scrum no es un marco de trabajo orientado a implementarse en cualquier dominio y contexto. Scrum está pensado especialmente para proyectos bajo "dominios complejos" [Snowden 2007] (ver fig. 1.1) donde existe un grado alto de incertidumbre y baja predictibilidad y para sistemas adaptativos complejos, donde mismas causas pueden generar diferentes consecuencias o diferentes causas pueden generar mismas consecuencias. 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] y desarrollo de productos con requisito inciertos y cambiantes. Es decir, está orientado a contextos que necesitan niveles altos de creatividad, innovación, interacción y comunicación. Por este

⁵Cynefin: Snowden (1999), "Liberating Knowledge"; Snowden (2003). "The new dynamics of strategy: Sense-making in a complex and complicated world"

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.

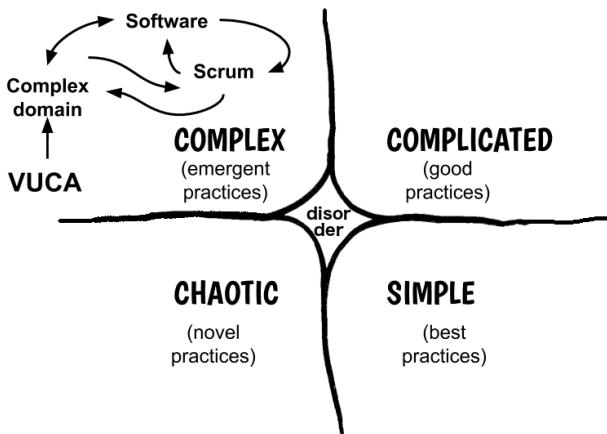


Figura 1.1: Scrum en el Modelo Cynefin

1.1.4 Visión general

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

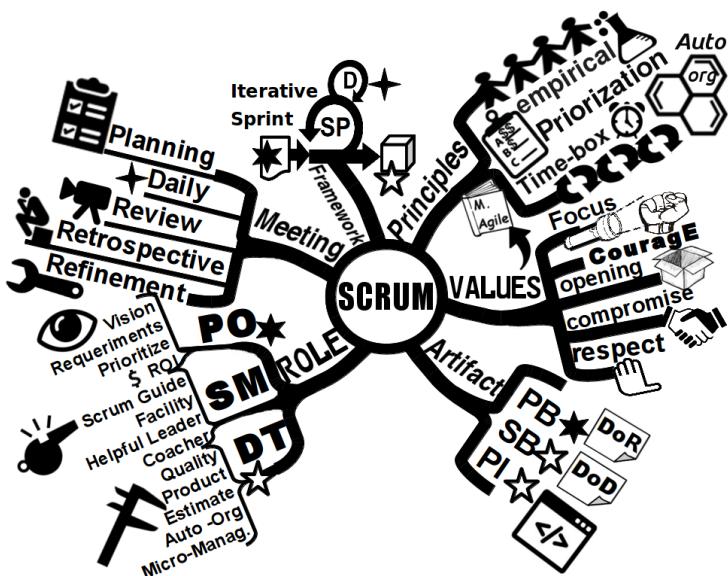


Figura 1.2: Mapa mental sobre Scrum

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 enfoque. Ken Schwaber fue el primer presidente de la Alianza Ágil fundada tras el Manifiesto Ágil.

2.1.1 Organismos certificantes

Desde que surgió y 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 básico 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 más reconocida organización certificante. La Scrum Alliance fue fundada en 2002 por Ken Schwaber, Mike Cohn y Esther Derby. Por otro lado, 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"). De este modo, Scrum.org se transforma en el principal órgano certificante y el libro "The Scrum Guide" (Jeff Sutherland y Ken Schwaber) la principal guía

de Scrum, lo que podemos llamar el núcleo de Scrum¹.

Hay otras organizaciones capacitadoras y certificadoras que no pertenecen al núcleo Scrum o al Scrum original. Como ScrumStudy, empresa (subsidiaria de VMEdú) que se basa en el libro "SBOK Guide" [SBOK, 2013], pero que está alejada del origen fundacional y de sus autores como Ken Schwaber².

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, se plantea un cambio paradigmático o de enfoque 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

¹Scrum.org (2018) certifica Professional Scrum Master (PSM), Professional Scrum Product Owners (PSPO) y Professional Scrum Developer (PSD).

²[Don Kim, 2016]

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 Mentalidad criticada

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

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

La mentalidad o cultura 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"³ que hace que respetemos o sigamos las opiniones de los expertos aunque las mismas estén equivocadas.

- **Cultura de Liderazgo:** También se critica la cultura basada en liderazgo de autoridad, proveniente del enfoque clásico de la administración (taylorismo y fayolismo), que han generado gestión de control y mando. En el ámbito empresarial y en cierta época, ha prevalecido el enfoque 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.

³Mecanismo 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.

- **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" y rígidas, también proveniente del enfoque clásico de la administración. 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 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 de los sistemas de "evaluación de desempeño individual" en vez de grupal⁴. 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 retroalimentación para la evaluación de desempeño son ciclos largos que parecen no ser realmente útiles⁵. También se suma la práctica de evaluación de desempeño verticalista, de arriba hacia abajo, en vez de en 360 grados. Cuando la evaluación es de arriba hacia abajo, se genera una cultura de obediencia que no favorece la creatividad ni la seguridad psicológica.

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

⁴Los métodos tradicionales de gestión de proyectos hacen hincapié en la responsabilidad individual hacia las responsabilidades del proyecto [SBOK, 2013].

⁵Las empresas Deloitte y Accenture comenzaron a desmantelar 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.).

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 2.6) de la industria, no al de gestión, y es simplemente un modelo, no una metodología.

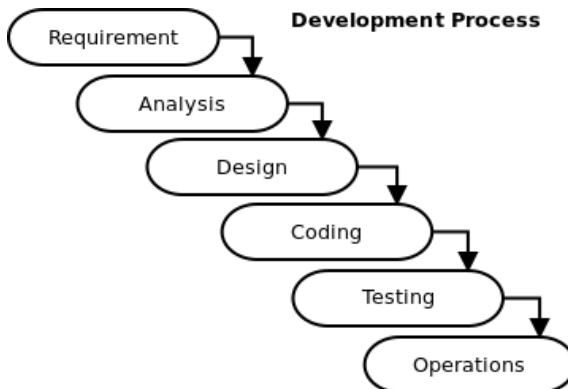


Figura 2.1: Modelo Cascada de desarrollo

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"⁶ 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 +- 400 por ciento y tiende a disminuir a lo largo del proyecto. Por tal motivo prever el proyecto en su primera parte no es efectivo y tomar decisiones difíciles de cambiar en esa etapa no sería conveniente.

⁶[McConnell, 2006], [Boehm, 1981], [Martin Alaimo, 2014]

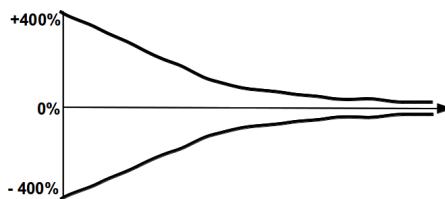


Figura 2.2: Cono de la incertidumbre en contextos inestables

2. Requerimientos no necesarios:

Cuando los requerimientos son solicitados 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 malalicitación de los mismos.

3. Fases separadas:

Cada fase es estrictamente separada [Scrum-Institute, 2015]. 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 "pesado" (estanco y burocrático) y prohibitivo para responder satisfactoriamente a los factores cambiantes de negocio [Martin Alaimo, 2014].

4. Equipos aislados

Al trabajar en fases separadas y con especialización del trabajo se llega a equipos especializados. Así es que podemos

llegar a tener a diversos equipos separados, como el equipo de negocio y descubrimiento, el de arquitectura, el de desarrollo (implementación), el de UX, el de QA (testing), el de operaciones y despliegue, el de seguridad, etcétera; y estos equipos pueden llegar a aislarse y a generar burocracia, cuellos de botella y lentitud en la dinámica de relación inter-equipo y en la administración de presupuestos y proyectos.

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:

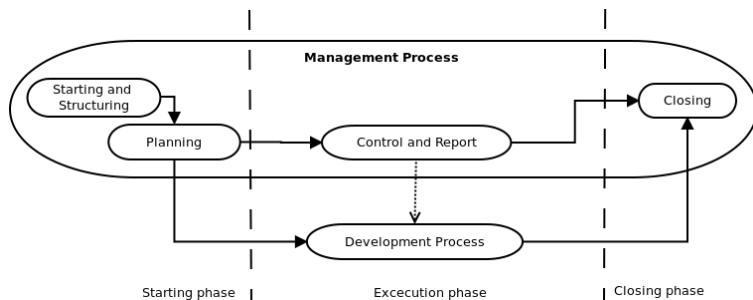


Figura 2.3: Marco de Gestión de Proyectos Clásica PMI

- 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, y luego durante la fase de ejecución 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

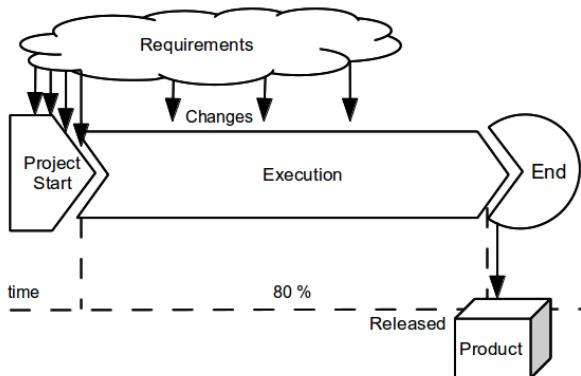


Figura 2.4: Ciclo de vida en una Gestión de Proyectos tradicional

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, alta variabilidad 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)⁷ y que se hace una planificación por adelantado, debido a que se considera que

⁷El 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.

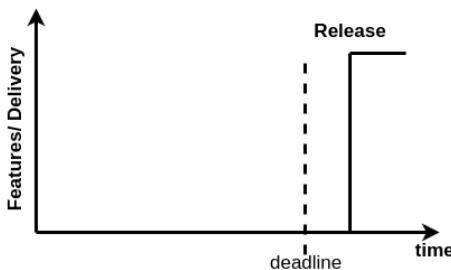


Figura 2.5: Ejemplo de entrega de valor en proyectos tradicionales en cascada que fracasan en entregar valor en la fecha límite y entregan todo de una vez

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 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 jefe de proyecto o gerente de proyecto tradicional o PM es contraproducente en un trabajo complejo y creativo [Ken Schwaber, 2011]. La dirección de un PM en base a un plan puede limitar la creatividad y la inteligencia del equipo en lugar de incentivarla para resolver mejor los problemas. Además, hay ciertas prácticas de los PMs que son consideradas negativas bajo un marco ágil, como pueden ser: considerar que la productividad aumentará ejerciendo presión o sumando personas, controlar el trabajo individual y la marcha de las tareas mediante diagramas de Gantt, planificar sin incluir a todos los responsables, centralizar las comunicaciones con otras áreas, ejercer excesivo foco en el proyecto más que en el producto, manejar en forma ligera y poco transparente la rotación de integrantes del equipo, premiar el trabajo individual más que el colectivo, dirigir el equipo y tomar decisiones por ellos, complejizar el trabajo de gestión y monitoreo de métricas innecesariamente, perder el foco en el equipo por lidiar con luchas de poder o políticas, no priorizar el trabajo en equipo, etcétera. Por otro lado, mantener una figura de poder (como la de un PM o jefe de proyecto) en un equipo, condiciona una cultura de obediencia. Si depende, en gran medida, de un PM la evaluación de desempeño, el aumento de sueldo, la rotación de funciones,

el prestigio, etc., entonces el integrante de un equipo se verá influenciado a la subordinación (como lo muestra el experimento de Milgram y otros estudios). Si se elimina la figura de PM y se delegan sus actividades de gestión en otros roles, junto con un enfoque de auto-organización y auto-regulación (monitoreo y control autónomo) del equipo, 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 PM, en una forma de liderazgo de mando y control y en jerarquías verticalistas. La excesiva centralización también centraliza demasiado las comunicaciones generando cuellos de botella. 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.

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 2.6). 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.

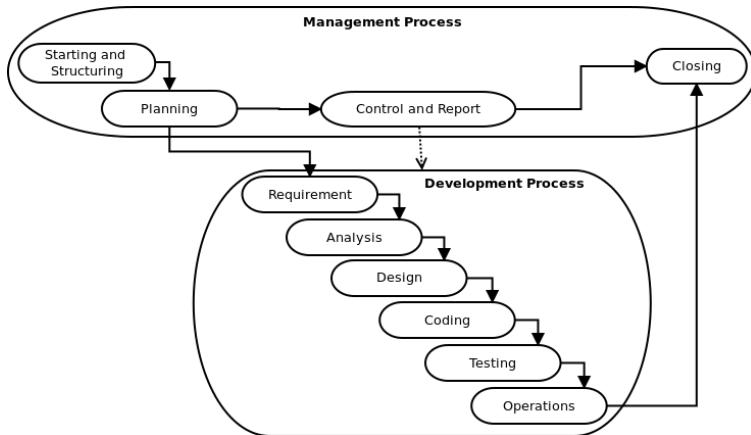


Figura 2.6: Metodología Cascada criticada por Scrum
(Diagrama integrador del Modelo Cascada
[Winston Royce, 1970], la Metodología Cascada
[Ken Schwaber, 1995] y la Metodología de Gestión de Proyectos
[PMBOK, 1996])

Capítulo 3

Enfoque Scrum

Discriminando todas las filosofías generales, como ideologías y concepciones religiosas, podemos decir que hay perspectivas filosóficas o enfoques relacionados, 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 pseudofilosofías 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 tratases 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” (perspectiva Ubuntu o cooperativa), escuchamos en una reunión técnica que “el software debe ser libre” (perspectiva Free Software) o que solo se desarrollara software propietario, que “las mejores arquitecturas, requisitos y diseños emergen de equipos

autoorganizados” [Beck, 2001] (perspectiva ágil); o que “el software es un mundo de objetos” (perspectiva 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 o enfoques que condicionan a las personas y al desarrollo de software. Por ejemplo, en particular, se da con las filosofías del pensamiento sistémico, pensamiento crítico, enfoque ágil, software libre, software abierto o software propietario, que son las principales que en el mundo del desarrollo de software influencian a sus actores. Estas perspectivas filosóficas suelen presentar principios a seguir y estos principio guían algunas metodologías (prácticas y métodos) de desarrollo de sistemas y a gran parte de desarrolladores de sistemas. Aquí nos vamos a centrar en las perspectivas Scrum y 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 ¹). O sea que la filosofía o perspectiva filosófica o ideológica 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.

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

¹Masa K Maeda es PhD, Founder and CEO de Valueinnova USA (capacitación Scrum).

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 un enfoque o una filosofía sin convicción es un primer condicionante de fracaso práctico en su implementación.

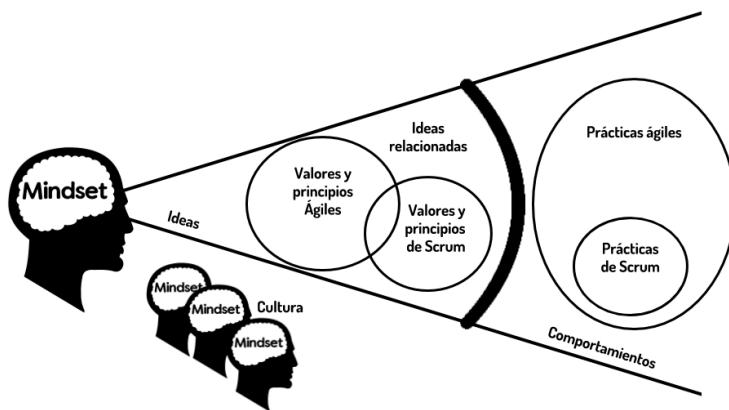


Figura 3.1: Mentalidad y versus prácticas

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 [Manifesto Agile 2001], el que expone valores y principios firmados el 2001 por diecisiete líderes de metodologías livianas, convocados por un ingeniero de software. Los principios del desarrollo ágil surgieron como síntesis de lo que consideraron central en esas metodologías livianas (ágiles) y que estaban surgiendo como alternativa a las metodologías clásicas y 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:

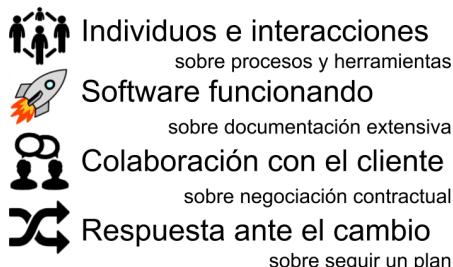


Figura 3.2: Valores ágiles

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 buscar 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 principios:

1. **Entregar valor temprano y continuamente:** Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor. Tratamos de entregar lo más rápido posible valor funcional.
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.
4. **Cooperación con el cliente:** Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto, potenciando de esta manera al equipo. 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. Un equipo que es libre de decidir sobre cómo hacer su trabajo y trabaja en un ambiente de seguridad psicológica, es bastante probable que sea un equipo motivado.
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. La priorización de conversar en persona, por sobre las conversaciones diferidas, acelera la toma de decisiones y la solución de impedimentos.
7. **Orientación al producto:** El producto funcionando es la medida principal de progreso. Es más importante que un producto funcione como se espera más que cumplir con un hito, medirlo o documentarlo.
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 al enfoque ágil.

11. **Equipos auto-organizados:** Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados (alineado al principio sistémico de emergencia). Los equipos auto-organizados son aquellos que tienen proactividad y autonomía para tomar decisiones de producción y gestión, como también así la de ejercer la auto-regulación mediante el monitoreo y control de sus actividades y resultados. Estos equipos tienen mayor libertad para crear.
12. **Equipos auto-reflexivos:** A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para, a continuación, ajustar y perfeccionar su comportamiento en consecuencia. Los equipos auto-reflexivos son los que evolucionan por medio de procesos de mejora continua constante.

3.3.3 Corazón de la agilidad

Como se observa anteriormente, los valores y principios suman 16 incisos y son bastantes para ser recordados fácilmente. Para ayudar en su aprendizaje, se puede recurrir a unificarlos en un mandala ágil o en un emblema con los cuatro conceptos claves, corazón de la agilidad: colaboración, orientación a la entrega (flujo de valor), adaptación y mejora continua (ver fig. 3.3).

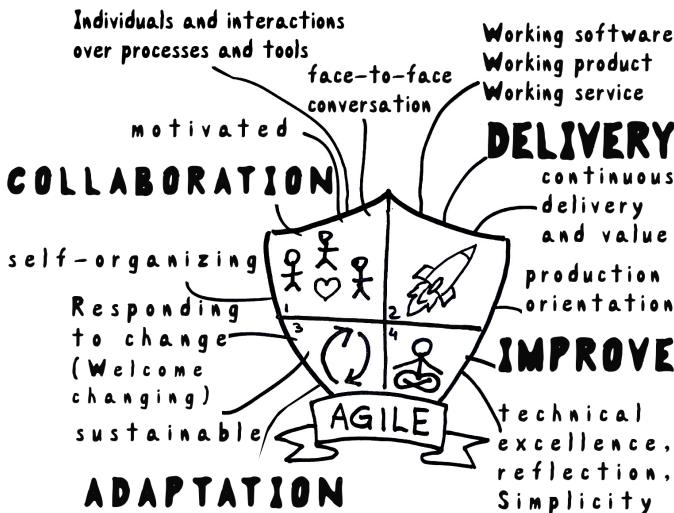


Figura 3.3: Emblema ágil

3.4 Valores de Scrum

El corazón de Scrum es el control empírico sostenido por la transparencia, la inspección y la adaptación². Aunque, sin embargo, la Scrum Alliance propuso cinco valores esenciales a seguir³:

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. Si bien Scrum no prescribe ninguna especificación detallada de foco, podemos entender como foco a: no desviarse del objetivo del producto y de iteración, buscar hacer trabajo (PBIs) alineado al objetivo de la iteración primero, hacer las tareas más prioritarias y en progreso, mantener el trabajo en progreso con una cantidad de

²[Ken/Jeff, 2013]

³[Scrum Alliance, 2015]

trabajo óptima para el equipo en la iteración, desarrollar conversaciones sin ruido comunicativo, ordenada, y hacer que los eventos y reuniones sean eficaces, manteniendo el foco en el objetivo de la reunión y en su éxito.

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 ágil y el enfoque de Scrum. Coraje para hacer bien las cosas y para trabajar en los problemas difíciles.
3. **Apertura:** Hay que tener apertura, actitud abierta y receptiva, 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. También significa franqueza y sinceridad. Franqueza que nos hace ser transparentes con el trabajo que hacemos, con el progreso, con el feedback y el conocimiento que tenemos.
4. **Compromiso:** Se busca lograr compromiso para el éxito gracias a promover el mayor control nuestro sobre lo que hacemos y nuestro destino. Compromiso para hacer el máximos esfuerzo posible para alcanzar los objetivos del equipo. Hay mayor probabilidad de lograr compromiso en las personas que deciden sobre lo que hacen. Nos debemos comprometer a ser autónomos, transparentes y colaborativos.

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. Respetamos el conocimiento, las habilidades y la experiencia profesional de nuestros compañeros. Buscamos conocer y comprender a nuestros compañeros, para saber hasta donde nuestra franqueza puede llegar, con respeto, a crear relaciones positivas, conversaciones constructivas y colaboración abierta. Equipo Scrum se respetan entre sí para ser personas capaces e independientes.

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 (auto-regulación). El proceso empírico permite adaptabilidad a requisitos que emergen del mismo proceso de desarrollo. Con este principio como marco, podemos usar metodologías, prácticas, 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.

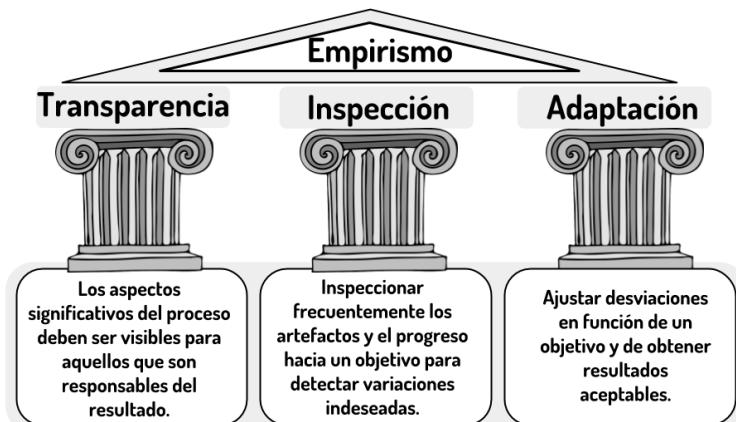


Figura 3.4: Los 3 pilares de Scrum (Scrum.org)

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.

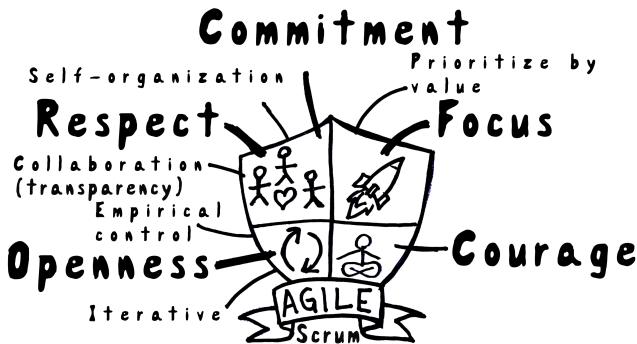


Figura 3.5: Emblema ágil para Scrum

3.6 Ideas relacionadas

A continuación algunas ideas relacionadas al enfoque ágil y a la mentalidad buscada bajo Scrum.

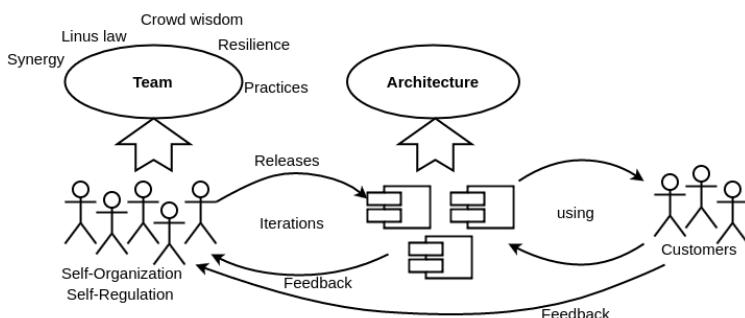


Figura 3.6: Diagrama de ideas relacionadas al enfoque

3.6.1 Emergentismo

Scrum adhiere al principio de auto-organización y está alineado al Manifiesto Ágil y a su enfoque. En el enfoque 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"⁴ que significa que desde un nivel de realidad dado (n) donde componentes se interrelacionan ($S_1, S_2, S_3 \dots S_n$) pueden emergir 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 3.7).

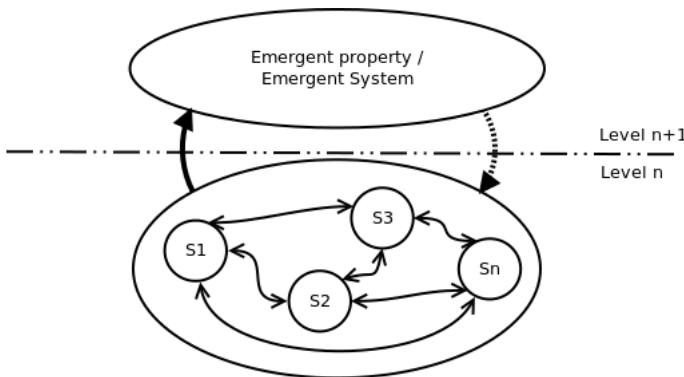


Figura 3.7: Modelo de emergentismo

En este sentido suele llamarse fenómeno emergente 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

⁴"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.

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 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 resuelto 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]⁵. Es más, a medida que el grupo es más grande, las decisiones o predicciones son más acertadas.

Claro que esto no sucede fácilmente, pues puede surgir todo lo contrario, como fallos de la inteligencia colectiva o irracionalidad colectiva, conformidad de grupo o sometimiento a la autoridad. Para propiciar la inteligencia colectiva se deben cumplir ciertas condiciones básicas. Para Surowieki (escritor del libro La Sabiduría de las Multitudes⁶) 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 irrationales. 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

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

⁶[James Surowiecki, 2005]

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 decisión métrica grupal se basa en el promedio de todas las decisiones⁷.

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

La sabiduría de multitud es una forma de inteligencia enjambre. En el trabajo en equipo colaborativo, la inteligencia enjambre es el santo grial a lograr. Como una bandada de aves, como bandadas de ibis eremitas, donde cada miembro es consciente de dónde está en relación a los otros, sin necesidad de un director, y se ubica y sincroniza de la mejor manera posible para que todo el equipo sea óptimo y mejor, logrando como equipo un mejor resultado que dirigidos por un experto.



Figura 3.8: Inteligencia enjambre

⁷Un 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.

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 3.9) [Neal Ford, 2010]. La otra forma es no anticipar nada, permitiendo que las soluciones software surgen 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 3.9). 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.

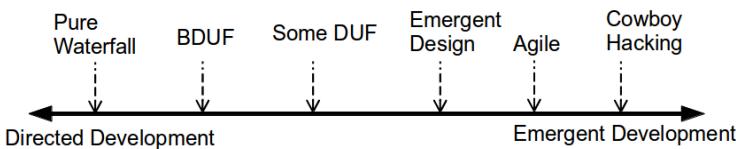


Figura 3.9: Modelo del espectro del tipo desarrollo de software (Espectro que abarca el diseño [Neal Ford, 2010])

Hay una obra literaria llamada "La catedral y el bazar"⁸ escrita por el hacker Eric S. Raymond en 1997 que expone esta idea. La misma analiza dos modelos de producción de software: la catedral

⁸[Eric Raymond, 1997]

que representa el modelo de desarrollo más hermético y vertical (el lado izquierdo de la figura 3.9) y por otro lado el bazar, con su dinámica horizontal y bulliciosa (el lado derecho de la figura 3.9).

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 del enfoque ágil que en muchas ocasiones se hace hincapié, aunque generalmente no se expicie, 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 productiva⁹. 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, en algunos momentos 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 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. Es decir, con Scrum se

⁹[OPS OMS, 1998]

promueve el desarrollo de la mentalidad resiliente para evitar la desmoralización del equipo por un cambio o falla, puesto que se reconoce que no se puede predecir a priori el resultado de un camino u otro y hay que estar abierto a los cambios o condiciones desfavorables para actuar, en respuesta, de la mejor forma posible.

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.

3.6.8 Autoregulación en equipo

En los equipos tradicionales de mando y control, el jefe o el PM es quien monitorea y ajusta autoregulando al equipo (ver

gráfico izquierdo de la figura 3.10). Recordemos que, desde la perspectiva cibernetica, la autoregulación es el proceso de ciclos de retroalimentación negativa, en donde mediante el monitoreo del entorno, se acciona para cumplir los objetivos del sistema, a pesar de las perturbaciones del ambiente. Desde este punto de vista, el jefe o PM es un sistema de control encargado de la autorregulación y los miembros del equipo son agentes pasivos a la autorregulación, son operarios. Pues, la agilidad vino a cambiar esta dinámica de trabajo. Si pensamos en nuestro organismo, las células sanas de nuestro cuerpo se autoregulan sin necesidad de dirección del cerebro y, sin embargo, son partes de nosotros. Los equipos ágiles en una organización se deberían comportar como las células sanas, que ante situaciones caóticas se adaptan manteniendo cierta estabilidad y el rumbo de sus finalidades. Ken Schwaber describe a Scrum como “caos controlado”, en el sentido de auto-controlado. Desde esta óptica, se entiende al desarrollo de software ágil como una serie de inspecciones constantes acompañadas de correcciones inmediatas, tanto en procesos internos como externos, monitoreando el entorno y ajustando en respuesta, en equipo colaborativo. Schwaber llama a esta forma de trabajar el proceso caórdico¹⁰, y nota que espontáneamente, los equipos ágiles de trabajo se coordinan con muy poca intervención de entidades externas autoregulándose (ver gráfico derecho de la figura 3.10).

3.6.9 El flujo

La idea del flujo ronda el enfoque ágil, ya sea con Lean, Kanban y también con Scrum. En esta vía, Jeff Sutherland dijo: “Scrum se trata acerca de permitir el mayor flujo posible”. Con este marco lo que se busca es que las personas fluyan sin gran esfuerzo, en un flujo de trabajo lo menos perturbado posible, en un proceso limpio, liviano y orgánico.

Las habilidades y las destrezas de un equipo, como de sus integrantes, deben estar en equilibrio con el reto de lo que se tiene que hacer. Cuando eso se logra, se alcanza una armonía de un

¹⁰Caórdico es una organización o sistema social auto-catalítico, autoregulador, adaptativo, no-lineal y complejo, cuyo comportamiento exhibe armoniosamente características tanto de orden como de caos.

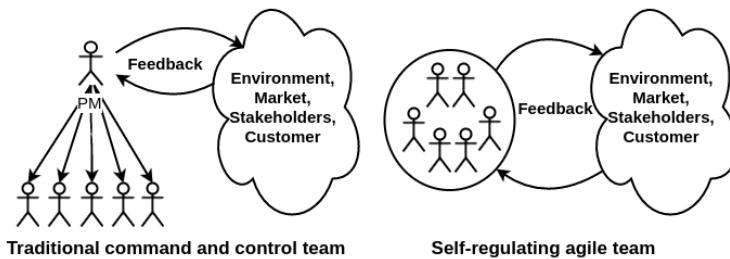


Figura 3.10: Gráfico de “equipo tradicional de mando y control” y “equipo ágil autoregulado”

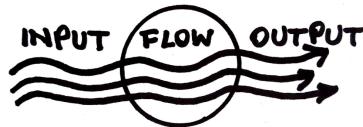


Figura 3.11: Sistema fluyente

proceso fluyente, con ritmo, con cadencia. Cuando se habla de cadencia se refiere a esta idea de fluir con ritmo. Desde un punto de vista sistémico, equivale a decir que debe existir cierta relación proporcional y estable entre el promedio de la tasa de entrada al sistema con el promedio de la tasa de salida (promedio de la velocidad), en relación al tiempo. Para que ocurra el sistema debe comportarse como un flujo de trabajo con personas jalando tareas a un ritmo sostenible y armónico. Y para fluir con ritmo es necesario limpiar el proceso, es decir, hacerlo magro: eliminar desperdicios, diluir problemas, mitigar riesgos, evadir obstáculos, etcétera. Un líder de Scrum es un facilitador, alguien que facilita el flujo. Y para facilitar el flujo es necesario, entre otras cosas, disciplina. El marco Scrum es el marco disciplinario que encauza el flujo.

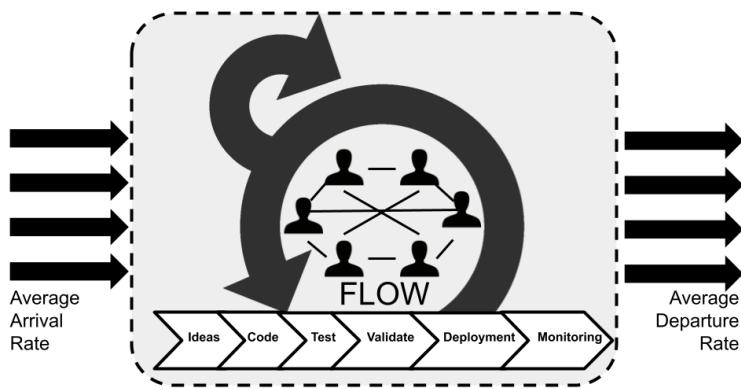


Figura 3.12: Flujo de desarrollo

Capítulo 4

Núcleo del sistema Scrum

Scrum es un sistema compuesto por principios y valores Scrum (el enfoque), como parte del sistema cultural, y tres conjuntos de componentes interrelacionados que forman el "Núcleo del Sistema Scrum"¹ (ver figura 4.1). Los tres conjuntos de componentes son: roles, eventos 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. Los eventos o actividades conforman la parte principal del sistema de procesos Scrum (Proceso Scrum) que determina las relaciones de organización entre eventos, 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 eventos o actividades regulares que permiten la repetibilidad rítmica (predictibilidad del flujo de trabajo) bajo un grado de prestancia.

¹El 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].

A continuación se describirán estos componentes y sus relaciones.

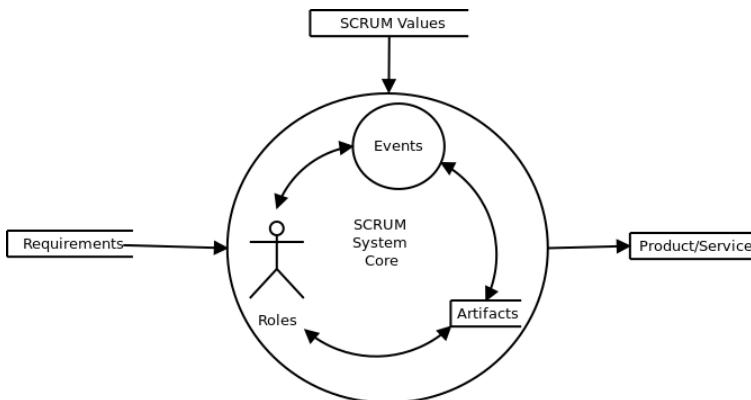


Figura 4.1: Diagrama del Núcleo del Sistema Scrum

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 en la iteración) y el Incremento de Producto potencialmente entregable (lo que logramos hacer y que se puede entregar). Estos artefactos son tratados en un flujo de trabajo en el cual se construyen productos en forma incremental y evolutiva, en una serie de ciclos cortos de tiempo llamados Sprints.

En cada ciclo Sprint del flujo de trabajo, se practican seis eventos Scrum: refinamiento de producto, planificación, reunión diaria o Daily, desarrollo (construcción), 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"² (aunque no se aconseja usar la palabra grooming), Mantenimiento de Backlog o Refinamiento (Refinement). Salvo el desarrollo, los eventos se consideran actividades o reuniones Scrum. El desarrollo o ejecución 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.

La síntesis del flujo de Scrum es como el mostrado en la figura siguiente:

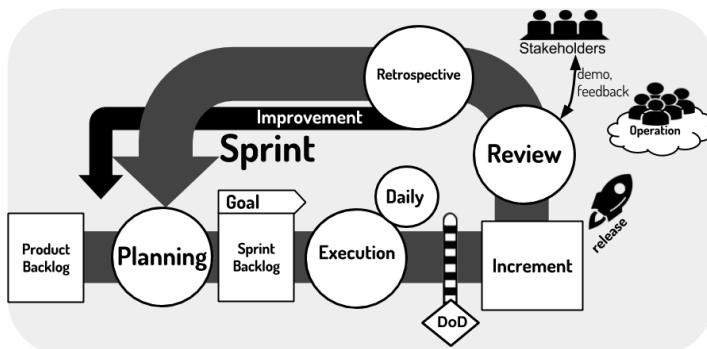


Figura 4.2: Flujo de Scrum básico según la Guía de Scrum de Scrum.org de 2017

4.2 Sistema de roles

El Sistema de Roles (ver figura 4.3), en el núcleo de Scrum, es el conjunto de roles y relaciones parte del sistema. 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,

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

Miembros del Equipo de Desarrollo o Desarrolladores Scrum). También hay roles secundarios y externos, que aunque no son roles de Scrum es útil identificarlos, como el de Stakeholder, expertos (equipo extendido y SME), vendedores y/o cuerpo de asesoramiento de Scrum (Agile Coaches) o grupo de trabajo del negocio. El rol Stakeholder es el más importante de los roles secundarios e incluye a: los clientes y usuarios, los dueños de unidades de negocio (Business Owner) quienes son responsables de la operación del producto y los patrocinadores (Business Sponsor) quienes financian el producto o negocio³.

Hay que tener en cuenta que este marco 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 tiene 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.

³[SBOK, 2013]

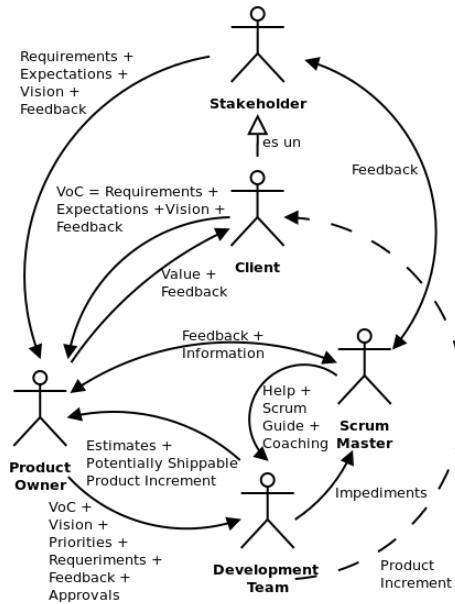


Figura 4.3: Diagrama del Sistema de Roles Scrum

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.

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

- Respeta, entiende y sigue los valores ágiles y de Scrum.
- Crea y gestiona el desarrollo de software comprometiéndose a entregas o 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 100% en el proyecto.
- Debe priorizar y promover la comunicación cara a cara.
- Provee estimaciones de ítems de trabajo en forma colaborativa.
- 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.
- Toma las mejores decisiones posibles para asegurar el progreso hacia la meta del Sprint, alineándose con el PO cuando esté disponible.
- Monitorea el progreso y el éxito con el resto de los roles del Scrum Team.
- Debe gestionar los riesgos y problemas junto con los demás roles. En esta vía debe levantar y buscar mitigar los riesgos técnicos.
- No debe ocultar impedimentos ni información.
- Debe evitar 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.
- Busca mantener el foco en los eventos orientados a cumplir los objetivos de la reuniones.
- Busca mantener el foco en el sprint haciendo que el equipo priorice y trabaje en sólo unos pocos elementos a la vez, acorde a la capacidad del equipo, en lugar de tener muchos trabajos abiertos en paralelo que pueden hacer peligrar el objetivo del sprint.
- Respeta los acuerdos de trabajo del equipo.

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 PO se deben tener en cuenta las siguientes afirmaciones:

- Respeta, entiende y sigue los valores ágiles y de Scrum.
- Debe colaborar con el Scrum Master y con el Equipo de Desarrollo. Colabora con frecuencia con el equipo para poder tomar decisiones informadas para equilibrar el esfuerzo y el valor de los elementos pendientes del Backlog. También colabora con el equipo para que construyan incrementos teniendo en cuenta las preocupaciones del usuario final y las partes interesadas.
- Es quien busca determinar el mejor producto a conseguir.

- Es responsable de desarrollar una visión, comunicarla y promoverla.
- Dirige la estratégica del producto relacionada, entre otras cosas, a la penetración en el mercado, evolución y madurez del producto en el mercado y generación de nuevos mercados.
- Es responsable del caso de negocio, retorno de la inversión ROI y de los resultados empresariales, evaluando continuamente el impacto en el negocio y buscando maximizar el valor.
- Trata de que se libere software a menudo, actualizando los indicadores clave de producto (KPI) después de cada lanzamiento y usando esta información para ajustar el trabajo en el Backlog. De esta manera busca que el producto creado a través de Scrum sea exitoso.
- Es quien provee las "hipótesis requerimientos"⁴, por lo que es responsable de entenderlas, escribirlas y transmitirlas en forma eficaz.
- Es dueño del artefacto Product Backlog y responsable de su orden, priorización y detalle.
- 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 (representa la voz del cliente).
- Busca la alineación con otros Product Owners cuando sea necesario desde una perspectiva general de producto, empresa o cliente.

⁴Bajo 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.
- Promueve la gestión del calendario de entregables o mapa de ruta del producto. Es dueño de responder ante el calendario planificado y estimado por el equipo.
- Busca la satisfacción de los usuarios finales.
- Ayuda a motivar al equipo para lograr el mejor producto posible.
- Hace análisis de sistema usando técnicas de modelado para lograr el mejor entendimiento del dominio del problema de negocio e impulsar el diseño del mejor producto posible ⁵.
- Es hábil para comprender el aspecto operacional del producto.
- Debe gestionar los riesgos y problemas junto con los demás roles. En esta vía debe levantar y buscar mitigar los riesgos de negocio.
- No estima, no provee estimaciones ni exige entregas obligadas por fechas.
- 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 líder facilitador y entrenador 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: administrar y asegurar que se siga el sistema Scrum, concientizar sobre el enfoque seguido bajo el mismo, capacitar y entrenar al equipo de desarrollo, impulsar el cambio en función de mejorar, facilitar la

⁵Characteristics of a Great Scrum Team, Barry Overeem, InfoQ, 2016.

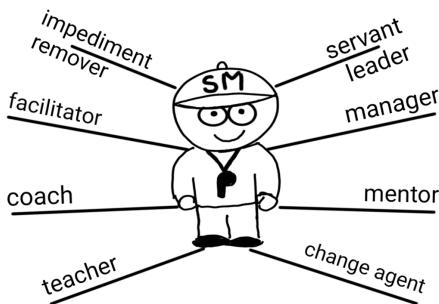


Figura 4.4: Aspectos de un SM según Scrum.org

resolución de impedimentos relacionados a la implementación de Scrum y liderar el proceso de desarrollo y mejora continua bajo Scrum. Es decir que el SM desarrolla ocho aspectos principales del rol: removedor de obstáculos, facilitador, entrenador, profesor, líder servicial, administrador, mentor y agente de cambio.

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

- Respeta, entiende y sigue los valores ágiles y de Scrum, como también así los fomenta.
- Es el responsable de que se siga Scrum y brinda apoyo a sus prácticas.
- Necesita desempeñar su rol con coraje ya que, dentro del equipo, es el principal referente como agente de cambio, movilizador de cambios culturales que pueden ser disruptivos y solucionador de conflictos en busca de facilitar resolver impedimentos.
- Debe asegurar la gestión de los riesgos y problemas con el equipo. En esta vía debe prestar particular atención a los riesgos relacionados al proceso.
- 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 y que se logren los objetivos.
- 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 negativas externas (si un Manager, Stakeholder o agente externo trata de ordenar al equipo que haga algo obligatorio, orden arbitraria de hacer cambios de miembros, manejar el backlog y compromisos, dividir el equipo, re-priorizar el sprint backlog a mitad de un sprint, etc.).
- Monitorea el progreso y el éxito con el resto de los roles del Scrum Team.
- Ayuda al equipo a encontrar una cadencia sostenible (con ritmo y naturalidad) para la entrega de incrementos potencialmente entregable de productos.
- Busca lograr un entorno seguro y de apoyo que genere confianza y respeto mutuo (seguridad psicológica). Ayuda a lidiar con los conflictos personales.
- Busca establecer acuerdos claros y que se cumpla lo pactado.
- Es un líder servicial cuyo enfoque se centra en las necesidades de los miembros del equipo, con el objetivo de lograr resultados alineados con los valores, principios y objetivos planteados.
- Es un entrenador y mentor de los miembros del equipo con un enfoque en la cultura, mentalidad y el comportamiento.

- Facilita la resolución de conflictos para abordar actitudes proactivas, productivas y colaborativas.
- Facilita el aprendizaje de Scrum, agilidad, técnicas y métodos complementarios relevantes para facilitar el proceso de trabajo.
- 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. O sea que debe facilitar la comprensión en los diálogos del equipo, detectar los ruidos de comunicación, hacer que todos se sepan expresar y se entiendan. Detectar cuando alguien no comprende y está quedando afuera de una idea o conversación para integrarlo. Buscar lograr inteligencia enjambre en equipo.
- La presencia en la Daily Scrum no es obligatoria, pero es aconsejable que esté para facilitarla y moderarla cuando sea necesario.
- Asegura el cumplimiento de los time-box facilitando la adopción de disciplina por parte del equipo.
- Procura lograr un equipo motivado.
- Un gran Maestro Scrum es un agente de cambio entendiendo que algunos cambios sólo se producirán si se es disruptivo. Está preparado para ser lo suficientemente disruptivo como para promover un cambio dentro de la organización⁶.
- Comprende sobre el aspecto técnico del desarrollo manejando diferentes técnicas y prácticas de desarrollo. En el caso de desarrollo de software sabe de ingeniería de software.

⁶Characteristics of a Great Scrum Team, Barry Overeem, InfoQ, 2016.

- 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.

Se puede ver un resumen del foco de las responsabilidades de los roles en la figura 4.5.

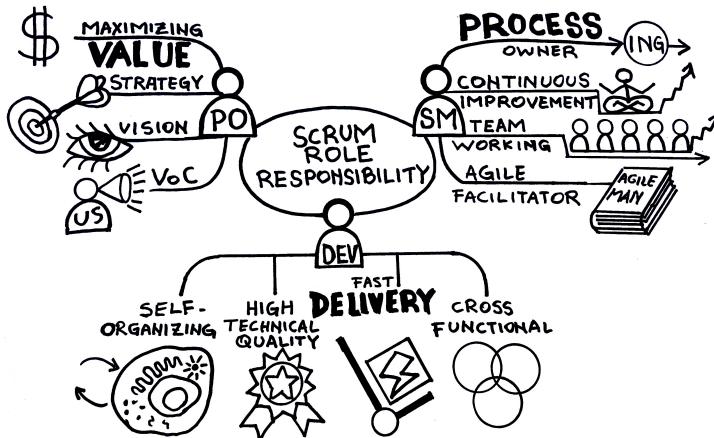


Figura 4.5: Mapa mental resumen didáctico del foco según las responsabilidades de los roles.

El PO se focaliza en maximizar el valor, la estrategia de negocio, la visión y la voz del cliente. El equipo de desarrollo (DEV) se responsabiliza principalmente de la entrega ágil, su auto-organización, la alta calidad técnica (excelencia técnica) y su multidisciplinariedad (trabajo colaborativo no especializado). El SM es dueño del proceso, por lo que se focaliza en el proceso de trabajo e ingeniería, la mejora continua, el trabajo en equipo y la facilitación ágil (referente ágil y de scrum).

4.3 Proceso Scrum

En el proceso Scrum o sistema de flujo de e Scrum (ver figura 4.6) 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.

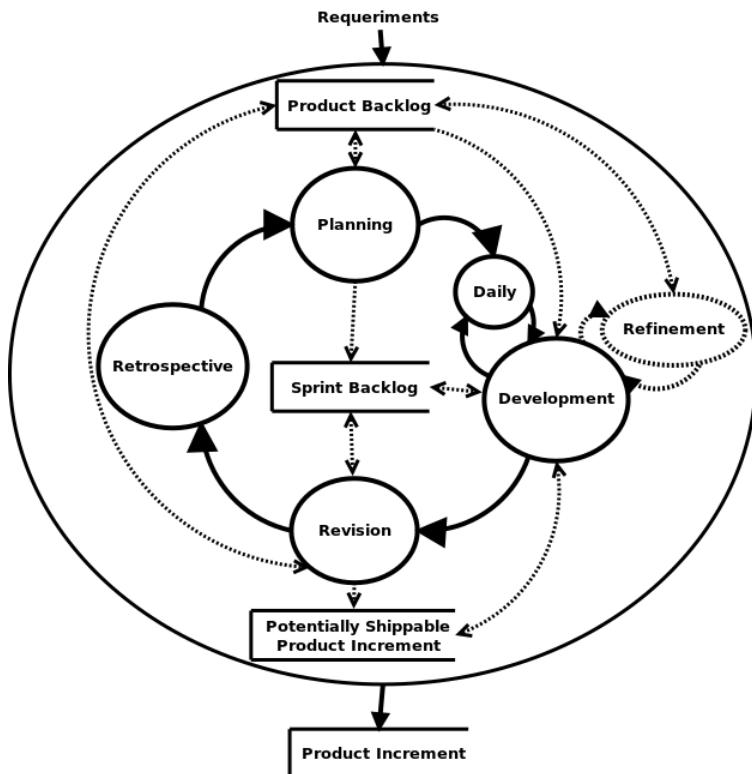


Figura 4.6: Diagrama de Flujo de Datos del Proceso Scrum

4.3.1 Reuniones principales

Como mencioné, los eventos son la planificación, la daily, la revisión y la retrospectiva. Si bien el refinamiento está planteado como una actividad en la ejecución del sprint, también se considera importante sin ser un evento oficial. A continuación se describen.

- **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 o de negocio relacionada al "qué", una segunda parte táctica o técnica relacionada al "cómo" y una tercera relacionada al acuerdo de cierre.

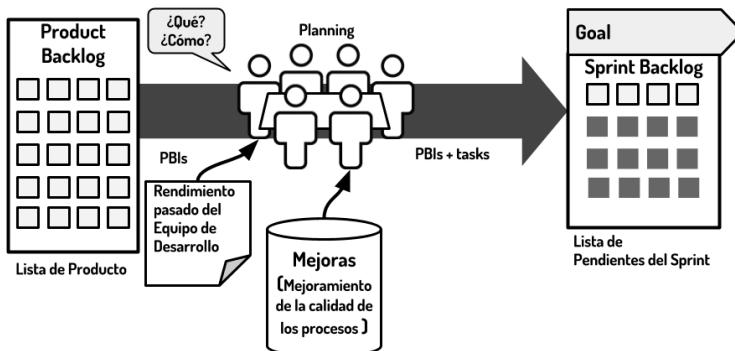


Figura 4.7: Evento de planeamiento

- **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 PO y están diseñados para asegurar que las hipótesis de requisitos del Cliente estén claramente representadas y puedan ser plenamente comprendidas por todos los Stakeholders y los desarrolladores del equipo. En la dinámica de la reunión, el PO cuenta cuáles son los PBIs disponibles que pasan cierto criterio de completitud, 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 PO 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 SM se encarga de facilitar el evento, moderar y tratar de asegurar de que todos los Stakeholder del proyecto que sean necesarios para 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 decidir y elegir finalmente el alcance del Sprint formando un acuerdo 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:** la Daily Scrum o Stand-up meeting es un evento 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:

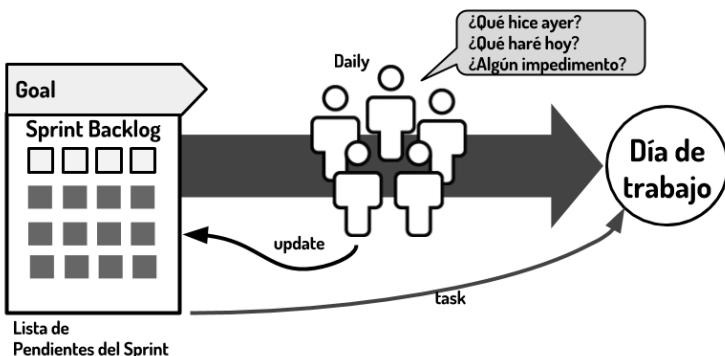


Figura 4.8: Daily

- ¿Qué hice ayer que ayudó a lograr el Objetivo del Sprint?
- ¿Qué haré hoy para ayudar a lograr el Objetivo del Sprint?
- Si tengo obstáculos: ¿Veo algún impedimento que evite que logremos el Objetivo del Sprint?

Es obligatorio que el Equipo asista a esta reunión y es aconsejable que esté el SM 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:** aunque es una reunión que no es un evento de Scrum original, es una actividad propuesta por la guía de Scrum útil para refinar la lista de PBIs o Product Backlog y que consiste en trabajar sobre las hipótesis de requerimientos, definirlas, analizarlas, 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 PO y el Equipo 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 un evento, actividad o 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 PO 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.

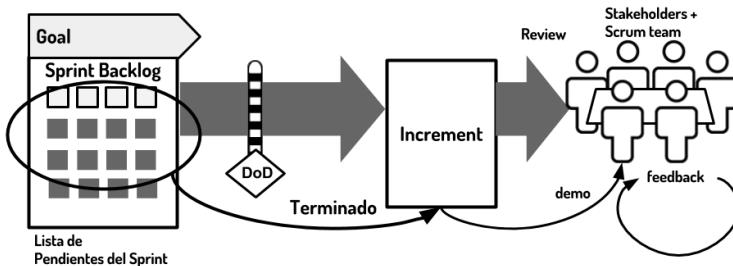


Figura 4.9: Revisión

- **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. También se puede editar el DoD si se considera necesario adaptarlo. En esta reunión es obligatoria la asistencia del Equipo Scrum (con el SM y el PO).

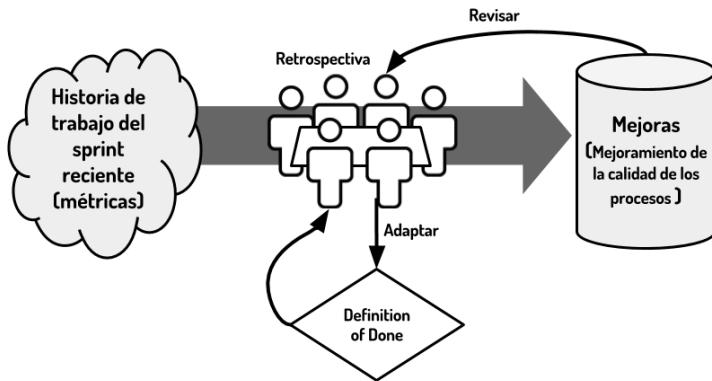


Figura 4.10: Retrospectiva

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 ítems 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 PO 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 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].

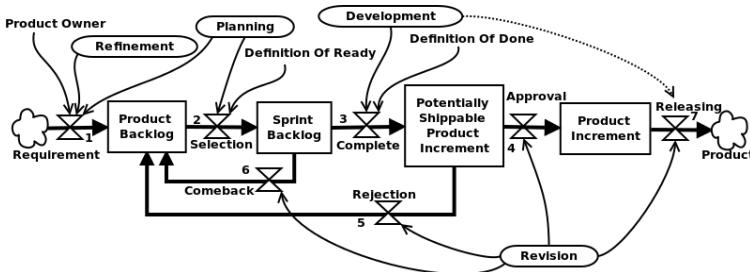


Figura 4.11: Diagrama de Flujo de Stock de artefactos Scrum

4.4.1 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 (o de un estado a otro). 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 (checklist) 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 completado⁷. La organización de desarrollo puede ser quien sugiera el DoD base estándar o es el propio equipo quien lo define y quien lo mejora.

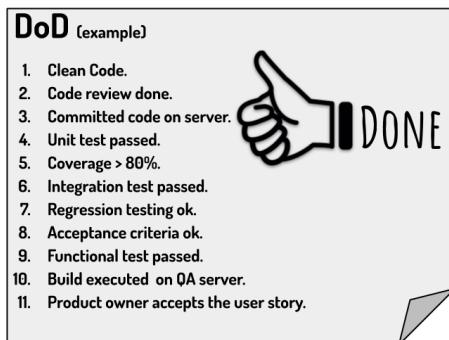


Figura 4.12: Ejemplo de una DoD

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 ± 2), o sea entre cinco y nueve (ver figura 4.14)⁸. Las Scrum Guide ([Ken/Jeff, 2017]) recomienda más de 3 y menos de 9 miembros en el equipo de desarrollo (sin PO y SM). Con menos de 4 miembros de equipo de desarrollo se tiene un equipo pobre que puede brindar un producto pobre. Por sobre 9 miembros 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, 2017]. La Reunión de Planificación

⁷[Ken/Jeff, 2013]

⁸El estudio de Lawrence Putnam sugiere 7 miembros de equipo óptimo con más o menos dos personas (2013). Algunos estudios dicen que el equipo debe rondar las 8 a 10 personas (Emery and Emery 1975, Bayer and Highsrrith 1994).

de Sprint debería tener un máximo de duración de ocho horas para un Sprint de un mes. El Scrum Diario 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.

	Duración de ceremonias según sprint			
Sprint	1 semana	2 semanas	3 semanas	1 mes
Planning	2 horas	4 horas	6 horas	8 horas
Daily	15 min	15 min	15 min	15 min
Review	1 horas	2 horas	3 horas	4 horas
Retrospective	45 min.	1 h 30 min.	2 h 15 min.	3 horas

Figura 4.13: Tiempos máximos Time-box segun Scrum Guide 2017.

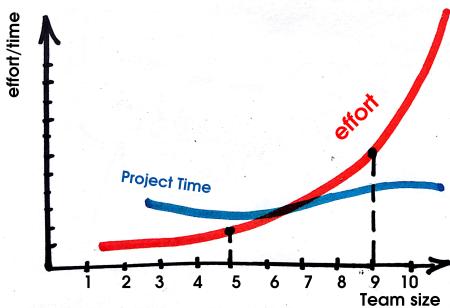


Figura 4.14: Tamaño de equipo óptimo según Putnam.

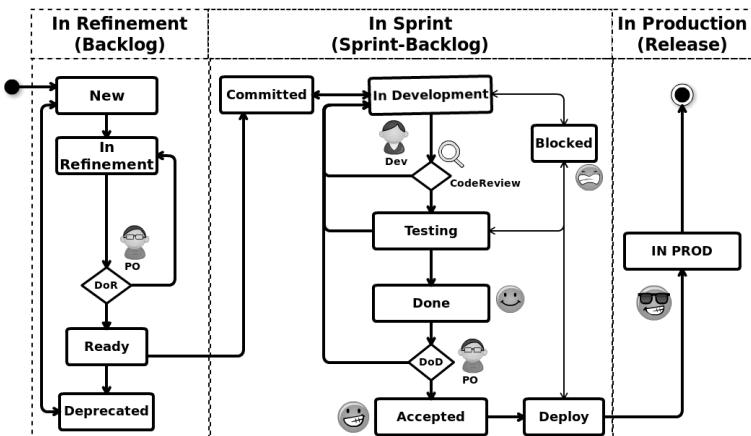


Figura 4.15: Diagrama de flujo de estados de un PBI o historia en el flujo de trabajo de desarrollo de software (ejemplo)

Capítulo 5

Gestión de proyectos

Scrum está orientado a productos, pues: "es un marco de trabajo para desarrollar, entregar y mantener productos complejos"¹. Sin embargo, en las organizaciones de gestión tradicional, los productos se desarrollan mediante la administración de proyectos. Entonces, bajo ese enfoque orientado a proyectos, Scrum puede verse como un marco ágil de gestión de proyectos para el desarrollo iterativo e incremental de productos, valiéndose de equipos autoorganizados.

Es en esta vía que en esta sección vine a ofrecer una interpretación y perspectiva comparativa y conciliadora con un enfoque de proyectos, no dejando de recordar que Scrum (según la Scrum Guide 2017) no trata sobre proyectos. Es el PMI con la Agile Alliance quienes formalizan la idea de Agilidad y el uso de Scrum en la gestión de proyectos de alta incertidumbre, mediante la Agile Practice Guide².

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

¹Scrum Guide 2017 (Scrum.org).

²Agile Practice Guide, PMI and Agile Alliance, 2017.

único³. 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 fig. 2.4). Con Scrum se puede implementar este proceso en una forma ágil, haciendo el inicio de forma lean, iterando el desarrollo en sprints de ejecución hasta llegar a un cierre donde el producto se mantiene, se traspasa o discontinua (ver fig. 5.1).

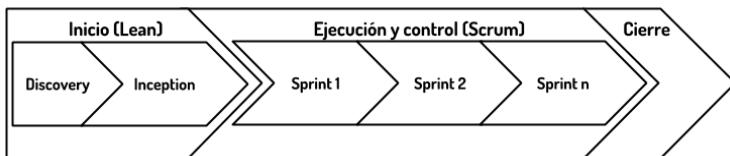


Figura 5.1: Ciclo de vida de un proyecto Scrum.

5.0.2 Inicio

En una organización ágil, en la fase de inicio del proyecto se inicia un producto o servicio mediante dos actividades principales: el descubrimiento (discovery) y la incepción (inception). Tengo entendido que estas etapas no están formalizadas en el marco ágil, aunque en la práctica, y en la mía en particular, sí se suelen emplear y son aceptadas en diferentes corrientes del movimiento ágil. En este contexto, es deseable que el inicio del proyecto aliente al desarrollo con Scrum con un primer backlog y cierta claridad para comenzar con baja incertidumbre el primer sprint.

Descubrimiento

En el desarrollo lean nos debemos enfocar en traer al cliente al proceso de desarrollo del producto para que se construya algo que, además de querer pagarla, lo quiera y tenga experiencias de usuario memorables. Necesitamos descubrir lo que la gente

³Se parafrasea la definición del PMBOOK [PMBOK, 2004].

quiere o necesita o un problema a resolver. El descubrimiento (discovery) se enfoca en iniciar un desarrollo lean y es parte, en la organización, del descubrimiento de iniciativas de proyectos o productos. Pues, es aquí donde nace el proyecto. En una organización lean, donde se implementa el método científico en el negocio mediante lean startup u otro método, es donde nacen las primeras “macro hipótesis” como parte de la iniciativa del proyecto. Es en este momento en donde se asocia, aunque sea temporalmente, alguna “métrica de éxito” (KPI) que nos permita validar la macro hipótesis. En esta etapa se comienza a entender y definir al cliente y se hace una investigación de cliente (customer research) para ello. Para esto se comienza a analizar el viaje del cliente en el producto o servicio (Customer Journey). También se revisan temas de factibilidad técnica. El resultado de esta fase será la entrada para la etapa de inception. En la organización es la etapa donde se genera la definición de la oportunidad de negocio en una evaluación de oportunidad (opportunity assessment) o caso de negocio (business case) que sirve de entrada para evaluación de iniciativas de proyectos (en la PMO si existiera) y presupuestación (en caso que se trabaje bajo presupuestación).

Incepción

En la incepción (inception) es donde se responde el por qué, qué y cómo, a alto nivel, para posibilitar el desarrollo posterior. Al inicio se hace el lanzamiento del producto o servicio (Kickoff), aclarando el porqué se hace el producto, el contexto, la visión y buscando lograr el compromiso de los interesados. Se dilucida a alto nivel las necesidades del cliente, las características del producto o servicio a alto nivel, una arquitectura o metáfora del producto a alto nivel, supuestos y estimaciones relativas iniciales. En aproximadamente una semana de trabajo, el equipo va a entender los objetivos del producto, al usuario o cliente objetivo, el alcance y una guía inicial o roadmap para el desarrollo evolutivo. De aquí saldrá el backlog inicial para comenzar a trabajar con Scrum, en la etapa de ejecución del proyecto, con el sprint 1. En resumen, el objetivo principal de la inception es lograr que el equipo converse, defina

y entienda, colaborativamente, lo que se va a desarrollar⁴.

5.0.3 Planificación y ciclo de vida

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 predictiva (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 iterativo e incremental de productos (por incrementos de producto) en iteraciones cortas, llamadas Sprint, donde cada iteración tiene su respectiva planificación (ver figura 5.2) y su incremento de producto, en caso de haberlo logrado (idealmente producto integrado y funcionando de cara al usuario o cliente).

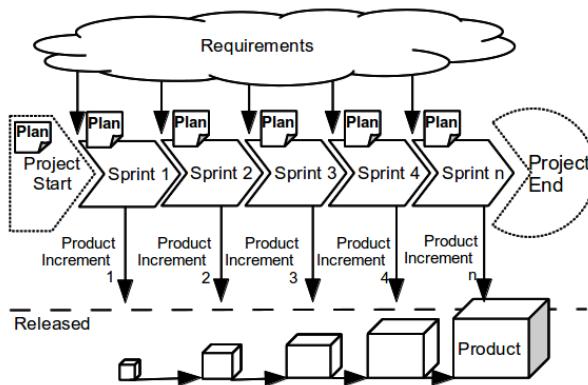


Figura 5.2: Proyecto Scrum

En un sentido, es como si un proyecto se dividiera en muchos pequeñitos, donde “cada Sprint puede considerarse un proyecto con un horizonte no mayor de un mes”⁵. Entonces, podemos

⁴[Paulo Caroli, 2017]

⁵Scrum Guide 2017 (Scrum.org).

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

Niveles de planificación

En resumen se puede decir que se suelen usar tres niveles de planificación (ver fig. 5.3), de los cuales dos son prescriptos por Scrum: Daily y Sprint Planning. La Daily es la planificación a corto plazo, diaria. La Sprint Planning es la planificación a mediano plazo donde se planifica la iteración. En la práctica, se puede utilizar una planificación a largo plazo (Long-term Planning) de la cual se puede obtener un Roadmap o un Release Plan. Esta última puede iniciarse al inicio del proyecto y refinarse en los refinamientos o como una actividad parte de la ejecución.

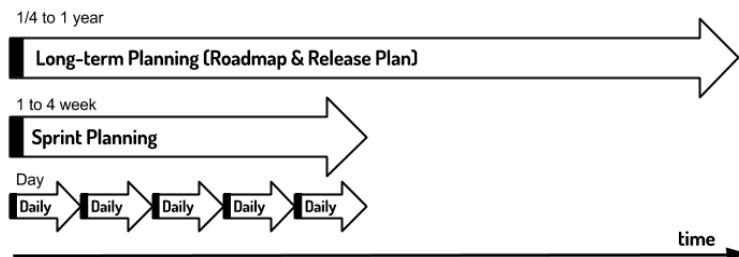


Figura 5.3: Niveles de planificación

5.0.4 Plan de ruta

El plan de ruta o roadmap (hoja de ruta) de un producto es un plan de alto nivel que describe como el producto va a ir evolucionando en el futuro, de algo simple a algo complejo. Nos permite proyectar en el tiempo donde queremos que esté

el producto en el futuro. El objetivo del roadmap es ser una especie de borrador para comunicar, mostrar, reflexionar sobre el futuro del producto, alinear la estrategia y guiar la construcción del mismo. Es una herramienta de planificación dinámica e interactiva, no un contrato para trabajar orientados a deadline.

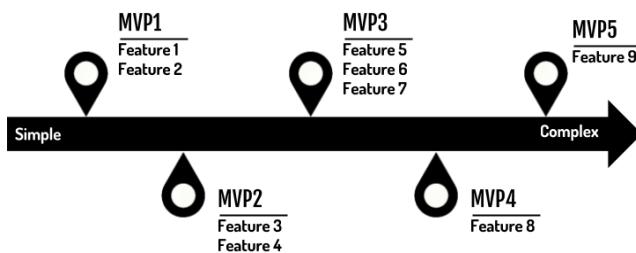


Figura 5.4: Roadmap ejemplo

5.0.5 Planificación de entregables

Con este marco de trabajo no es necesario hacer una entrega final (o "releasing") ya que se pueden hacer entregas paulatinas. En cada Sprint se puede entregar valor según lo planificado dentro del mismo sprint (planning). Además, para hacer diversas entregas intermedias planificadas se puede crear un plan de muy alto nivel para múltiples Sprints durante una planificación de lanzamientos llamado "Release Plan". Este plan de entregables o de lanzamientos es una guía con la que se pretende reflejar las expectativas sobre qué funcionalidad se implementará y cuándo, aproximadamente, se completará⁶. 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 hitos de releases no deberían ser compromisos rígidos y contractuales y, además, el desarrollo del proyecto no debería centrarse en respetar el plan. Por este motivo el plan de lanzamiento no es un plan estático o rígido; pues, se

⁶[Scrum-Institute, 2015]

cambia durante todo el proyecto cuando nuevos requerimientos o conocimientos están disponible y, por ejemplo, cuando entradas PBIs en el Scrum Product Backlog cambian y se re-estiman. Por lo tanto, este plan debe ser revisado y actualizado colaborativamente en intervalos regulares, por ejemplo, después de cada Sprint (en refinamientos o reuniones acordadas).

Release plan

Un plan de entregas o "Release Plan" ágil es, normalmente, un conjunto de historias de usuario (o épicas) agrupadas por "releases" (versiones del producto) que se ponen a disposición de los usuarios ⁷ o agrupadas por Sprints (ver fig. 5.5). En otras palabras, es una planificación a media distancia como una proyección hacia adelante en una serie de sprints ⁸. 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" ⁹. Se puede utilizar Scrum con éxito sin necesidad de utilizar "Release Planning".

⁷Release plan, jmbeas, 2011; Agile Estimating and Planning, Mike Cohn

⁸Release Planning, Retiring the Term but not the Technique, Mike Cohn, 2012.

⁹Gone 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

Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5
Objetivo 1	Objetivo 2	Objetivo 3	Objetivo 4	Objetivo 5
Release	Release	Release	Release	Release

Figura 5.5: Release plan en Sprints ejemplo

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

5.0.6 Triángulo de la gestión de proyectos

Bajo el marco de Scrum se cambia la idea relacionada al triángulo clásico de la gestión de proyectos (o triángulo de hierro). El compromiso ya no es entre el tiempo, presupuesto y calidad;

sino que se basa en el triángulo de: presupuesto (costo), tiempo (agenda) y funcionalidad (alcance) (ver figura 5.6)¹⁰. Además, tradicionalmente se ha intentado fijar el alcance para negociar y variar el presupuesto y el tiempo. En cambio, desde la perspectiva ágil, se intentan mantener fijos el tiempo y el presupuesto, mientras se varía el alcance¹¹. Cuando consideramos el presupuesto fijo (costo del equipo), pretendemos mantener al equipo fijo (equipo estable). Pues se desea equipos estables que maduren, se fortalezcan y, en consecuencia, se transformen en equipos de alto rendimiento. Por otro lado, una manera de trabajar con tiempos fijos es que el equipo se comprometa a trabajar en iteraciones fijas (sprints). Además el equipo podría fijar releases o bloques temáticos dado un tiempo. Y esto no es trabajar bajo “deadlines”, ya que el enfoque no es trabajar orientados a fechas, sino a valor de negocio. Entonces nos queda el alcance, descompuesto en ítems de backlog, y la pregunta ahora es: ¿cuánto puedes hacer en este tiempo y bajo un costo fijo para alcanzar los objetivos de negocio?

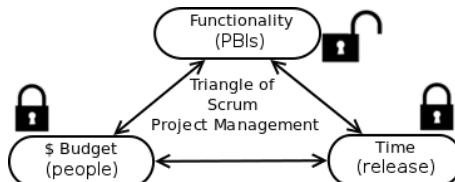


Figura 5.6: Triángulo de Gestión de Proyectos Scrum

5.0.7 Priorización trade-off

La ventaja de esta aproximación es que en vez de focalizarnos en plazos con fechas de entrega, nos centramos en desarrollar valor. ¿Cuál es el máximo valor que podemos desarrollar en un tiempo dado? Aquí entra el concepto de priorización trade-off, que se lleva a cabo en la ejecución del proyecto (o construcción de un

¹⁰The iron triangle of planning, Atlassian, Tareq Aljaber, 2017.

¹¹[Martin Alaimo, 2014]

producto). Trade-off es un acuerdo en el cual se deja algo fuera para priorizar otra cosa que se desea más. Pues, ante cambios, se prioriza un ítem reemplazando por otro de esfuerzo semejante para cumplir con entregar valor. Si en una iteración llega un cambio rompe fila, entonces vemos qué podemos sacar del sprint con mismo peso de esfuerzo. Lo mismo si queremos cumplir con un objetivo deseado en un tiempo planificado y necesitamos introducir un nuevo trabajo, pues revisaremos qué podemos sacar de esfuerzo semejante y cuyo valor podemos despriorizar en relación al cambio deseado.

Como vemos, aquí el enfoque viró rotundamente en relación a la administración clásica. En vez de asfixiar a los equipos para que lleguen a cumplir con una fecha, por lo general impuesta arbitrariamente, se les permite tener cierta autonomía (con un responsable de negocio) para que prioricen qué valor de negocio pueden entregar, para un objetivo planeado, con colaboración y participación del equipo mismo.

5.0.8 Gestión orientada a producto

En la gestión orientada a producto nos centramos en iniciativas que evolucionan productos. Entendiendo producto como productos software, servicios, recursos compartidos, suscripciones, agencia, aportación de público, etcétera; basado en software y que ofrece algún tipo de experiencia digital que satisface alguna necesidad o deseo o resuelve un problema, por el cual alguien está dispuesto a pagar. Este tipo de productos pasan por etapas de descubrimiento, desarrollo, marketing y operación. Solo que esta secuencia no es estrictamente en una sola secuencia en cascada, sino más bien en múltiples ciclos de aprendizaje y desarrollo iterativo, incremental y evolutivo, a partir de hipótesis, prototipos y MVPs, como a continuación se explica.

Producto mínimo viable

El verdadero objetivo de Scrum es conseguir un producto mínimo viable o MVP en manos de los futuros clientes cuán rápido sea

possible y obtener sus comentarios como feedback temprano¹². MVP es una estrategia para el aprendizaje de forma iterativa sobre sus clientes para poner a prueba las hipótesis fundamentales del negocio¹³. Validamos una propuesta de valor con un conjunto de características, probando la experiencia del usuario con un conjunto de clientes representativos de un segmento de mercado. Es un incremento de producto, subconjunto del 20% de features que representan parte del 80% de valor¹⁴. Las características o features del MVP representan conceptualmente al producto final completo (aunque no sea el producto final) y se prueba con un grupo de usuarios tempranos o "early adopters". Las pruebas deben ser medibles y se hacen sobre las hipótesis fundamentales del producto como negocio. Cabe aclarar que, originariamente, el MVP es el primer producto mínimo viable en una cadena evolutiva de entregables.

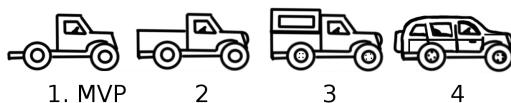


Figura 5.7: El MVP

Evolución de un MVP

Hay una evolución desde prototipos a MVP a MMP, hasta que un producto calza con el mercado. Habitualmente, en la práctica y por simplicidad explicativa, hay quienes denominan MVP a cualquier incremento conceptual, funcional y evolutivo del producto como un todo (ver fig. 5.7). La verdad que, a los fines explicativos de este libro, no importa qué terminología use (Versión, Release, MVP, etc.), lo importante es que todos los implicados tengan la misma comprensión de los conceptos implicados en la evolución del producto.

¹²[Jeff Sutherland, 2014b]

¹³[Greg Gehrlich, 2012]

¹⁴[Jeff Sutherland, 2014b]

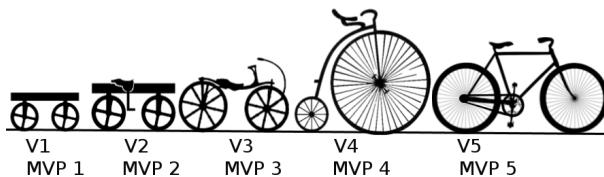


Figura 5.8: Evolución de una bicicleta

Oferta mínima viable económicamente

Por otro lado, hay otro concepto que se maneja en el ámbito de la agilidad, el mínimo producto vendible o 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. Hay quienes lo denominan mínimo producto viable económico o EMVP, ya que es la oferta mínima viable económico en un mercado. A partir de un MMP es que se llega, posteriormente, a lograr el ajuste con el mercado o Market-Fit, que es la puesta a punto del motor de la propuesta de valor mediante el desarrollo, la venta y la comercialización, impulsando la demanda a los clientes mayoritarios.

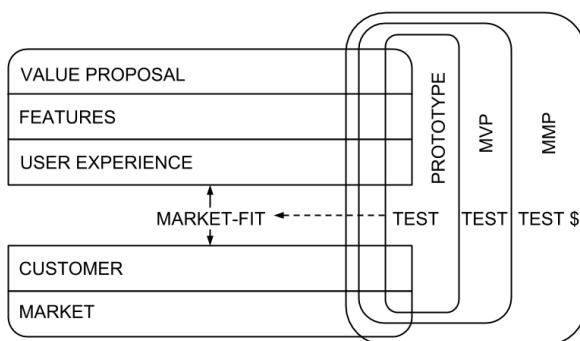


Figura 5.9: Ajuste de mercado, MVP y MMP

Ciclo de vida del producto

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. Despues 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. 5.10)¹⁵.

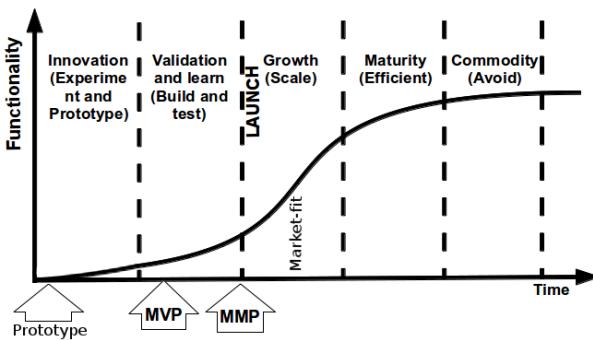


Figura 5.10: Crecimiento del producto

Llevar trabajo a los equipos

Al cambiar la perspectiva hacia estar orientados a productos y mantener equipos estables, cambia el enfoque de la gestión de proyectos. Ya no buscamos recursos humanos para proyectos, desarmando y armando equipos en función de proyectos. Lo que se busca hacer, entre otras cosas, es llevar trabajo a los equipos, buscando productos para equipos. Buscamos alinear a equipos en

¹⁵[Greg Gehrlich, 2012]

función de objetivos estratégicos. Mantener productos o servicios de valor para objetivos estratégicos. Donde los equipos son activos de la empresa y que buscan siempre maximizar el valor entregado. Esto puede llevar a cambiar toda la dinámica de funcionamiento de una PMO e, inclusive, desarrollar independientemente de presupuestos (tema que queda fuera del alcance de este libro).

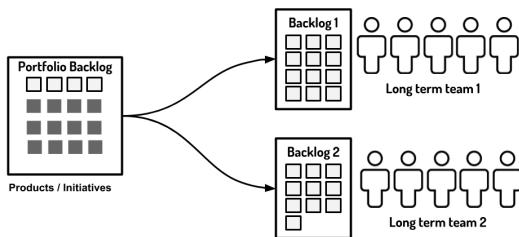


Figura 5.11: Llevar trabajo a los equipos de larga duración

5.0.9 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, dificultosa 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.

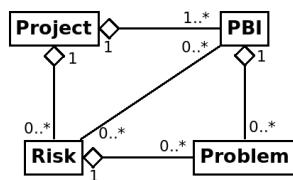


Figura 5.12: Diagrama de riesgos, problemas e ítems del proyecto (PBIs).

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. Si considera necesario no formalizar en registros esta gestión, no lo haga, puede ser un trabajo desperdiciado. Se debe buscar la simplicidad y el foco en el flujo de trabajo sin impedimentos.

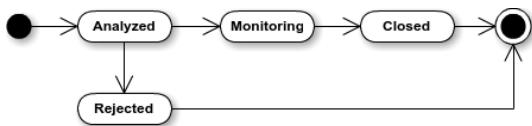


Figura 5.13: Estados posibles de un riesgo.

Capítulo 6

Métricas

¿Para qué medir? Si no medimos no podemos adaptarnos bajo un marco empírico como propone Scrum. Medimos en forma transparente, en procesos de inspección y adaptación. Las métricas ayudan al equipo a evaluar su propio desempeño en el proceso de trabajo, para poder hacer cambios basados en hechos. También es un soporte a la gestión de proyecto/producto para poder medir el progreso en cuanto a impacto de negocio (no a seguir un plan), tomar decisiones de negocio, revisar la actividad de generación de valor y el desempeño¹ del equipo en cuanto a grado de madurez o evolución. Además hay indicadores de calidad de producto que nos pueden señalar aspectos claves de prestancia y salud de un producto. Se puede ver a las métricas como parte del monitoreo de salud del equipo, sistema de trabajo, producto y negocio; y parte del sistema de estabilización de la empresa y del sistema de mejora continua y aprendizaje.

En síntesis, se pueden categorizar a las métricas en cuatro aspectos: 1) métricas de equipo (team metrics); 2) métricas de proceso de trabajo (process metrics); 3) métricas de calidad de producto (product metrics); 4) métricas de resultados de negocio (outcome metrics) [ver fig. 6.1].

¹Las 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.

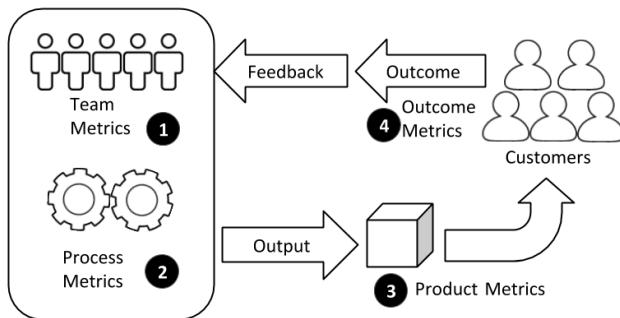


Figura 6.1: Modelo de 4 categoría de métricas o KPIs.

6.0.1 Métricas de equipo

El trabajo colaborativo es uno de los cuatro aspectos claves de la agilidad. Por tal motivo podemos medirlo de alguna manera simple para ayudar en la mejora continua del equipo. A continuación puedo mostrar algunas sugerencias.

1. **Team Happiness:** Un estudio de Harvard muestra que la felicidad aumenta la producción de cualquier tipo de trabajo, pues "la gente feliz es 12 % más productiva"². Además, un principio de la agilidad es desarrollar en torno a individuos motivados. En base a esto podemos intentar medir la felicidad Sprint a Sprint³.

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 6.2).

²[Group of U.K. Researchers, 2014]

³[Jeff Sutherland, 2014a]



Figura 6.2: 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 feliz⁴.

- (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.

- 2. **Team Morale:** Indicador moral de equipo dice cómo está la moral en tu equipo. La moral del equipo es una medida más estable y confiable que la felicidad del equipo⁵. Esta métrica mide el orgullo, entusiasmo, energía, resiliencia, disposición para ayudarse mutuamente y motivación por el significado y propósito. Se puede medir mediante encuestas periódicas. Un cuestionario simple y práctico puede ser el siguiente (Las preguntas individuales se califican en una escala de 1 a 7 o de 1 a 5):

- (a) ¿Me siento bien (siento que encajo) y me siento fortalecido en mi equipo?
- (b) ¿Estoy orgulloso del trabajo que hago para mi equipo?
- (c) ¿Estoy entusiasmado con el trabajo que hago para mi equipo?

⁴[Group of U.K. Researchers, 2014]

⁵Agile Teams: Don't use happiness metrics, measure Team Morale. Agilistic, Christiaan Verwijs, 2014.

- (d) ¿Encuentro el trabajo que hago para mi equipo de significado y con propósito?
3. **Team Stability:** Se considera que se logra buena colaboración con equipos estables. Por ese motivo, bajo el marco ágil, se prioriza a los equipos estables por sobre los proyectos⁶. Es decir que se recomienda mantener a los equipos estables y evitar mezclar personas entre equipos o desarmar equipos para asignar a otros proyectos (la rotación). Los equipos estables tienden a conocer mejor su capacidad, lo que permite cierta capacidad de predicción y mayor sinergia. Los miembros del equipo deben dedicarse a un solo equipo siempre que sea posible. Pero esto no siempre se logra por la rotación laboral o necesidades de la empresa. Por eso es útil medir el aspecto de “Team Stability” o de rotación. Pues hay que mantener una rotación baja saludable. Una forma de indicador puede ser: número de integrantes que salen del equipo al año / la media del número de integrantes de ese año o período.
4. **Team Skill Balance:** Si existe un desbalance de conocimientos y de esfuerzo en el equipo puede provocar la fatiga de algunos integrantes, problemas por eslabones débiles (en contingencias, entrevistas, solución de problemas, etc.), dependencias sobre alguna persona o desavenencias por percepción de inequidad de esfuerzos. Un indicador de balance de competencias del equipo (Team Skill Balance Average) podría ayudar a que el equipo sea capaz de trabajar como una unidad integrada. Cada integrante debe ser un poco más multifuncional, colaborando en tareas o actividades que no son su fuerte o especialidad.

6.0.2 Métricas del proceso de trabajo

La entrega de valor fluida y continua es otro de los cuatro aspectos claves de la agilidad. Para mejorar este flujo de valor se busca, entre otras cosas, un flujo estable, sostenible y en un

⁶The Disciplined Agile (DA) Framework

proceso de trabajo limpio (sin impedimentos ni desperdicios). En este sentido hay muchos indicadores clave del proceso de trabajo, de rendimiento, internos del equipo, que nos pueden ser de utilidad para monitorear el proceso de desarrollo y que nos permita la inspección y adaptación constante, como por ejemplo los siguientes⁷:

1. Velocity: La 'velocidad'⁸ 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 Sprint⁹. Aquí hay que tener en cuenta que 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. Representa la complejidad o esfuerzo necesario para terminar las tareas de una historia¹⁰. El SP sirve como estimación de la 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.

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 sprint¹¹, incluyendo las de valor indirecto. En este sentido, las historias de usuarios completadas (tomadas del backlog técnico) que tienen valor indirecto para el cliente, las

⁷Scott 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).

⁸"Sum of original estimates of all accepted work" [Scott/Jeff, 2013].

⁹[Jipson Thomas, 2015]

¹⁰[Jipson Thomas, 2015]

¹¹"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]

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 velocidad¹². 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 Shore¹³) 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 de unidades de trabajo o SP promedio estimados y aceptados

¹²[David Koontz, 2014]

¹³[James Shore, 2015]

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, estén terminados o no¹⁴. 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 reportado¹⁵. 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) **Capacity done:** la capacidad hecha aceptada por el PO, es el trabajo total potencialmente entregable aceptado que incluye Historias de Usuario e Historias Técnicas. Se suele usar cuando se hace la distinción entre Historias de Usuario y técnicas.
 - (b) **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.¹⁶
 - (c) **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 ($8 \times 6 \text{ hs} \times 10$).

¹⁴Scott y Jeff [Scott/Jeff, 2013]

¹⁵Scott y Jeff [Scott/Jeff, 2013]

¹⁶[Satish Thatte, 2013]

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 es partes del marco de trabajo aceptada por la comunidad, pero Working capacity no lo es del todo aceptado, aunque es ampliamente usado.

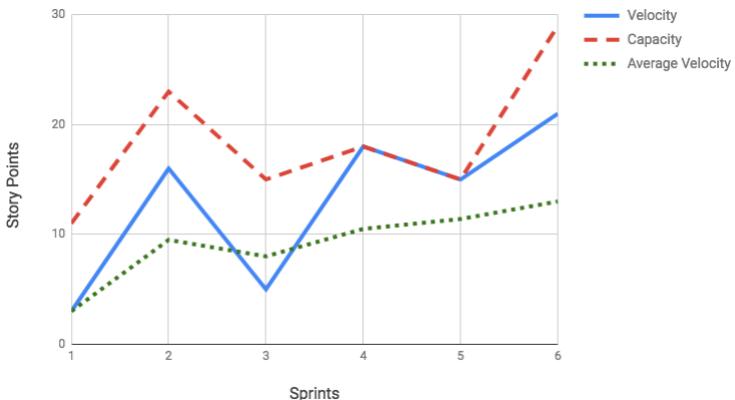


Figura 6.3: Gráfico de velocidad, capacidad y velocidad promedio

Sprint	Velocity Forecast	Capacity Forecast	Velocity	Capacity	Average Velocity	Member count
1	3	21	3	11	3	5
2	16	30	16	23	9,5	6
3	5	21	5	15	8	6
4	18	18	18	18	10,5	6
5	15	15	15	15	11,4	6
6	21	29	21	29	13	6

Figura 6.4: Datos del gráfico de velocidad, capacidad y velocidad promedio

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 velocidad y la capacidad (o capacidad de trabajo): $(\text{Velocity} / \text{Capacity}) \times 100\%$. La misma debe permanecer, según recomendación, en la vecindad de 80 % en promedio para un equipo saludable. En el caso

de contemplar la capacidad de trabajo (work capacity), 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. Si lo que se usa es la capacidad aceptada (Capacity done), puede ser un indicativo de que cuanto se ha trabajado en historias de usuario en relación a historias técnicas.

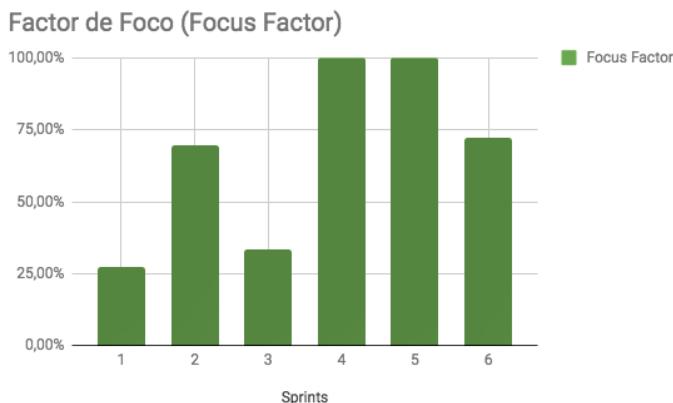


Figura 6.5: Gráfico de factor de foco

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): (Current Sprint's Velocity / Original Velocity) x 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. **Effectiveness:** La efectividad de un sprint medida como historias o features entregadas (aceptadas) versus las elegidas en la planning, las que se tomaron en el sprint (forecasting). También se podría medir la efectividad relacionada a si se cumplió con el objetivo del sprint.

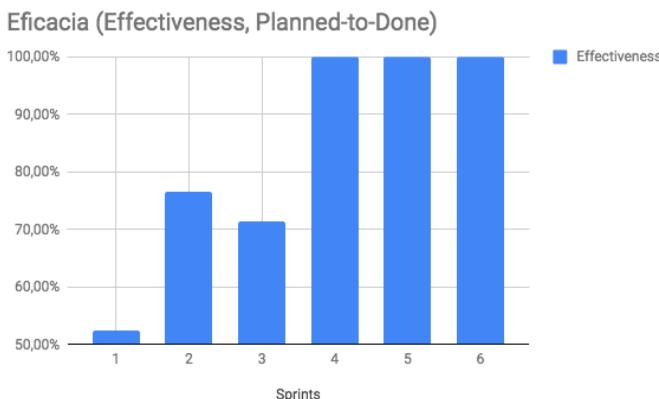


Figura 6.6: Gráfico de efectividad

Sprint	Velocity Forecast	Capacity Forecast	Velocity	Capacity	Focus Factor	Effectiveness	Average Velocity	Member count
1	3	21	3	11	27,27%	52,38%	3	5
2	16	30	16	23	69,57%	76,67%	9,5	6
3	5	21	5	15	33,33%	71,43%	8	6
4	18	18	18	18	100,00%	100,00%	10,5	6
5	15	15	15	15	100,00%	100,00%	11,4	6
6	21	29	21	29	72,41%	100,00%	13	6

Figura 6.7: Datos consolidados

7. **Delivery Frequency:** ¿Con qué frecuencia se despliega el código a producción? La “frecuencia de entrega” (Delivery Frequency¹⁷) o la “tasa de entrega” (Delivery rate) es un indicador que nos puede servir para ver la evolución de los tiempos de entrega, que impacta en el tiempo de

¹⁷[Jez Humble, 2018]

salida al mercado. Es un buen indicador cuando importa más el outcome que el output del equipo. Nos puede ayudar a mejorar el continuous delivery. Se puede calcular como tiempo promedio entre subidas a producción o como cantidad de subidas promedio en un período de tiempo (por ejemplo por sprint). La fórmula más simple puede ser, uno sobre el promedio de los períodos de despliegues.

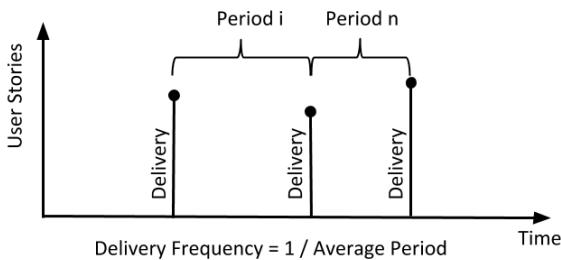


Figura 6.8: Esquema del Delivery Frequency

8. **Stories Inventory:** El “inventario de historias” mide el inventario de código o producto potencialmente entregable. Es decir, mide las historias que se terminaron pero no se subieron a producción para que estén funcionando de cara al cliente. En Scrum se busca un flujo limpio (lean) y el código no entregado a producción es un inventario, que es una forma de grasa del sistema de producción y que queremos reducir. Esta métrica, al igual que el Delivery rate, nos puede ayudar a visibilizar esta grasa para mejorar el continuous delivery.

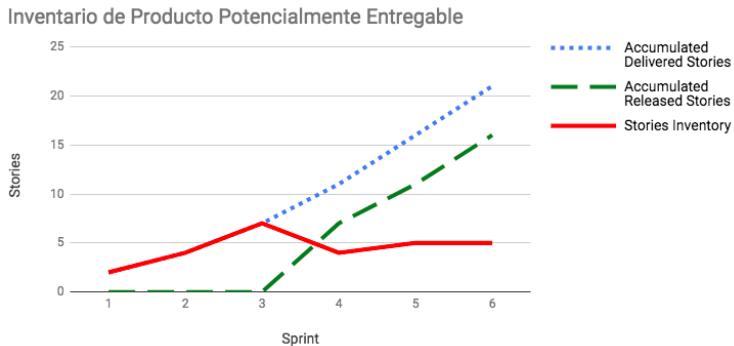


Figura 6.9: Gráfico de Stories Inventory

Sprint	Delivered Stories	Released Stories	Accumulated Delivered Stories	Accumulated Released Stories	Stories Inventory
1	2	0	2	0	2
2	2	0	4	0	4
3	3	0	7	0	7
4	4	7	11	7	4
5	5	4	16	11	5
6	5	5	21	16	5

Figura 6.10: Datos consolidados del gráfico Stories Inventory

9. **Mean Time To Restore (MTTR):** Cuánto tiempo lleva generalmente restaurar el servicio para la aplicación principal o el servicio en el que trabajan cuando ocurre un incidente de servicio¹⁸.
10. **Métricas de flujo:** Scrum no prescribe métricas, por lo que bien podrían usarse métricas de flujo en vez de las basadas en SP, tomando cosas del método kanban. Algunas de ellas son el Throughput, el Cycle time (Ct) y el Lead time (Lt). Siendo el Throughput la velocidad, como número medio de unidades procesadas en un tiempo determinado. El Cycle time, tiempo real de trabajo sobre una unidad sin los tiempos de cola o espera. Y el Lead time, tiempo total en que una unidad pasa por el sistema de trabajo.

¹⁸[Jez Humble, 2018]

- (a) **Development Lead Time:** El lead time del desarrollo de software se puede calcular como el tiempo desde que una historia es creada hasta que es desplegada en producción¹⁹. Este lead time también es un buen indicador cuando se prioriza el outcome sobre el output del equipo, indicando parcialmente cuán rápidos somos para salir al mercado para obtener resultados.
- (b) **Delivery Lead Time:** Desde el punto de vista de DevOps se puede medir el Lead Time desde que se hace el primer commit hasta que el código se despliega en producción²⁰.

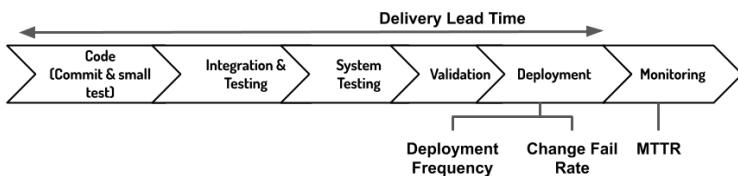


Figura 6.11: DevOps pipeline y sus métricas

Es recomendable tener en cuenta que estas métricas (las que seleccione el equipo) deben ser de uso interno del equipo, principalmente las de velocidad. En el caso que se requiera medir output (para evaluar la performance) de varios equipos, una tribu o una organización, se pueden usar métricas como las sugeridas por Jez Humble en su libro Accelerate²¹: Delivery Lead Time, Delivery Frequency, Mean Time To Restore (MTTR) y Change Failure Rate. De este modo se puede evaluar la prestancia del sistema de desarrollo de software y no medir equipos.

¹⁹Agile Alliance Resources 2018

²⁰[Jez Humble, 2018]

²¹[Jez Humble, 2018]

6.0.3 Métricas de calidad de producto

Bajo el marco ágil buscamos la excelencia técnica y en esa vía buscamos productos de calidad excelente. Pues bien, como se imaginan, tendremos que medir de algún modo la calidad de nuestro producto, ya sea calidad interna o del producto operando. Podemos citar algunos indicadores.

1. **Maintainability:** Hay métricas asociadas a la mantenibilidad, como lo son: test Coverage, CLOC, issues, complexity, technical debt, etcétera.
 - (a) **Test coverage:** Por ejemplo la cobertura de prueba es una herramienta útil para encontrar partes no probadas de una base de código, lo cual nos ayuda a saber si se están realizando pruebas suficientes para apoyar el aseguramiento de calidad.
 - (b) **Accumulated technical debt:** La deuda técnica creada es una forma de inventario no deseado y que debemos manejar adecuadamente, para tener una buena calidad. Medir la deuda técnica acumulada versus la resuelta acumulada nos da una visibilidad de este inventario.

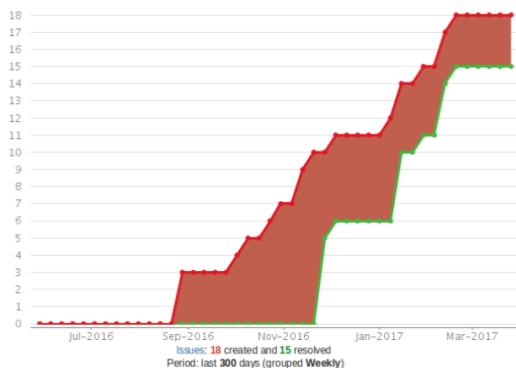


Figura 6.12: Gráfico de deuda técnica acumulada versus resuelta

2. **Reliability:** La fiabilidad es una medida de qué tan robusta es una aplicación. Puede ser medida con otros KPI indirectos como defectos escapados (Escaped defects), tasa de errores (Error rate) o tiempo medio entre fallos (mean time between failures).
 - (a) **Escaped defects:** Un defecto escapado es un defecto que no fue encontrado por el equipo en el proceso de desarrollo y que escapó al control de calidad interno del equipo. Por lo general, esos problemas son incidentes en operaciones (OPCON), es decir que los encuentran los usuarios finales una vez que la versión de release fue publicada y puesta a su disposición. Su cálculo más simple es cantidad en un período dado (por ejemplo cantidad de OPCON por sprint). Como mencionamos antes, es una medida de fiabilidad, cuanto menos defectos más fiable.
 - (b) **Change Failure Rate:** El porcentaje de cambios a producción (releases o cambios) que dan como resultado un servicio degradado o fallo y posteriormente requieren remediación (hotfix, rollback, fix-forward, patch, etc.). Esta métrica también puede usarse como prestancia o calidad del proceso de desarrollo ²².
3. **Performance:** Las métricas de prestancia o rendimiento de aplicación nos ayudan a detectar problemas o posibles mejoras en la experiencia del usuario. En el caso de aplicaciones web hay una infinidad de indicadores que se pueden usar como: "Availability", "Uptime", "Loading time" o "Time to Load" (TTL), "Time to Interact" (TTI), "Average Application Response Time", "Peak Response Time", etcétera. El equipo debe elegir los necesarios para el contexto de operatividad y objetivos de negocio.
4. **Security:** Métricas relacionadas a seguridad como: "Dynamic analysis issues" o "Static analysis issues".

²²[Jez Humble, 2018]

Estas métricas listadas son solo una guía para investigar otras y analizar cuáles pueden ser necesarias para algún caso particular.

6.0.4 Métricas de resultados de negocio

Siguiendo una de las claves de la agilidad, el flujo de entrega de valor continuo, es que podemos suponer que la verdadera medida de éxito en Scrum es el incremento de producto que es valioso, que genera resultados valiosos. Pero, ¿qué es valioso y para quién? Cabe aclarar que, el valor de negocio no se refiere solo a valor monetario, también puede ser de valor intangible, como el posicionamiento de una marca o el valor dado por mejorar la experiencia del usuario, siendo un beneficio para el usuario o cliente y, en consecuencia, también para la compañía. El valor es lo valioso para la compañía y para el cliente. Si estamos centrados en el cliente debemos pensar en qué es valioso para él. Pues bien, ¿cómo medir qué es valioso? Scrum no prescribe métricas de resultados de negocio (business outcome) o indicadores clave de impacto de negocio, aunque es necesario usarlos para saber si realmente estamos entregando valor o generando el impacto deseado.

El PO tiene el poder de tomar la decisión última de negocio y necesita datos empíricos para hacerlo. Además, en el proceso de inspección y adaptación, para construir el producto adecuado, el equipo con el PO necesitan observar indicadores de producto, orientarse, decidir y actuar, en busca de maximizar el valor: “maximizar el outcome con el mínimo output”. En esta vía, el PO tiene la posibilidad de manejar un conjunto mínimo y suficiente de indicadores, que se los suele llamar KPI's (business outcome KPIs), y que podemos interpretarlos como indicadores claves del impacto del producto o indicadores de resultados del negocio.

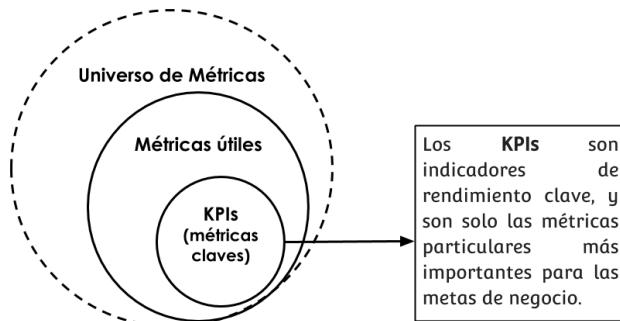


Figura 6.13: Métricas vs. KPIs

Lo importante de los indicadores que elijamos es que deben estar alineados a la visión y objetivos que hayamos definido para el producto. Algunos son los siguientes.

Sh-SAT: Grado de satisfacción de los Stakeholders. 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, con la Review o producto. Este relevamiento relacionado a la percepción de valor entregado se puede hacer en la misma Review. Este es un KPI indirecto e interno del equipo de trabajo.

C-SAT: El Customer Satisfaction es el índice de grado de satisfacción de los usuarios o clientes. Por ejemplo, es el porcentaje de usuarios que respondieron "sí" para encontrar el servicio o producto útil en la encuesta de comentarios en comparación con el número total de encuestados.

NPS: El Puntaje Neto de Promotores (Net Promoter Score) mide la lealtad general de los clientes hacia un producto o servicio (fidelización). Mide qué probabilidad hay de que el cliente recomiende la compañía y, de forma indirecta, la propensión del consumidor a seguir siéndolo y resistir irse a la competencia.

TTM: El Time-To-Market es el tiempo que toma un nuevo producto/servicios desde que fue ideado o comenzó a ser conceptualizado, hasta que llega al mercado, disponible al cliente. El TTM es fundamental para el negocio en un entorno empresarial global altamente competitivo, ya que a un tiempo más corto ganamos una ventaja competitiva. La agilidad ayuda a entregar valor en forma temprana, por lo que este KPI debería mejorar en una empresa ágil.

Revenue: Los beneficios o ingresos obtenidos se suelen medir como los ingresos en un periodo dado. Hay que tener en cuenta que esta medida puede ser multifactorial, o sea que el resultado de su aumento o disminución puede ser causado por múltiples agentes (como marketing). Por este motivo, tener un enfoque reduccionista basado exclusivamente en el beneficio, no necesariamente se correlaciona con el valor realmente entregado.

Cost Saving: La reducción de costos es el impacto monetario resultado de economizar recursos o reducir gastos. Hay productos o servicios que impactan en la disminución de costos, prescindiendo de recursos, insumos o servicios que representan un costo para la organización. Como por ejemplo los costos de contact center, arriendo de infraestructura, licencias de software, horas hombre, retrasos (cost of delay), desperdicios (por errores o merma), gasto de un servicio de un tercero, etcétera.

Conversion Rate: La tasa de conversión (CR) mide cuántos usuarios de los que entran a la aplicación hacen la acción objetivo querida, es decir que, es el porcentaje de usuarios que realizan una acción específica, como una consulta, compra, descarga, registro o una reserva. Por ejemplo, si 20.000 visitas generaron 300 ventas, entonces la CR es $300 / 20.000$ (objetivos conseguidos / número total de visitas), igual a 1,5%. También nos sirve para hacer un embudo de conversión en una aplicación que tiene varios pasos o acciones objetivos principales.

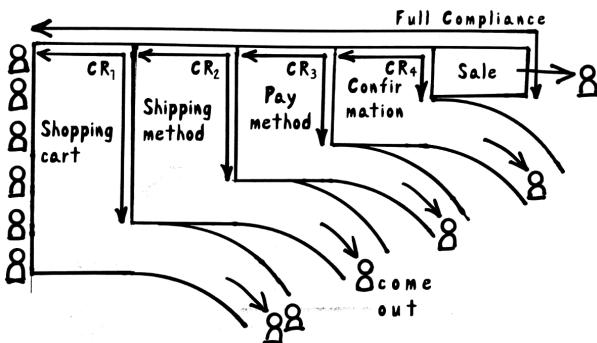


Figura 6.14: Embudo de conversión (conversion funnel)

Full Compliance: La tasa de finalización o cumplimiento es una tasa de conversión que mide el porcentaje de usuarios que finaliza con éxito o generan un resultado exitosamente, después de hacer un conjunto de acciones desde un inicio. Se calcula dividiendo el número total de interacciones completadas con éxito por el número total de interacciones iniciadas. También hay quienes lo llaman "Completion Rate", como el porcentaje de visitantes a un sitio web que cumplen el objetivo del sitio, por ejemplo, tasa de finalización de compra (ver fig. 6.14).

Volume of Use: El volumen de uso representa a cuantos usuarios usan nuestra aplicación.

Digital take-up: La aceptación digital o tasa de adopción para el servicio digital²³. Es decir que es el porcentaje de personas que usaron el canal digital en un período dado en relación con el uso de otros canales, por ejemplo, papel o teléfono. Este KPI representa el porcentaje de la audiencia objetivo que se ha alcanzado con el canal digital, calculado como la cantidad de transacciones digitales completadas durante un período fijo, dividido por el número total de transacciones de todos los canales.

²³Performance Dashboard del gobierno de australia (dashboard.gov.au)

Adoption: La adopción, penetración o take-up, es algo similar a la adopción digital, aunque usada en forma más genérica. Indica cuántos usuarios, de los que ya son clientes, usan nuestra aplicación del volumen total que usan dicho servicio en relación a todos los canales. Esto marca la parte del “Volume of Use” que crece por que los usuarios dejan de usar otros canales.

Market Share: Es un índice de competitividad que indica el nivel de desempeño de la aplicación o empresa en relación a su competencia relativa al mercado objetivo. Esta se puede cuantificar de diversas maneras dependiendo del mercado. Por ejemplo, cuántos usuarios, de los usuarios que no son clientes, fueron captados del mercado objetivo. O sea, que puede indicar la cantidad de usuarios nuevos captados de la competencia o del mercado. Esto marca si el “Volume of Use” crece por captación de nuevos usuarios o no. Esta métrica guarda cierta similitud con la penetración en el mercado o IPM (Market Penetration Index) que establece la posición relativa de la aplicación o empresa respecto a la competencia.

Acquisition: Nuevos usuarios que vienen de otros canales o usuarios que bajan y usan la aplicación por primera vez.

Activation: Usuario que se sienten feliz al usar la aplicación y que realmente están conectados con la aplicación o que firman y hacen una orden para hacerse clientes. O sea que son los usuarios que se activan como clientes. De aquí se puede calcular la tasa de conversión de clientes como clientes potenciales que se han convertido finalmente en clientes.

Engagement: grado de compromiso o incremento de órdenes o valor de cuenta por cliente.

Retention: La retención es un KPI de fidelización que indica usuarios recurrentes o clientes que vuelven a usar la aplicación o que firman y hacen órdenes mensualmente, permaneciendo como clientes. De este KPI se puede obtener una tasa de retención o “retention rate” que nos permite

saber el porcentaje de clientes que repiten o se quedan con nosotros año tras año. La forma simple de cálculo es el 100% menos la tasa de deserción.

Referral: Usuarios felices que se lo dicen a sus amigos, es decir que recomiendan la aplicación.

ROI: El ROI (Return Over Investment) es tanto una técnica como una métrica e indica el retorno de una inversión, es decir, la cifra que evalúa el rendimiento del capital en una acción. Pues, nos permite cuantificar económicamente el beneficio. Su fórmula es: $ROI = (IP - CP) / CP$; Donde IP es el ingresos del proyecto o período, entendido como beneficios económicos esperados estimados, y CP es el Costo del proyecto o período como costo de desarrollo.

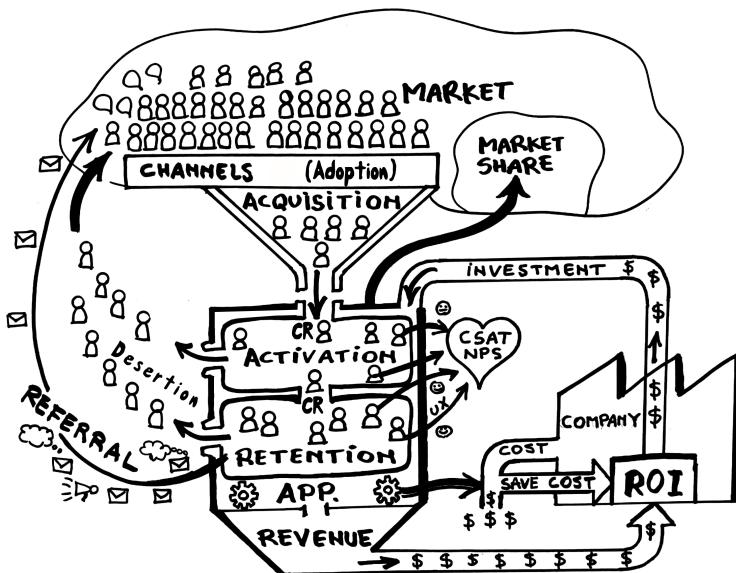


Figura 6.15: Modelo de KPIs de impacto de negocio.

Cabe recalcar que pueden haber muchos otros KPI's, como

pueden ser: MAU (Monthly Active Users), DAU (Daily Active Users), User Stickiness, ARPU (Average Revenue Per User), ARPPU (Average Revenue Per Paying User), Churn Rate, CPS, CLTV, etcétera. También que, los que elijamos, sean lo justo y necesario; y actúen como una brújula, que nos posibilite decidir si debemos continuar con el mismo rumbo o tenemos que cambiarlo.

6.0.5 Dashboard

Las métricas o KPIs, que intervienen en la consecución de los objetivos, son representadas gráficamente mediante el dashboard. Este debe ser útil, simple de entender (limpio y ordenado), visible a todos los interesados y de propiedad compartida del equipo. Debe servirle al equipo, en su conjunto, para monitorear, analizar y pivotar en función de corregir desviaciones o aprender de experimentos.

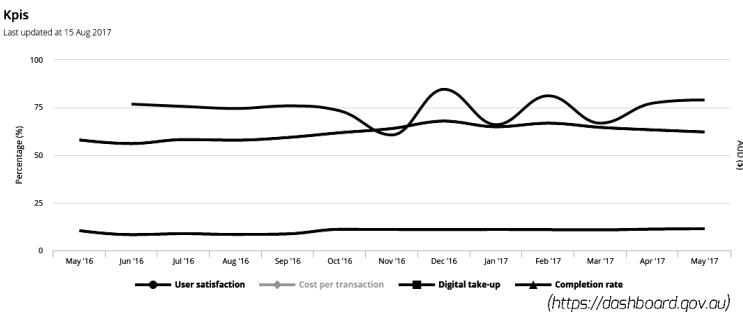


Figura 6.16: KPIs del dashboard del gobierno de australia

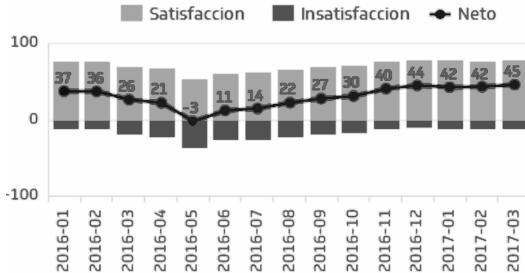


Figura 6.17: Dashboard C-SAT ejemplo

Se recomienda que el equipo mantenga un monitoreo de sus dashboard visible y fácilmente accesible por cualquier stakeholder interesado (como la fig. 6.19). Por ejemplo, a continuación podemos ver un dashboard con el embudo de compra adaptado a un chatbot donde se pueden ver: las sesiones web, sesiones con chat abierto, consultas de productos, agregar al carro y transacciones. Y el equipo puede ver sistemas operativos, usuarios nuevos y la evolución del DAU, WAU y MAU, entre otras cosas.

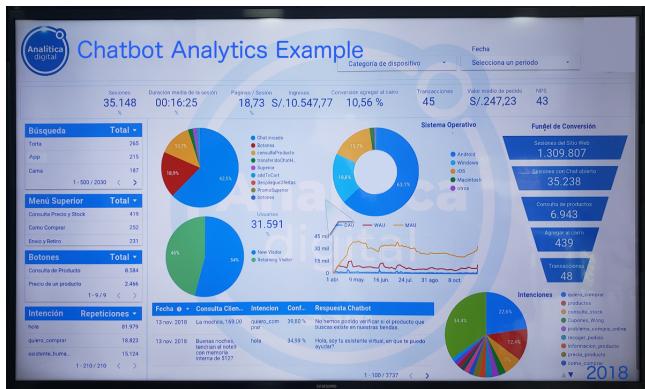


Figura 6.18: Televisor con KPI relevantes para un equipo.

Se puede usar un esquema como el siguiente.

Metas			Q1			Q2			Q3			Q4		
KPI	Valor original	Valor objetivo	1	2	3	4	5	6	7	8	9	10	11	12
KPI 1	Valor	Valor												
KPI 2	Valor	Valor												
KPI 3	Valor	Valor												
Dashboard KPI 1			Dashboard KPI 2						Dashboard KPI 3					

Figura 6.19: Monitoreo de Dashboards

Pero independientemente del formato la información debe ser útil al equipo y a posibles stakeholders interesados.



Figura 6.20: Un equipo (Sierra India en LATAM Airlines) con su Dashboard visible

En próximos capítulos vamos a ver irradiadores visuales que pueden servir como dashboard simples para uso bajo el marco Scrum.

6.0.6 Métricas ágiles

Por último, hay que tener en cuenta que para que las métricas estén bajo un marco Scrum deben respetar los principios de la 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-organize 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). Los KPI's no se deben convertir en objetivos que deben lograrse por el solo hecho de cumplir un plan ni en herramienta de control de empleados. Las métricas son parte del proceso de aprendizaje, de los ciclos de inspección y adaptación, es decir, de aplicar el método científico en la empresa (lean startup).

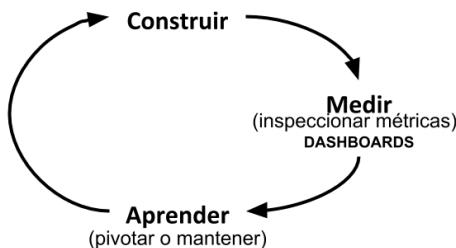


Figura 6.21: Métricas en Lean Startup

Capítulo 7

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.

7.1 Formación de equipos

Cuando iniciamos una organización o migramos desde una tradicional, de management 1.0 orientada a proyectos, a una ágil basada en equipos orientados por misión, tal vez uno de los desafíos más relevantes y difíciles es el de organizar equipos. ¿Cómo juntamos a las personas apropiadas en forma sostenida en el tiempo? Para que un equipo sea uno, debe tener un propósito claro y distingible de otro de la compañía. Y, en consecuencia, debería tener todas las habilidades y competencias necesarias para cumplir el propósito de forma sostenida en el tiempo. Cuando ya tenemos equipos Scrum formados, tal vez la tarea sea más fácil. Pues, en "Scaling Scrum" los equipos funcionan como células que, a medida que la organización se expande, se clonian sus estructuras como en reproducción celular. Las estructuras de las células dependen del problema que se quiere resolver en cada caso y según los nuevos propósitos que surjan.

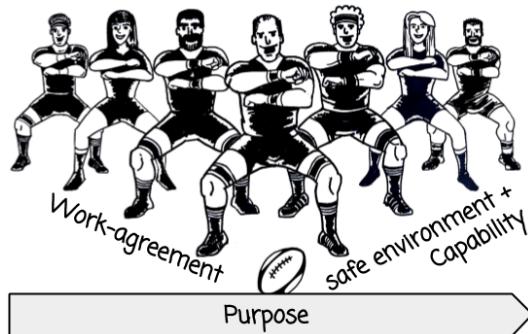


Figura 7.1: Equipo Scrum orientado por misión

De esta idea se desprende la pregunta de ¿qué propósito damos a cada equipo? o ¿cómo estructuramos a los equipos según qué propósitos? Una manera de hacerlo, pensando en productos y en

el sector de producción de la compañía, es tomando una unidad de negocio o producto y ver si su alcance o tamaño es suficiente para un equipo o para varios. Si es para varios, una forma es analizar la jornada o flujo de experiencia del producto, determinar los KPIs involucrados y desglosar en subproductos con sus KPIs asociados. Así cada equipo puede ser responsable de un subproducto y de impactar según los KPIs identificados. Esta es una manera de hacerlo de arriba hacia abajo y orientados a producto. Luego, otra pregunta que surge es: ¿cómo los estructuramos? Bien, si el equipo es de producción como el desarrollo de software de productos, con trabajo planificable y divisible en lotes, Scrum viene como anillo al dedo. O sea que necesitamos a un SM y un PO en el equipo. Un equipo Scrum puede estar formado por diferentes roles, tales como: UX/UI, SM, PO, BA, QA, FE Dev, BE Dev, etcétera. Por ejemplo, un equipo puede estar formado por: PO, SM, UX/UI, 3 FE Dev y 3 BE Dev; otra estructura puede ser: PO, SM, BA, QA, 2 FE Dev y 2 BE Dev. O sea que podemos tener una infinidad de configuraciones. ¿Cuál puede ser la apropiada? No solo depende del propósito, sino de cómo desarrollaremos ese propósito o cuán autónomos somos en relación a él. En esta vía tenemos tres lineamientos básicos de configuración: equipos Scrum de características, equipos Scrum de componentes o uno mixto. Y cada una de estas modalidades pueden tener diversas configuraciones de roles.

7.1.1 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"¹ con características de miembros full-stack, 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 7.2). O sea que cada equipo trabaja sobre determinadas características de producto (Features) o PBIs

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

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 multidisciplinares. Para lograr esto se debe conformar una organización de aprendizaje donde los equipos practican el aprendizaje continuo, donde aprenden para abarcar los componentes arquitectónicos.

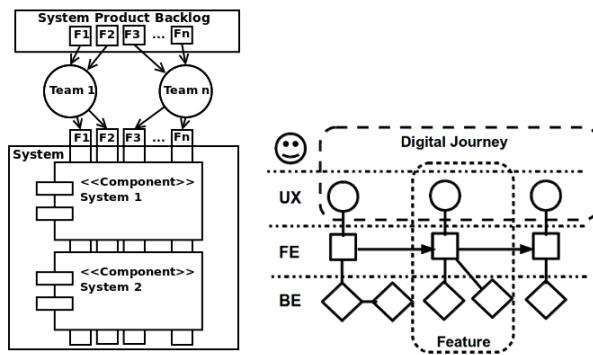


Figura 7.2: Esquema de equipos de características

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].

7.1.2 Equipos de componentes

Abordar el problema de proyectos grandes con "Equipos Scrum de componentes" (ver figura 7.3) 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).

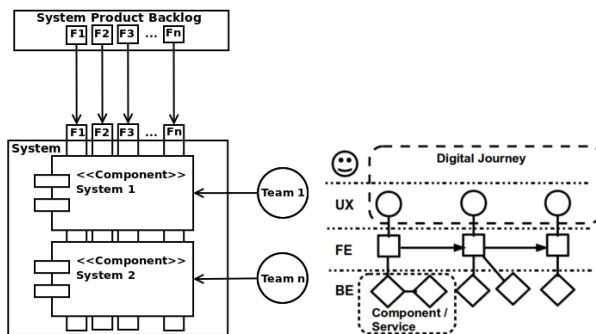


Figura 7.3: Esquema de equipos de componentes

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 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"² que sugiere que las organizaciones pueden replicar su arquitectura en los productos que ellas producen [Martin Fowler, 2014].

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

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.

7.1.3 Equipos mixtos

También es posible la configuración de equipos mixtos que son la combinación de equipos de características y de componentes (ver figura 7.4).

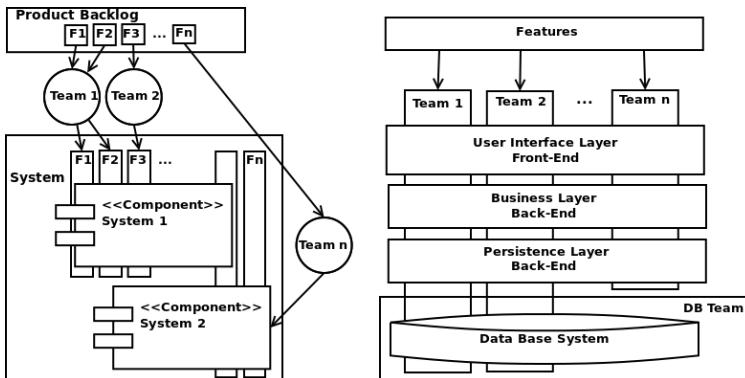


Figura 7.4: Ejemplo de equipos mixtos.

7.2 Integración

7.2.1 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 portfolios. 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-equipo. 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.

7.2.2 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"³ o SoSoS, que consiste en una reunión frecuente a un nivel superior de la SoS.

³[SBOK, 2013]

7.2.3 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 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.

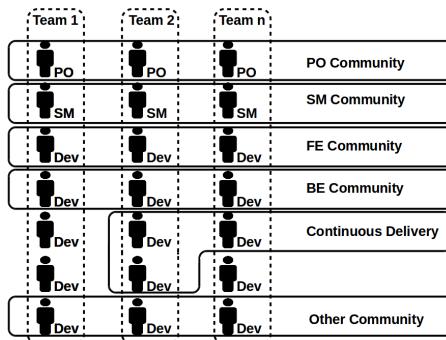


Figura 7.5: Ejemplo de comunidades

7.2.4 Integración UX

Un desafío a la hora de hacer Scrum es incorporar UX en el desarrollo incremental por sprints. Se pueden adoptar, entre otras, dos estrategias principales: desarrollo UX distribuido y gobernado; o desarrollo UX centralizado (equipo externo).

Desarrollo UX Distribuido y Gobernado

Una estrategia es tener UXer, como miembros de equipo Scrum, distribuidos en cada equipo. Esto permite empoderamiento y

velocidad, pero puede generar una UX poco unificada o con poca “integridad conceptual”. Para resolver este problema de unificación se puede acudir a la estandarización UX. La misma podría ser generada y mantenida por los equipos o mediante un equipo externo responsabilizado de la integridad conceptual y experiencia unificada (equipo de estandarización UX transversal) y usar un sistema de versionado de estándares que sirve como otra herramienta de coordinación.

Desarrollo UX Centralizado

Otra estrategia es tener un equipo externo de UX que agrupe a todos los UXer y que provea, como un equipo de servicios, los diseños UX a los equipos de desarrollo. Sin embargo, es probable que el equipo central se convierta en un “cuello de botella” para los equipos de desarrollo y, además, pueda generar más “dependencias” que deben ser abordadas. Para mitigar estos problemas, un modelo híbrido a menudo puede ser aplicado, además de mecanismos de coordinación entre el equipo central y los equipos restantes.

Mecanismo de coordinación de tarea UX en una historia

En ambas propuestas es necesario coordinar e integrar el flujo de trabajo UX con el flujo de desarrollo Scrum. Esto sucede principalmente porque el flujo de trabajo UX no suele entrar dentro de un sprint corto (dos semanas) para que las historias puedan ser finalizadas. Lo que se puede hacer, es que el trabajo UX sea previo al sprint (adelantado al sprint) como parte de la etapa de refinamiento, o como etapa previa al compromiso de planning de sprint. La historia no se termina de refinar o no se compromete en un sprint, hasta que no se considera el trabajo UX lo suficientemente maduro como para que sea abordada para desarrollo. Una manera de hacer esta sincronización es agregando una cláusula de completitud de tarea UX al DoR de la historia. Y, además, asociando a las historias tareas UX que, a su vez, tengan DoR y DoD propios. De este modo, una tarea UX no se considera terminada, para que su historia asociada sea abordada

en un sprint, si no cumple su “DoD UX”. Además, para trabajar a sprint adelantado hay que considerar que antes del sprint 1 debería hacerse un sprint 0 (sprint metáfora y/o inception).

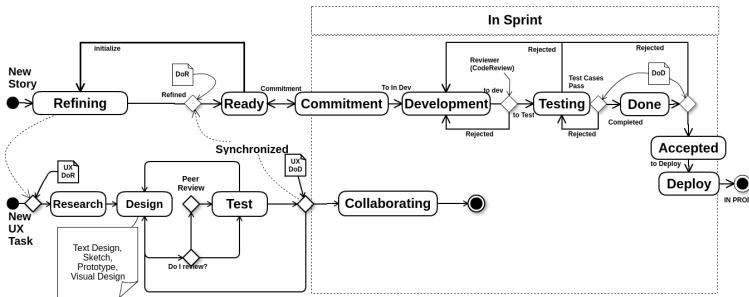


Figura 7.6: Ejemplo de coordinación entre flujos de tarea UX e historia

7.2.5 Scrum y 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 ágil⁴.

⁴[Sanjeev Sharma and Bernie Coyne, 2015]

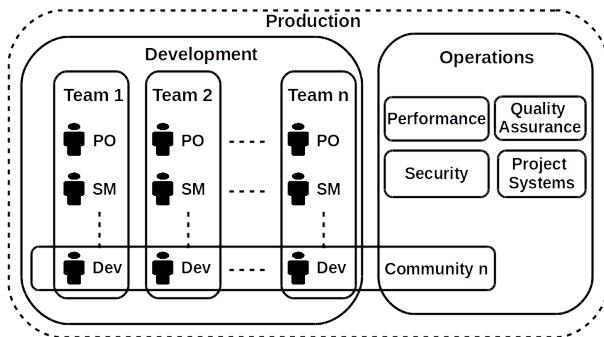


Figura 7.7: Ejemplo de integración de equipos en DevOps

Como primer estrategia, los propios equipos pueden responsabilizarse de DevOps y de su propio pipeline de Continuous Integration y Continuous Delivery, ya sea trabajando colaborativamente entre los Devs, Testers y Ops o con perfiles de trabajo DevOps. Algunas organizaciones usan esta estrategia, la de no contar con equipos de desarrollo y operaciones separados. En su lugar, tienen un único equipo “NoOps” que asume ambas responsabilidades.

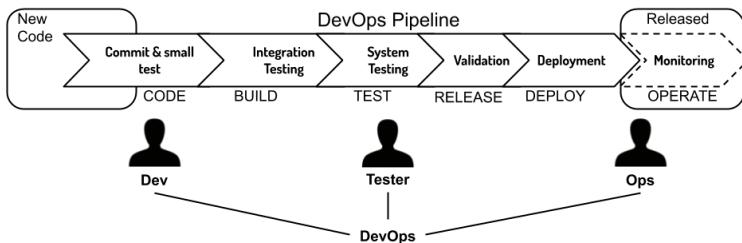


Figura 7.8: DevOps=Dev+Tester+Ops

Por otro lado, como segunda estrategia, se puede mantener el área de operaciones (o equipo DevOps o de plataforma e infraestructura) el cual 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. Además de mantener herramientas compartidas, misma cultura, comunidades en común, procesos ágiles compartidos, comunicación estandarizada, etcétera. Este equipo, más que trabajar como auditor, rector y bajo demanda (un modo push) deberían trabajar más en un modo facilitador, líder y como soporte (un modo pull) de los equipos de desarrollo, promoviendo un sistema de autogestión (donde los equipos puedan autogestionar sus sistemas DevOps incluyendo su plataforma e infraestructura).

De esto se desprende una tercer estrategia plausible, que consta en que la organización tenga equipos DevOps dedicados e intermediarios, que desarrollan herramientas comunes, crean estándares, solucionan conflictos y fomentan la colaboración interequipo. Estos equipos funcionarían como centro de excelencia DevOps sin añadir burocracia ni convertirse los dueños de DevOps, sin tener que hacer frente a todos los problemas relacionados. Porque se podría transformar en un cuello de botella por sobrecarga.

DevOps, como parte de ingeniería de software, es sólo un paso importante para unirse a la cultura general de la colaboración ágil e involucrar a todas las disciplinas, en una organización, en busca de entregar software funcional en entregas más pequeñas y frecuentes.

7.3 Modelos de escalamiento

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

7.3.1 Scrum@Scale

El framework Scrum@Scale fue creado por Jeff Sutherland para integrar múltiples equipos, que crecen en forma orgánica

y sostenible, y se basa en formar estructuras de equipos (distribuidos) integradas en organización fractal mediante dos mecanismos de reuniones: escalar SoS de SMs (SoS, SoSoS y Executive Action Team) y escalar SoS de POs (CPO, CCPo y Executive Meta Scrum). Para más información se puede consultar la “Scrum@Scale Guide” (ScrumatScale.com).

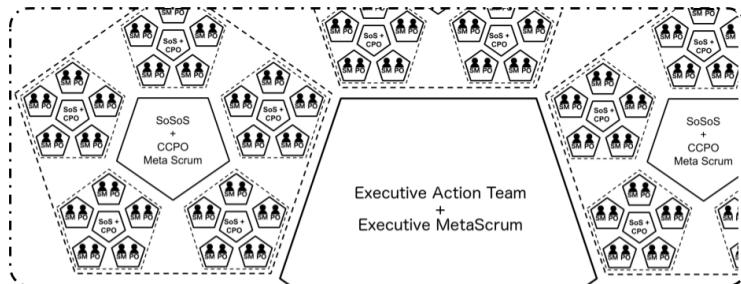


Figura 7.9: Modelo Scrum@Scale

7.3.2 Spotify

Spotify es un caso de estudio que demuestra que puede ser aplicado más allá de pequeñas empresas o startup, sino en empresas más grandes (Spotify constaba en 2014 de aproximadamente de 250 personas en tres países). Por este motivo es un modelo o caso de referencia. El caso de Spotify consta de equipos escuadrones (squad), equivalentes a Equipos Scrum, con un PO y un SM llamado también Team Facilitator (Coach del equipo). Los escuadrones relacionados se agrupan en tribus (tribe) de no más de 100 personas, conformando un área de producto (como por ejemplo un tipo de producto) o área funcional (como por ejemplo infraestructura). La tribu tiene un líder de tribu (Tribe lead) quien se encarga de asegurar un alto rendimiento de la tribu, dar soporte a los líderes de los capítulos, facilitar y entrenar a los SM y construir liderazgo dentro de la tribu.

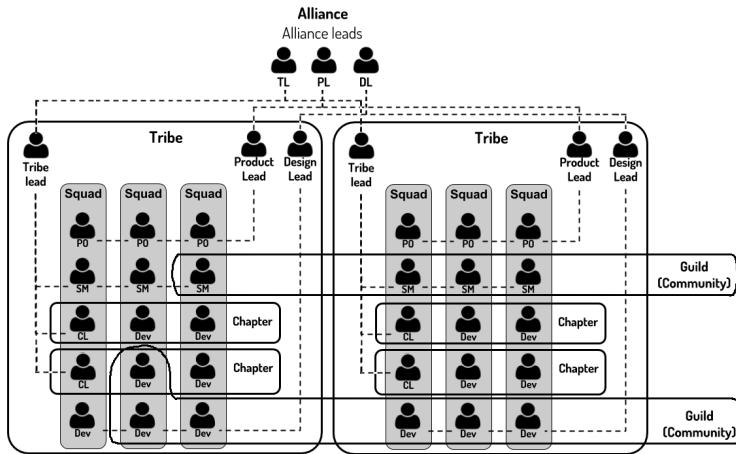


Figura 7.10: Modelo organizacional Spotify

La tribu también tiene un líder de producto (PL) y uno de diseño (DL). Dentro de cada tribu hay capítulos (Chapter) y cada uno engloba un área funcional técnica de ingeniería, agrupando a los miembros de diferentes escuadrones con habilidades similares y que trabajan dentro del mismo área de competencia (por ejemplo el capítulo QA, BE, FE, etc.). Cada capítulo tiene un líder del capítulo ("Chapter lead" o CL) quien se encarga del desarrollo profesional, la cultura de la ingeniería, el apoyo del escuadrón y en asegurar la contratación de las personas adecuadas. Los líderes de capítulo suelen ser desarrolladores a tiempo parcial, por lo general se sientan en uno de los escuadrones de la tribu. Las diferentes tribus se pueden relacionar de diferentes maneras. Dos de ellas son la alianza (Alliance) y las comunidades (Guild). Una comunidad es una "comunidad de interés" (explicadas anteriormente). Los capítulos siempre son locales para una tribu, mientras que una comunidad generalmente es transversal a varias tribus. Cada comunidad puede tener un "coordinador de comunidad" y su alcance es más flexible. Y por último, la alianza une a diferentes comunidades por cohesión funcional de producto y tiene tres

líderes serviciales (TL, PL y DL) quienes facilitan el trabajo de los líderes de las tribus unidas y el funcionamiento de las tribus, en general.

7.3.3 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. SAFe propone que los equipos deben tener sus iteraciones sincronizadas y a tiempos periódicos deben tener una iteración de integración y planificación de todos los equipos o productos (se planifican trenes de releases). Para más información hay que remitirse a la guía y definición de prácticas dada por el sitio web de SAFe.

7.3.4 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.

7.3.5 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.

7.3.6 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.

Si bien estos modelos tienen distintos grados de aceptación y éxito en muchas organizaciones, no se deben tomar como ley en piedra aplicándolos en cualquier contexto u organización. No se deberían implementar como recetas a seguir o soluciones ideales a implantar. Para aplicarlos en una organización deben ser evaluados y, en caso de aplicar alguna de sus pautas, ajustados

según contexto y necesidades. Se recomienda aplicar también el pensamiento sistémico, ingeniería de sistemas y perspectivas de otras disciplinas.

7.4 Equipo de mejora continua

Como se ha visto, el escalamiento no es tarea simple y no existe una solución universal. Se puede dejar que los cambios organizacionales emergan de abajo hacia arriba (desde los equipos, comunidades o individuos), sean dirigidos por un ente director (coaches, consultoras o un líder transformacional) o algo mixto. Hay frameworks que recomiendan que, cuando se desea llevar a cabo transformaciones organizacionales hacia la agilidad, se constituya un equipo de expertos ágiles llamado centro de excelencia, centro de transformación digital, centro de soporte ágil o equipo transversal de mejora continua⁵. Este equipo se encargaría de hacer coaching a equipos y a otros roles, proponer iniciativas de mejoras organizacionales y apoyar la transformación cultural. El equipo puede estar formado por Coaches Ágiles Empresariales, gerentes ágiles, ingenieros de sistemas, ingenieros de operaciones, capital humano, etc. Los miembros deben ser capaces de trabajar en cualquier nivel de la organización, teniendo una visión multidisciplinaria, global y perspectiva sistémica.

⁵ScrumStudy recomienda un Scrum Guidance Body y DAD recomienda un equipo de excelencia CoE.

Capítulo 8

Complementos de Scrum

Originariamente Scrum no tiene como objetivo dar instrucciones precisas a los equipos sobre la forma concreta en la que deben llevar a cabo su desarrollo. El Scrum propuesto por la Scrum Guide de Scrum.org es bastante simple y básico (ver fig. 4.2). Con este marco se espera que los equipos hagan lo que sea necesario para ellos, en función de entregar el producto de calidad. Esto significa mejorar Scrum o ir más allá.

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. Es así como los equipos a medida que maduran agregan eventos y actividades necesarias para su contexto conformando un flujo de Scrum Extendido y adaptado a su realidad (ver fig. 8.1). Además, estos equipos adoptan prácticas ágiles e ingenieriles necesarias para desarrollar software de calidad.

Desde esta perspectiva, se puede consensuar un anillo de complementos útiles que rodea al 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 capítulos trataremos estos temas.

8.1 Scrum extendido

La guía de Scrum propone un marco simple y algo estructurado y, aunque funciona bien como contenedor para otras técnicas, metodologías y prácticas, hacer Scrum es al menos seguir ese marco. Sin embargo, debemos usar lo necesario que en nuestro contexto nos ayude a solucionar los problemas y cumplir los objetivos que nos planteamos. Por tal motivo, debemos ser libres de tomar lo que consideramos relevante de scrum, o mezclarlo con otros métodos o expandirlo con otras actividades, técnicas y prácticas.

En mi experiencia he visto como equipos han necesitado robustecer Scrum con otros eventos, actividades y procesos como la inception, el refinamiento, demos internas, monitoreo constante de feedback, proceso de OPCON, break, etcétera.

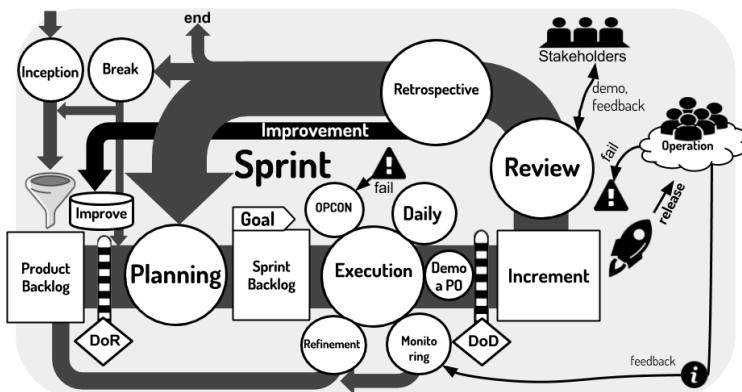


Figura 8.1: Flujo de Extended Scrum

Por ejemplo, cuando un equipo ya constituido comienza con un nuevo producto, es aconsejable tener un proceso inmediatamente previo a comenzar los sprints para definir o poner a todo el equipo en la misma página respecto a la visión del producto, los objetivos y para poder hacer una primera alimentación formal y colaborativa del backlog. En esta reunión es cuando se hace el

kickoff, se definen los tipos de usuarios, experiencia de usuarios a alto nivel, visión del producto, features principales, posibles temas o épicas, un posible roadmap, y las historias sobre las cuales se puede comenzar a trabajar. A este proceso se lo puede llamar “inception”.

Otro ejemplo es el refinamiento o “refinement” que no es un evento oficial de Scrum, sin embargo se puede hacer necesario formalizar el refinamiento como evento para fortalecer el trabajo en equipo sobre el backlog y llegar con historias más maduras (con menos riesgo) a la planning. De hecho, las historias y el DoR no son parte dictada por la guía de Scrum, pero es de bastante utilidad trabajar con historias (como se verá más adelante) y asegurarse de que hay un mutuo entendimiento de la completitud necesaria de una historia para ser abordada en una planning.

También podemos considerar hacer demostraciones “Demo” al PO previas a la review, como instancias donde el equipo junto al PO controlan el DoD y los criterios de aceptación, para controlar una historia antes de que se llegue a la review. De este modo se llega a la review con certeza de cuáles historias se concluyeron satisfactoriamente y cuáles no, evitando que la review se transforme en una instancia de aprobación de historias.

El monitoreo constante de feedback (“Monitoring” o Continuous Monitoring) puede transformarse en un proceso pautado, en el cual el equipo se autogestiona para realizar inspección y adaptación constante, monitoreando las principales métricas del producto, KPIs y registros de actividad del producto funcionando en operaciones. Esto es radicalmente importante para pasar de ser un equipo reactivo a proactivo. Aunque también se debe contemplar ser lo suficientemente efectivo en las acciones reactivas de solución de contingencias en operaciones (resolución de fallos o bugs en operaciones). Y para ello se puede tener un proceso consensuado de cómo manejar las OPCON de forma eficiente.

Trabajar con Scrum se puede sentir, en ocasiones, la disciplina de la cadencia como sofocante. En algunos contextos puede ser producente tener un respiro para que el equipo trabaje en algo fuera del foco del producto o para trabajar en actividades como refinar el backlog, capacitaciones, innovación, planificaciones,

teambuilding, discovery, etcétera. Para eso se usa el concepto de "break", un lapso de tiempo para hacer estas actividades sin contar como sprint. Una especie de interludio inter-sprints.

En síntesis, debemos ser libres de adaptar Scrum o potenciarlo como consideremos apropiado. Ser libres de experimentar.

8.1.1 Definición de preparado

Otro ornamento ya mencionado anteriormente pero no explicado es el "Definition of Ready" o DoR. La definición de preparado es un tipo de Definición de Terminado particular (previo al desarrollo) que, si bien no es parte de la guía de Scrum original, es particularmente útil. Pues, se corresponde con las condiciones para que una historia (PBI) pueda pasar a formar parte de un Sprint Backlog. Si una historia 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. Es decir necesita refinamiento. O sea que, la definición de preparado permite entonces tener un punto de acuerdo entre el PO 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:

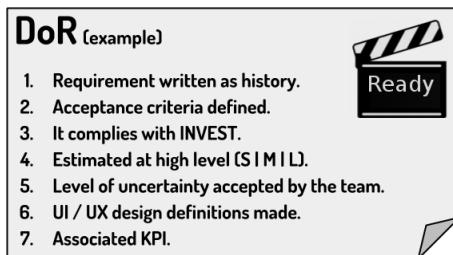


Figura 8.2: Ejemplo de una DoR

8.2 Dinámicas de grupo

8.2.1 Coordinar el silencio

Una técnica usada por rescatistas después de terremotos para hacer silencio es particularmente útil para lograr silencio en un grupo numeroso. Consta de pedir silencio cuando se levanta la mano. Pues, 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.

8.2.2 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.

8.2.3 Constitución de un equipo Ágil

No hay nada mejor para iniciar un equipo ágil que comenzar con motivación, foco y compromiso, para lo cual una dinámica de “Constitución de un equipo Ágil” es útil. Pues mediante esta técnica se busca lanzar un equipo que tenga identidad como tal, forma de trabajo clara, acuerdos de convivencia, un propósito como foco de equipo y valores que guíen su espíritu.

Es una inversión de esfuerzo inicial en un grupo para lograr cohesión, promoviendo su sinergia, para el éxito futuro como equipo. En esta reunión de constitución se recomienda comenzar con alguna dinámica rompe hielo y seguir con una de conocimiento interpersonal, como por ejemplo la dinámica de “mapa personal” (personal map). Luego se puede hacer la dinámica de “Lienzo de Constitución de Equipo Ágil” (ver figura 8.3).

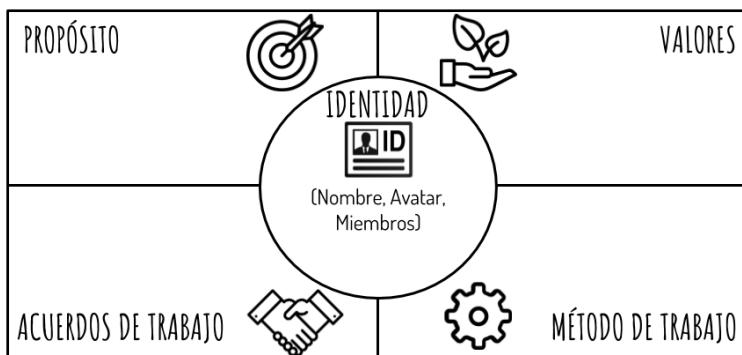


Figura 8.3: Lienzo de Constitución de Equipo Ágil

En esta dinámica, mediante un lienzo de cinco regiones, completamos la identidad del equipo, su propósito como tal, los valores del equipo, los acuerdos de trabajo (working agreement) y el método de trabajo. La identidad del equipo puede constar de un nombre y un avatar. También se pueden escribir los nombres de cada integrante para que queden registrados como acta de constitución. El propósito debería ser simple y claro, que cohesionne la finalidad del equipo. Aquí podemos recordar que es deseable que se constituyan equipos a los cuales se les da trabajo o se les trae productos o proyectos y no al revés, armar equipos para un proyecto en particular para luego desarmarlos. Es recomendable orientarse a productos o servicios con equipos estables que a proyectos. Respecto a los valores, además de hacer énfasis en los valores ágiles y los de Scrum, el equipo puede definir sus propios valores o agregar valores a seguir. Los acuerdos de trabajo son esenciales para marcar las reglas del equipo como

acuerdos de convivencia. Aquí se puede agregar la hora de comienzo de la daily, por ejemplo. Y por último, en método de trabajo se aclara el uso de Scrum, la duración del sprint y se pueden especificar algunas prácticas a seguir.

8.3 Técnicas de justificación de negocio

8.3.1 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.

8.3.2 Valor de negocio 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 8.4) o Business Value BurnUp¹. Además, también es útil para ayudar en la priorización de historias de usuario, pues logrando hacer un diagrama de Business versus Complexity (Ver figura 8.4) 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.

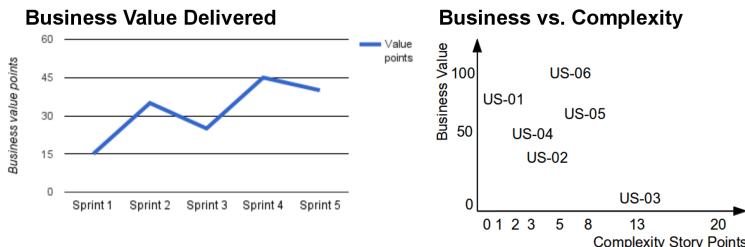


Figura 8.4: Diagramas de Business Value

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

8.4 Técnicas para requerimientos

8.4.1 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"². 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 ³. 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 ⁴.

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" ⁵. 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 negocio y respetando la siguiente pauta de estructura:

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

Por ejemplo:

²Las "User Story" son una técnica de eXtreme Programming (XP)

³[Scrum-Institute, 2015]

⁴[Scrum-Institute, 2015]

⁵[UNTREF, 2014], [Dan North, 2015]

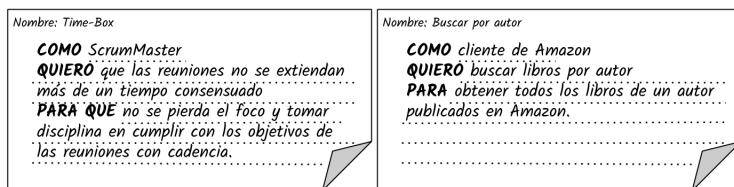


Figura 8.5: Escritura de historias ejemplo

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 conversación⁶.

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 negocio⁷.

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"⁸.

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 criterio de demarcación aceptado en metodología ágil en general es el criterio INVEST⁹ y que consta de seis características a cumplir

⁶[UNTREF, 2014]

⁷[Scott Bellware, 2008]

⁸[Dan North, 2015]

⁹INVEST es un acrónimo creado por Bill Wake.

¹⁰:

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"¹¹ 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 ¹². 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 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.

¹⁰[UNTREF, 2014], [Scrum Alliance, 2015]

¹¹"Las user stories no son obligaciones contractuales"[Cohn, 2004]

¹²[Scott Bellware, 2008]

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 User Story. Por ejemplo una tarea puramente técnica (que le interesa sólo al desarrollador), un refactoring técnico, deuda técnica, una mejora técnica o un Spike no cumplen con INVEST y en su generalidad no son User Story. Sin embargo, a todas las historias que no son tareas o spike las podemos tratar como Story, haciendo la distinción entre User Story y Technical Story (ver fig. 8.6).

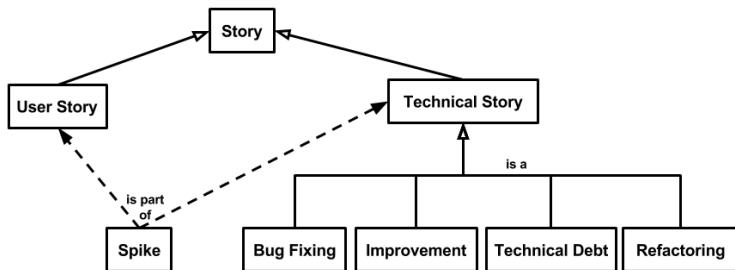


Figura 8.6: Clasificación de historias

Veamos los siguientes ejemplos:

- **Technical Story:** “Cada historia tiene que ser valorada por los usuarios. Pero eso sería un error.”¹³ Pues en realidad, una historia de usuario describe la funcionalidad

¹³[Cohn, 2004]

que será valiosa para un usuario o comprador de un sistema o software¹⁴. 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 técnico¹⁵, 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"¹⁶.

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 desarrollo¹⁷.

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 usuario¹⁸.

¹⁴[Cohn, 2004]

¹⁵[Scott Ambler, 2015]

¹⁶[Scott Ambler, 2015]

¹⁷[Cohn, 2004]

¹⁸[Scrum Alliance, 2015]

- **Spike:** No es una User Story ya que es una tarea de investigación y/o experimentación en formato timeboxing, por lo que no cumple con ser probable ni estimable (no cumple con dos criterios INVEST); puede ser redactada como una historia de investigación¹⁹, como por ejemplo: **COMO** desarrollador **QUIERO** investigar sobre **PERFORMANCE PARA QUE** luego se pueda mejorar la prestancia de la aplicación. Un Spike debe ser una tarea corta, preferentemente se trata de que no dure más de 6 horas.
- **Refactoring:** Es una tarea técnica de reingeniería, mejora técnica (Improvement) o deuda técnica (Technical Debt), que no es valioso 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"²⁰. 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". Puede ser categorizada como historia técnica. 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

¹⁹[Cohn, 2004]

²⁰Deuda 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.

de usuario) del Sprint, es decir, el equipo puede mantener un factor de enfoque en historias de usuario para asegurar tiempo para corregir errores²¹. Hay que tener en cuenta que la corrección de uno o varios errores puede ser tratada simplemente como cualquier otra historia²² (aunque no sea exactamente una historia de usuario).

Componentes de una historia de usuario

Las historias de usuarios se componen de tres elementos comúnmente denominados las tres 'C'²³:

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.

²¹[Henrik Kniberg, 2007]

²²[Cohn, 2004]

²³Las 3 Cs de Ron Jeffries.

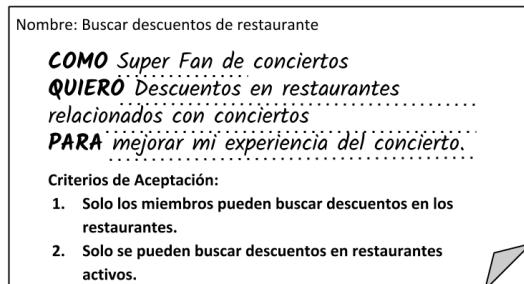


Figura 8.7: Historia de usuario como ejemplo simple

Jerarquía y desglose del Backlog

Otras categorías a tener en cuenta son las abstracciones de alto nivel que permiten jerarquizar el desglose de trabajo en el Backlog. Esta depende del autor al cual uno se refiera y la convención del equipo. Por ejemplo hay quienes quieren trabajar con temas (theme), pero en mi experiencia es mejor manejar pocos conceptos y el de más alto nivel es el producto. A partir de un producto comenzamos a desglosar en épicas. Las épicas se pueden dividir en dos o más historias de tamaño más pequeño. Es difícil trabajar con épicas porque con frecuencia contienen múltiples historias. Generalmente se clasifican en una de dos categorías: historia compuesta o historia compleja (Mike-Cohn, “User Stories Applied”). Hacer este tipo de categorizaciones nos ayudará a hacer mejor el desglose de trabajo y ordenar mejor el backlog. A este modelo lo considero simplificado (ver fig. 8.8). Es simplificado porque no trabajamos con features por separado. Desde un punto de vista más semántico: la features es una parte del sistema que está intentando construir, y la historia es una manera de describir esa parte. Según Kent Beck y Martin Fowler, las features y user stories son sinónimos. De hecho, podemos trabajar con historias que tengan título, donde el título representa a la descripción abreviada de la feature.

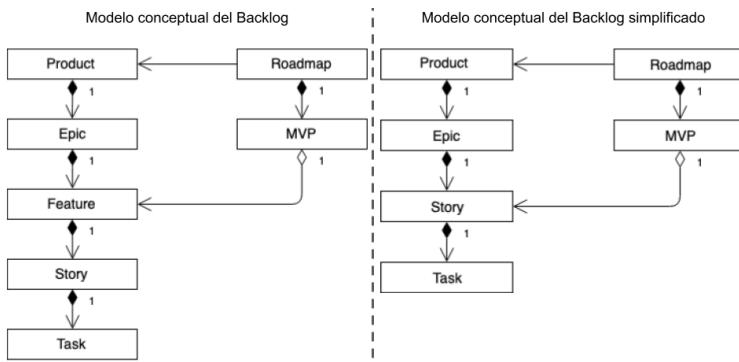


Figura 8.8: Modelo conceptual del backlog

Uno modelo más complejo es trabajar con features (ver fig. 8.8). Podemos usar features como contenedor de historias (ver fig. 8.9). En cualquier modo que elijamos podemos trabajar con MVPs, como parte del roadmap del producto. Los MVPs van a referenciar a features o a títulos de historias. El modelo conceptual que se use dependerá del equipo y el PO, de lo que les sea más cómodo y útil.

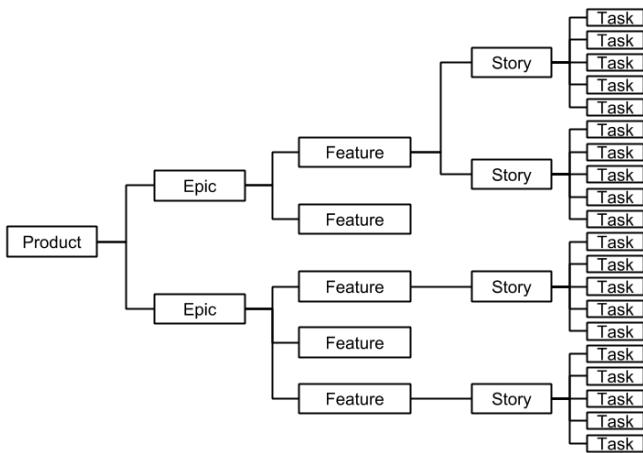


Figura 8.9: Desglose del backlog

8.4.2 Slicing

En la actividad de refinamiento del Backlog, el PO conversa y analiza con el equipo sobre si las historias son lo suficientemente pequeñas para desarrollarse y cumplir con el DoR. En caso de que aún sean demasiado grande podrán subdividirse según algún criterios de un conjunto de patrones de Slicing. Estos patrones ayudarán a encontrar la manera de pasar de historias grandes o complejas a otras más pequeñas y simples, manteniendo siempre la calidad de User Story.

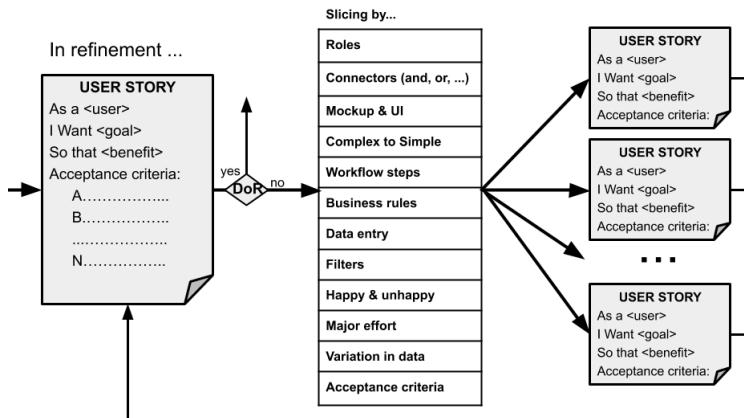


Figura 8.10: Actividad de Slicing

8.5 Técnicas de estimación

El desarrollo de software es una actividad de creación por evolución del conocimiento y, bajo el marco ágil, esta evolución es adaptativa, produciendo valor en lotes pequeños de trabajo. Esto hace que las planificaciones adaptativas sean más a corto plazo, donde las estimaciones de esfuerzo son realizadas con frecuencia para estimar iteraciones y conversar.

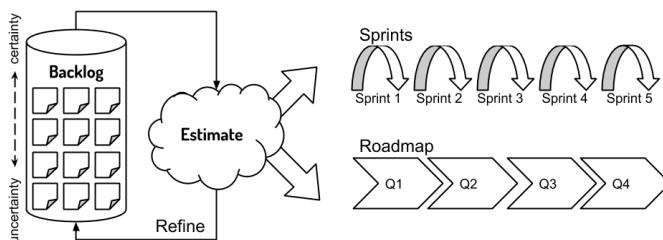


Figura 8.11: Estimación

Hay que considerar, entonces, que bajo el marco Scrum las

estimaciones se hacen con dos razones principales: planificar sprints (sprints, roadmap, etc.) y evolucionar el conocimiento compartido (refinando las historias). La primera está relacionada a la predictibilidad del trabajo en pos de lograr una cadencia sostenible. En este aspecto, la predictibilidad se hace más certera al aumentar el grado de confianza al trabajar con sprint y hacer estimaciones a corta distancia en el tiempo. Además, es parte de un mecanismo de autoregulación del equipo en buscar lograr un flujo de trabajo sostenible, estable y homogéneo. Para ello, el equipo busca desglosar trabajo en esfuerzos equiparables y de tamaños similares. La segunda razón, que se puede considerar la principal para estimar, es la de evolucionar el conocimiento compartido en forma colaborativa. El tiempo insumido en estimar es tiempo aprovechado para charlar en equipo, generando entendimiento común en lo que se tiene que hacer y cómo hay que hacerlo. Si bien el entendimiento común se va madurando en los refinamientos, donde se pueden hacer estimaciones a alto nivel, es en la planning cuando se valida que todos están en la misma página y, desde esta perspectiva, es una excusa para conversar. A continuación comento algunas técnicas elementales.

8.5.1 Estimación relativa

¿Para cuándo va a estar el desarrollo terminado? ¿Cuántas horas vas a demorar en terminar la historia? No queremos estimar en tiempo absoluto, porque estimar cuánto nos vamos a demorar en desarrollar una historia de software, es tan imposible como estimar cuánto dinero exacto vamos a ganar con esa historia. Debido a la complejidad inherente al desarrollo de software es que se recomienda la estimación relativa y que consiste en estimar bajo determinada incertidumbre y con valores relativos. Se suelen usar los puntos de historia o "Story Points", que 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. Hacer estimaciones relativas con SP significa que las historias se comparan entre sí, buscando mantener una relación proporcional entre ellas según estos 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.

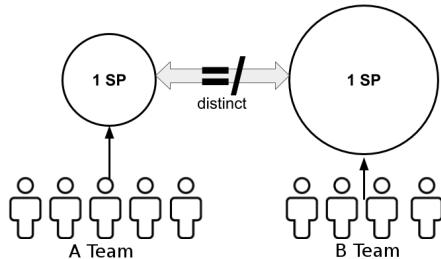


Figura 8.12: Estimación relativa

Cabe recalcar que la estimación es relativa, porque es subjetiva al equipo que estima. Entonces, 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 y que es 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 forma²⁴.

También cabe aclarar que "bajo ningún punto de vista la estimación (en story points) es un compromiso"²⁵ 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.

²⁴[Cohn, 2004]

²⁵[UNTREF, 2014]

8.5.2 Poker de planeación

El Poker de planeación o "Planning Poker"²⁶ 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 estimación

Las Cartas de Poker de Planificación que más se usan son las T-Shirts y las de Fibonacci:

1. **T-Shirts:** Las cartas de talles de camisetas (polera) contempla los valores XS (muy pequeño), S (Pequeño), M

²⁶El "Planning Poker" proviene de un paper presentado por James Grenning 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]

(Mediano), L (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. Este tipo de estimación se suele usar en los refinamientos del backlog y nos puede sugerir que una historia es lo bastante grande como para considerar desglosarla (Story Slicing) o lo suficientemente pequeña para tomarla en un futuro sprint.

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. Este tipo de estimación se suele usar en el planeamiento del sprint para estimar cuántas historias abordar.

8.6 Técnicas de monitoreo visual

Para buscar la adaptación y que el equipo se autoregule, debe monitorear constantemente su trabajo y sus resultados. Aquí se muestran algunas herramientas de monitoreo visual.

8.6.1 Gestión visual

La gestión visual es la utilización de elementos y técnicas visuales, como complemento, para la organización del trabajo y la comunicación o irradiación visual del trabajo. Esto es útil para soporte de coordinación y comunicación 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 para todo el equipo.
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.

8.6.2 Tableros Scrum/Kanban

El tablero Scrum/Kanban o "Scrum Kanban Board" es una técnica de comunicación y monitoreo que consiste en utilizar un tablero Kanban (ver figura 8.13), 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

Los tableros se usan para reflejar el sprint backlog de una manera que sea útil para el equipo. Un buen tablero refleja el flujo real de trabajo y los respectivos WIP asociados a las etapas de trabajo (columnas). Hay equipos que tienen dos tableros: uno para el flujo de historias (story board) y otro para el flujo de las tareas (task board) (ver fig. 8.17). Esta manera permite tener una verdadera visibilidad del flujo de las historias, por un lado, y sus correspondientes tareas, por otro. Otros equipos tienen las historias y tareas integradas en un solo tablero (ver fig. 8.19). La estructura y estándar de colores y nomenclaturas dependerá de cada equipo.

PB	SB	TO DO	ON GOING	DONE	BLOCKED	OK

Figura 8.13: Ejemplo de un tablero Kanban para Scrum.

Backlog (Sprint Backlog)	Impl. (3)	Peer Review (3)	In Test (1)	Done	Deploy ment	In Prod
Fast track						
Blocked						

Figura 8.14: Ejemplo de un tablero Kanban para historias.

Sprint Committed (9)	In Dev.	Testing	Done	Accepted for Deploy (by PO)	In Production
Fast track (OPCON)					
Blocked					

Figura 8.15: Ejemplo de un tablero Kanban físico que usamos con un equipo (Sierra India) en LATAM Airlines.

To do	In Progress								Done
	Analysis (limit)		Development (limit)		Testing (limit)			Delivery	
	Sprint Backlog	In Analysis	Ready for Dev	In Dev.	Ready for Test	In Test	Acceptance (by PO)	Ready for Deploy	Deploy
Stories									
Fast track									
Other									

Figura 8.16: Ejemplo de un tablero Kanban para Scrum sofisticado.

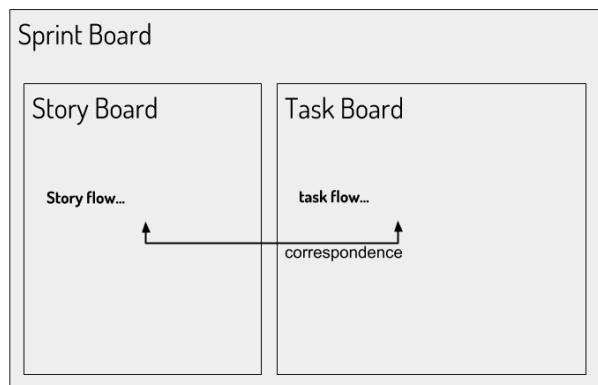


Figura 8.17: Esquema de un Sprint backlog.



Figura 8.18: Ejemplo de tableros físicos en LATAM Airlines 2017.

Ejemplo de un sprint backlog integrado simple

Un tablero de tareas (taskboard) físico (Ver figura 8.19) es una tabla donde se colocan Post-its representando tareas, que pueden estar asociadas a historias. Este se puede integrar al de historias aunque simplifica el flujo real de la historia ya que se centra en las tareas (ver fig. 8.19).

STORY	TO DO	ON GOING	DONE
story	task task	task	task
story	task task task	task	
Blocked			

Figura 8.19: Ejemplo de un tablero integrado de historia y tareas.

8.6.3 Tablero de obstáculos

El Tablero de Obstáculos (ver figura 8.20) 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 o facilitar remover todos los obstáculos que puedan paralizar o ralentizar el trabajo del equipo o desviarlo de su foco esencial. El tablero también es útil para tener la visibilidad del estado de los obstáculos (ver figura 8.20).

OBSTACLE	
NEW	ESCALATE
ASSIGNED	BACK TO TEAM
CLOSED	

Figura 8.20: Ejemplo de un tablero de Obstáculos.

8.6.4 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 8.21.).

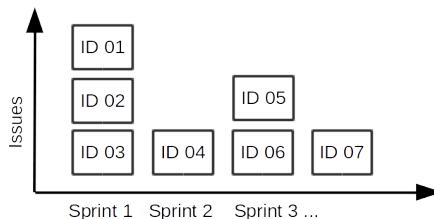


Figura 8.21: Calendario de obstáculos ejemplo.

8.6.5 Gráficos de esfuerzo pendiente

Gráfico Burndown

Un gráfico de trabajo pendiente o "Burndown charts"²⁷ (figura 8.22) es una técnica de monitoreo para la auto-regulación que muestra el trabajo restante ("Remaining Scope") o que queda

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

por hacer versus tiempo. También refleja 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.

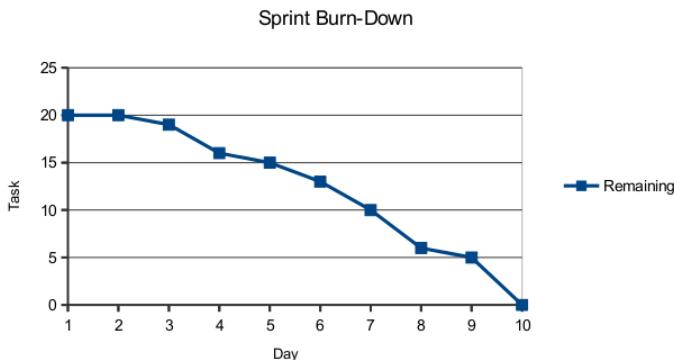


Figura 8.22: Ejemplo de un gráfico Burndown con un sprint de dos semanas y 20 tareas comprometidas.

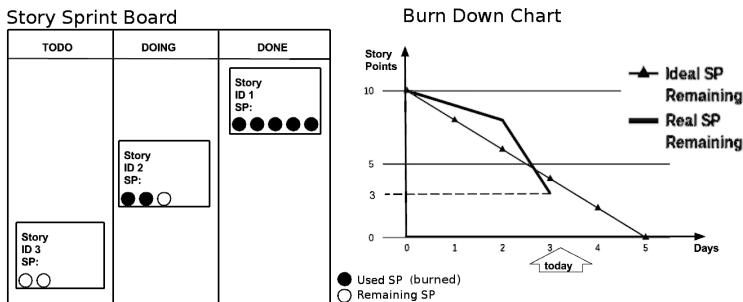


Figura 8.23: Ejemplo de cómo llevar un Burndown con un sprint de 5 días y quedando 3 storypoints por quemar.

Se pueden utilizar dos gráficos de esfuerzo pendiente:

- 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 8.22 y 8.23).
- 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-up²⁸ (figura 8.24) es muy similar al Burndown, con la diferencia de que se parte del cero, y

²⁸El 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 y ver la diferencia entre estimado y real (ver 8.25).

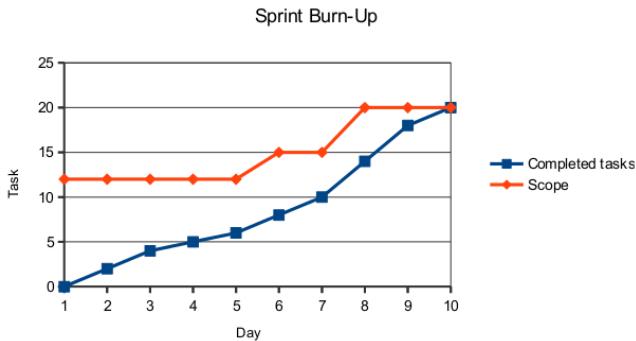


Figura 8.24: Ejemplo de un gráfico Burnup de tareas comprometidas en un sprint de dos semanas.

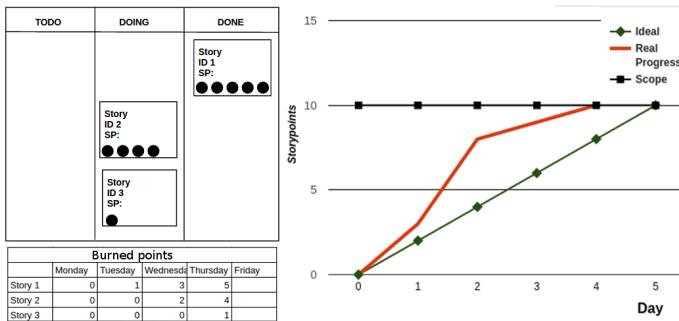


Figura 8.25: Ejemplo de cómo llevar un Burnup con un sprint de 5 días y 10 puntos quemados al cuarto día.

8.6.6 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 realizadas²⁹. También es una herramienta de monitoreo para auto-regulación ante desvíos o caídas del rendimiento y para visualizar la evolución de la capacidad de trabajo del equipo. Un gráfico de la velocidad típico, de un equipo de proyecto ágil, podría ser como el siguiente gráfico de la izquierda (ver 8.26):

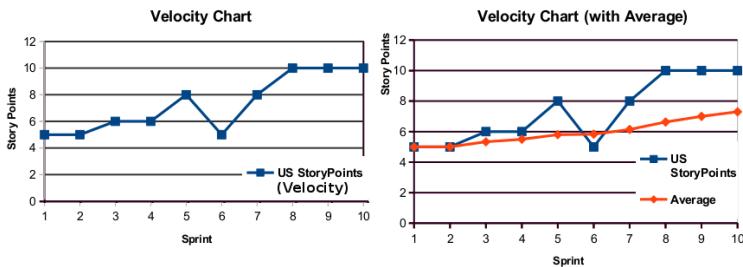


Figura 8.26: Diagrama de Velocidad y Diagrama de Velocidad con Velocidad Promedio.

También se puede incluir la velocidad promedio (ver gráfico de la derecha 8.26).

²⁹[Scrum Alliance, 2014]

8.7 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:

8.7.1 Técnica Pomodoro

Se usa 'Pomodoro'³⁰ 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 primeros 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.



Figura 8.27: Esquema pomodoro

8.7.2 Calidad integrada

Un programador que busca la excelencia técnica es partidario de la calidad desde el inicio o "calidad integrada", algo que en "lean manufacturing" se conoce como "quality built in". O sea que,

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

ser un programador ágil no es entregar rápido a expensas de la calidad y acumulando deuda técnica; ser ágil es entregar buen código, que significa que el código es legible, entendible, cubierto por test automatizado, simple y hace lo que se espera que haga. Las revisiones de código, las técnicas de TDD, los estándares de código, un DoD que contempla QA y la gestión de deuda técnica, son algunas de las prácticas que ayudan al buen código. Un buen libro que trata el tema es “Clean Code, A Handbook of Agile Software Craftsmanship” de Robert C. Martin.

8.7.3 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.

8.7.4 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 pares³¹. Por este motivo, implementar este tipo de 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 a pares"³². 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 mejorarlos 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).

8.7.5 TDD

El "Test Driven Development" (TDD) o "desarrollo guiado por pruebas"³³ es una práctica, técnica de desarrollo ágil y técnica de programación que hace foco en el comportamiento especificado de unidades de software³⁴ 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

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

³²[Boehm, 2001]

³³[Jurado, 2010]

³⁴[Wikipedia 2014]

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 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). Por eso hay que considerar que, si bien se pueden lograr productos de calidad, demanda un costo considerable.

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".

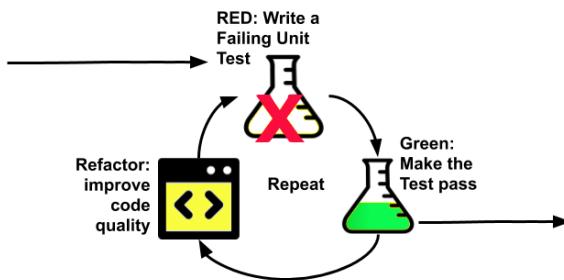


Figura 8.28: Procesos de Test Driven Development

8.7.6 BDD

Una manera de ayudar al desarrollo ágil a partir de requerimientos es usar BDD (Behavior Driven Development). El desarrollo conducido por comportamiento se puede ver como una extensión de TDD. La mejora clave con BDD es la introducción de un lenguaje específico de dominio que es muy accesible para los usuarios de negocios. BDD generalmente estará más en el nivel de prueba funcional ya que es el resultado natural de trabajar con

un lenguaje centrado en el negocio. De este modo el PO se sienta con el cliente y escribe oraciones significativas sobre cómo debería funcionar todo el sistema. Se han usado herramientas software como Pepino para soportar el trabajo con BDD. Hay que tener en cuenta que esta metodología es muy parecida a ATDD. Solo que tiene un alcance algo distinto, ya que BDD trabaja muy cerca del cliente (capa testing end-to-end) y ATDD es más a nivel de la creación de la US (capa testing de integración). Aunque las diferencias son algo difusas. El libro "Specification by Example", dice que ATDD, BDD y otros términos, son distintos nombres para referirse a la misma metodología. Un libro que coincide es "ATDD by Example", añadiendo que también podemos conocerlo con el nombre de Specification by Example, Story Testing o Agile Acceptance Testing. Mi consejo, por simplicidad, es no prestar importancia a las diferencias y centrarse en ATDD si es que decide aplicar esta metodología.

8.7.7 ATDD

ATDD (Acceptance Test Driven Development) es una metodología en la cual todo el equipo discute en colaboración criterios de aceptación de una historia de usuario, con ejemplos, y luego crea un conjunto de pruebas de aceptación en forma de escenarios (por ejemplo en formato Gherkin) en concreto. 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]...

Continuadamente se automatiza la prueba de aceptación de los escenarios que fallarán (fail) en su primera corrida. Luego recién se procede a programar la implementación de la historia de usuario para que pase (pass) los escenarios. Aquí es donde se relaciona con TDD, ya que la implementación del código de la historia se puede

hacer usando la técnica TDD. Este proceso se repite por cada historia de usuario. Las herramientas para trabajar con ATDD incluirían a Cucumber y Fitnesse, entre otras.

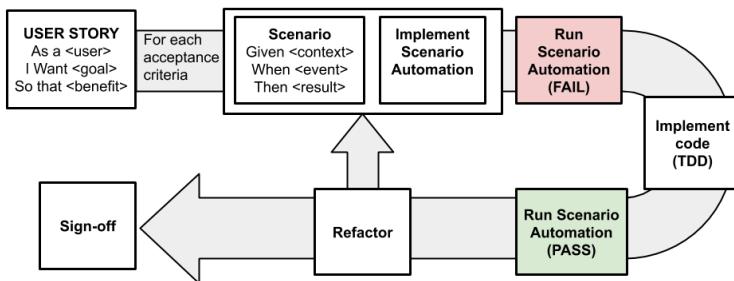


Figura 8.29: Esquema del proceso básico de ATDD

8.7.8 Integración continua

La integración continua (Continuous Integration³⁵) 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.

8.7.9 Entrega y despliegue continuo

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. Tal vez la única diferencia entre Continuous Delivery y Continuous Deployment es que en esta última todo está automatizado y en la anterior incluye un trabajo manual en el despliegue a producción. Trabajar bajo el marco ágil y no trabajar con testing continuo, integración continua

³⁵Continuous Integration fue propuesto inicialmente por Martin Fowler.

ni entrega continua puede conducir a un trabajo poco ingenieril y menos eficiente. Es parte de reducir el time to market y el tiempo de despliegue, facilitar el flujo entre el desarrollo y despliegue, reduciéndo el tiempo lo máximo posible y manteniendo calidad. Aquí el desafío es lograr hacer el desarrollo con TDD (ATDD o DDD), integración y despliegue, todo dentro de un mismo sprint.

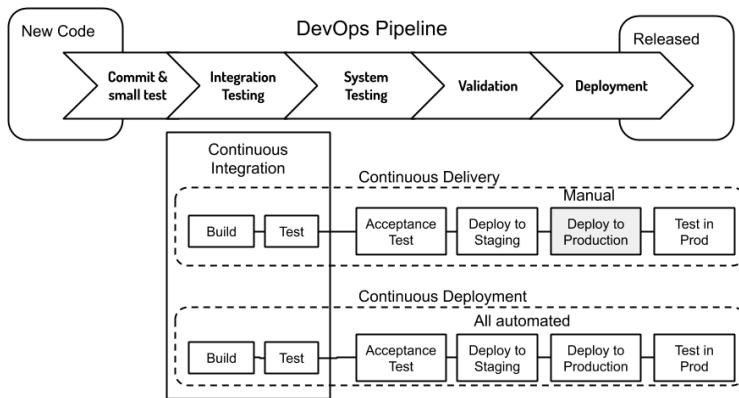


Figura 8.30: Continuous Delivery vs. Continuous Deployment

8.8 Metodologías amigas

El marco Scrum no es excluyente de XP, Lean, Kanban u otras metodologías. Pues, puede combinarse con cualquier otra. A Scrum se lo puede ampliar tomando aspectos de otra metodología o métodos. También se puede buscar ampliar otra metodología o métodos tomando aspectos de Scrum. Por ejemplo, se puede combinar con XP, Lean o Kanban.

8.8.1 Scrum con XP

Si hay dos sistemas ágiles de desarrollo parecidos, son Scrum y Extreme Programming. Con ambos se trabaja con iteraciones, se implementa el juego de planificación (Planning Game) o planning con el cliente o representante del negocio, se busca lograr una propiedad colectiva de código, se trata de lograr versiones pequeñas de producto para entregar rápidamente, se persigue un diseño simple, existe el rol de facilitador (Coach en XP y SM en Scrum), existe un rol que representa al cliente (On Site Customer en XP y PO en Scrum) y ambas comparten los valores de respeto y el coraje. Entre las diferencias se encuentran algunas buenas prácticas técnicas de XP y que pueden usarse bajo el marco Scrum: metáfora de sistema, pairing, continuous integration, TDD, refactoring y estándares de código.

En mi experiencia personal, ha sido bastante común ver de la mano Scrum y XP. Aunque actualmente se ve a XP simplemente como prácticas de ingeniería de software. Se puede sumar XP a Scrum aplicando las prácticas XP en los eventos y actividades de Scrum, incluyendo el desarrollo. La construcción de un modelo mental simple del dominio del problema como metáfora al inicio del producto, en un sprint 0 o en una “inception” (incluyendo posteriormente el uso de DDD) se hace necesario para aportar más certidumbre al inicio y una guía desde donde evolucionar la estructura del sistema. El testing es crucial para construir software de calidad, incluyendo automatización, y TDD es una buena guía al respecto. Para desarrollar en forma incremental y evolutiva, sin acumular deuda técnica y con mejora continua del software, es necesario practicar refactoring. Para fomentar la

colaboración y la propiedad colectiva de código, la programación de a pares es útil. Para asegurar simplicidad y código claro y limpio, además de facilitar la colaboración, es que el estándar de código se usa como acuerdos de programación. Y una manera de formalizar XP en Scrum es agregando prácticas y reglas XP en el DOD. Es así como en ingeniería de software con las prácticas XP se potencia a Scrum.

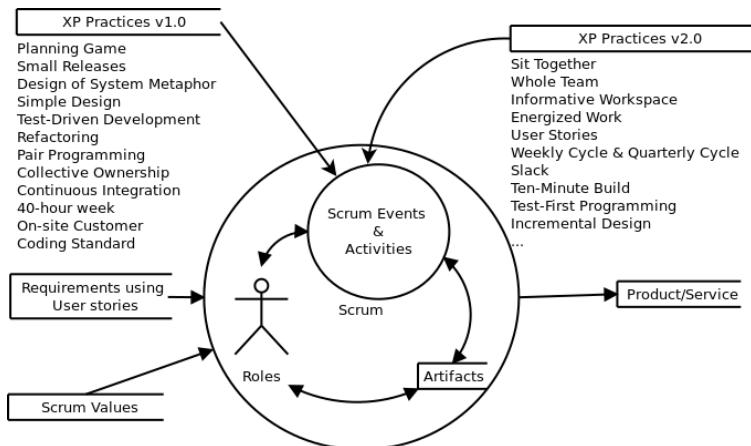


Figura 8.31: Scrum más XP

Si quieras saber más sobre XP puedes consultar los libros "Extreme Programming Explained: Embrace Change" de Kent Beck (primera y segunda edición) y "Planning Extreme Programming" de Kent Beck y Martin Fowler (primera edición).

8.8.2 Scrum con Lean

Scrum y Lean tienen muchas coincidencias conceptuales. Para ambos, el componente humano es la piedra angular del éxito y la mejora continua una práctica fundamental. Ambos están centrados en el cliente y en entregar valor tempranamente. Más allá de esto, se puede aplicar Lean en Scrum si prestamos particular importancia a la calidad y a la reducción de

los desperdicios. Un desperdicio puede ser el inventario o código acumulado sin estar operativo de cara al usuario. También los tiempos de espera o colas en cuellos de botella, tanto en actividades de desarrollo como en dependencias con otros equipos. Las comunicaciones innecesarias o gastos en comunicación burocrática. El exceso de procedimientos y protocolos corporativos. Además de la cantidad de defectos que se buscan disminuir. Las actividades que no son de valor añadido. Básicamente, añadir Lean es facilitar el flujo de trabajo, limpiandolo de los desperdicios, además de respetar los demás principios Lean. Buscar mecanismos para amplificar el aprendizaje, creando y compartiendo conocimiento. Retrasar las decisiones fundamentales, tratando de tomarlas con la mayor cantidad de información posible. Entregar valor desarrollando soluciones con integridad conceptual y cohesionadas como producto (algo que parece lógico en ingeniería de software). Y, principalmente, siempre mantener una visión global, sin encapsularse en el equipo y en soluciones locales. Esto último es prácticamente el componente de pensamiento sistémico de Lean y que el equipo puede aplicar bajo Scrum. En todo caso, de nos ser Lean usado por el equipo, tranquilamente es el SM quien puede usarlo como parte de su facilitación del proceso de trabajo.

Si quieres saber más sobre Lean puedes consultar el libro "Implementing Lean Software Development: From Concept to Cash" de Mary Poppendieck y Tom Poppendieck.

8.8.3 Ampliar Scrum con Kanban

Scrum se puede complementar y mejorar agregando aspectos del método Kanban. Para eso se busca entregar valor con entregas tempranas y frecuentes, focalizando en hacerlo mediante un flujo de trabajo constante, estable y eficiente. Para lograrlo se siguen las siguientes prácticas:

1. Visualizar.
2. Limitar el trabajo en progreso.
3. Gestionar el flujo.

4. Hacer explícitas las políticas.
5. Implementar bucles de retroalimentación.
6. Mejorar colaborativamente, evolucionar experimentalmente.

Para mostrar el proceso hay que reemplazar en el tablero kanban los estados ambiguos o genéricos por concretos, como por ejemplo el estado “en progreso” (“on going” o “doing”) por todos los estados reales del flujo de trabajo en curso. Un ejemplo de un flujo completo (ver fig. 8.33) puede ser: nueva (new), refinanda (refining), lista (ready), comprometida (committed), desarrollando (developing), probando (testing), completa (done), aceptada (accepted), desplegando (deploy) y en operación (operative). Hecho esto, el equipo debe buscar mantener el foco de trabajo del sprint en un límite de cantidad de historias que se trabajan en paralelo (WIP). El equipo debe buscar descubrir cuál es el límite con el que pueden trabajar con cadencia sin peligrar el objetivo del sprint. Debe buscar que no hayan cuellos de botella en ninguno de estos estados o pasos del flujo. Y, finalmente, revisar y mejorar constantemente sus reglas de juego: DoR, DoD, acuerdos de trabajo, nomenclaturas de los tableros kanban y hasta el marco de trabajo mismo.

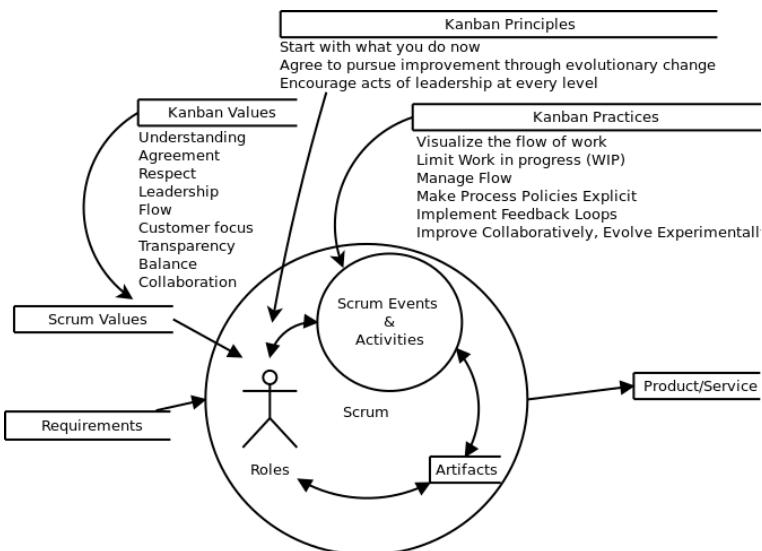


Figura 8.32: Scrum más Kanban

Esto es lo básico que se debería hacer para sumar el método Kanban a Scrum. Desde un punto de vista, prácticamente sistémico y Lean, un SM debe pensar a su equipo como si fuera un sistema a estabilizar y facilitar la maximización de su flujo de trabajo. El SM debería ser también una especie de “Flow Master”, un facilitador del flujo. Es así que también se pueden agregar el manejo de las métricas Kanban, como el cycle time y throughput (o delivery rate), y herramientas de monitoreo visual como el diagrama de flujo acumulado o CFD.

Si quieres saber más sobre Kanban puedes consultar el libro de referencia "Kanban Esencial Condensado" de David J Anderson.

Backlog			Sprint						Operation
New	Refining	Ready	Committed	Developing	Testing	Done	Accepted	Deploy	Operative

Figura 8.33: Tablero Scrum-Kanban ejemplo para un equipo de desarrollo

8.8.4 Ampliar Kanban con Scrum, Scrumban

Una metodología amiga de Scrum es Scrumban. Se trata de una pseudo-metodología mixta y flexible entre Scrum y el método Kanban. Combina las la flexibilidad de Kanban y las características básicas de Scrum. Por un lado, se toma de Kanban lo de mantener un trabajo continuo, exponer el flujo de trabajo, el tablero de trabajo persistente, auto-asignaciones de tareas solamente por el sistema del tomar y jalar (pull), limitar el trabajo en curso y buscar optimizar el flujo teniendo en cuenta la métrica “cycle time”. Por otro, se agrega de Scrum el evento planning, la retrospectiva como reunión de mejora continua (Kaizen), el trabajo en ciclos o iteraciones sin ser limitativo o restrictivo de trabajo y la priorización, que se recomienda hacer en cada planificación. Dentro de la flexibilidad que brinda, hace optativo el usar otras partes o prácticas de Scrum. Bajo esta metodología no es necesario estimar, puede ser algo optativo, ya que no se necesita calcular el trabajo que entra en una iteración porque el trabajo es continuo. Aunque sí se pueden estimar tiempos de entrega y/o tamaños de paquetes de trabajo. Por otro lado, al ser el trabajo continuo y no tener un compromiso formal de trabajo comprometido en una iteración, los cambios y re-priorizaciones pueden hacerse en cualquier momento.

También es muy flexible el tipo de roles de los miembros del equipo debido a que no prescribe roles específicos ni es necesaria la multidisciplinariedad que recomienda Scrum. Sin embargo

en los equipos Kanban o Scrumban han emergido las figuras de Product Owner y Flow Master (facilitador equivalente a un Scrum Master). En Scrumban el Flow Master es, además de un facilitador ágil, un facilitador del flujo del proceso. Usando la facilitación visual hay que tener en cuenta que, como se mencionó antes y al igual que en Kanban, el tablero permanece persistente, mientras que sólo cambian las tareas y sus prioridades. A diferencia de Scrum, donde el tablero de sprint se renueva en cada iteración. Relacionado a la planificación, en Scrumban se planifica focalizados en los releases y no es obligadamente necesario hacerse en la planning de cada iteración; pues, la planificación se puede realizar solo bajo demanda. Este método es muy útil principalmente para procesos de ritmo rápido, para startups, proyectos que requieren fabricación de productos constante, equipos y proyectos con ambientes muy dinámicos y cambiantes, equipos de solución de problemas de contingencias en producción, equipos de mantenimiento, soporte o desarrollo de infraestructura. También es útil para integrar (coordinar y sincronizar) otros equipos de soporte a los equipos Scrum, por ejemplo bajo el marco DevOps. En estos casos, los equipos soporte pueden usar Scrumban y los de desarrollo Scrum.

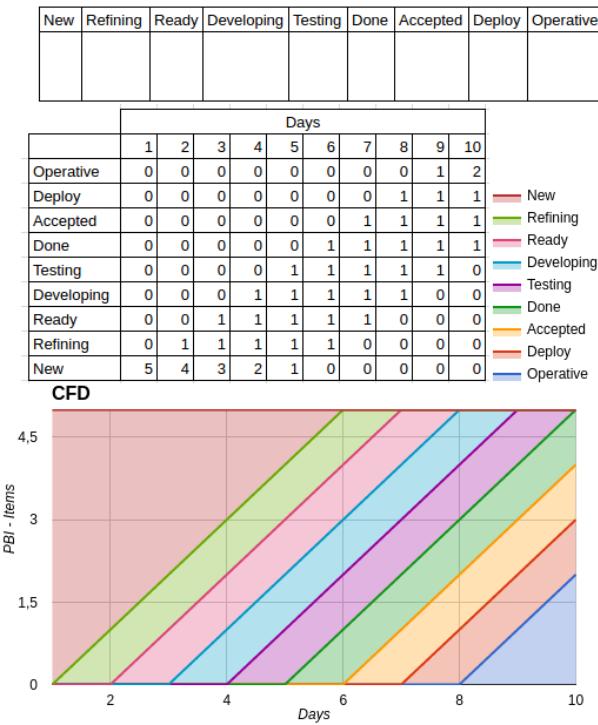


Figura 8.34: Tablero Scrumban y CFD ejemplo para un equipo de desarrollo

8.9 Checklists

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

8.9.1 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.).

8.9.2 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.).

8.9.3 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.).

8.9.4 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 9

Ingeniería de Software Ágil

9.1 Proceso de desarrollo

Hacer agilidad no nos asegura calidad. Para ello primero hay que asegurarse de hacer ingeniería de software además de agilidad. Hay diferentes procesos de ingeniería o ciclos de vida y podemos tomar el siguiente: descubrimiento y definición de requerimientos (Discovery & Requirements), análisis y diseño (Analysis & Design), implementación y pruebas (Implementation & Testing), integración y pruebas (Integration & Testing), despliegue (Delivery) y "operación y monitoreo" (Operation & Monitoring). En la última fase se contempla operaciones, monitoreo y el análisis de datos. Operaciones en cuanto a aseguramiento de la operatividad y rendimiento del software. El monitoreo de datos es el seguimiento a determinadas acciones que podemos cuantificar y que nos arrojaran datos relevantes para la estrategia (KPIs). Y el análisis de datos es la base de la optimización de la estrategia, que nos permite modificar, continuar o establecer nuevas directrices, sin perder de vista el objetivo final del software o producto.

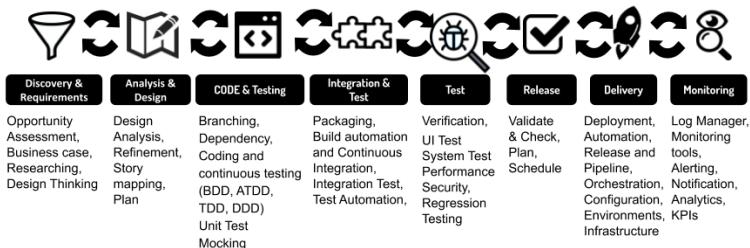


Figura 9.1: Ciclo de ingeniería de Software

Dicho esto, hay que considerar que el proceso de ingeniería de software ágil se hace sobre piezas de software (batch, feature o historia de usuario), y si bien es secuencial, tiene las fases del ciclo de vida que se solapan con las de otros piezas (ver fig. 9.1). Es decir que el software se va desarrollando de a fragmentos o pequeñas piezas de software, en forma incremental y evolutiva. Cada pieza de software se desarrolla con diferentes instancias del ciclos de vida.

Correr este proceso bajo un marco ágil como Scrum, con prácticas ágiles es lo que podemos llamar ingeniería de software ágil (Agile Software Engineering). Además es conveniente introducir la idea de prácticas continuas, es decir refinamiento continuo (análisis y diseño con trabajo UX continuo), testing continuo (continuous testing), integración continua (continuous integration), despliegue o entrega continua (continuous deployment & continuous delivery) y monitoreo continuo (continuous monitoring). Dicho esto, tenemos que tener en cuenta cómo unir y ajustar el proceso de ingeniería de software con el marco de trabajo Scrum. En este sentido es que se plantea ejecutar las fases del proceso de ingeniería, sobre historias de usuario, dentro del tren de sprints. En un equipo extremadamente autónomo y ágil, todas las fases se pueden desarrollar bajo el tren de sprints.

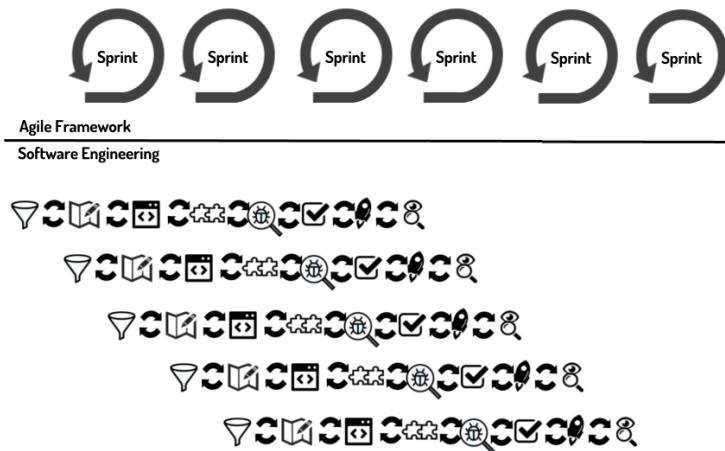


Figura 9.2: Ingeniería de Software Ágil

De este modo, se integra el testing en fases más tempranas del desarrollo, por lo que QA no retrasa al equipo de desarrollo, porque QA está dentro del equipo y se trabaja con testing continuo. Lo mismo sucede con UX y Operaciones.

9.2 Testing continuo

Si pensamos en Agile Testing pensamos en que queremos un testing dentro del equipo, con todo el equipo colaborando, que no sea solo una fase en un cascada, que reduzca el tiempo para recibir retroalimentación y asegure calidad suficiente. Cuando se incorpora agile testing, se introducen algunas prácticas, como por ejemplo Testing de “todo el equipo”, integración continua, testing guiado por pruebas (TDD), desarrollo guiado por pruebas de aceptación (ATDD), implementar la V de calidad usando historias, entre otros.

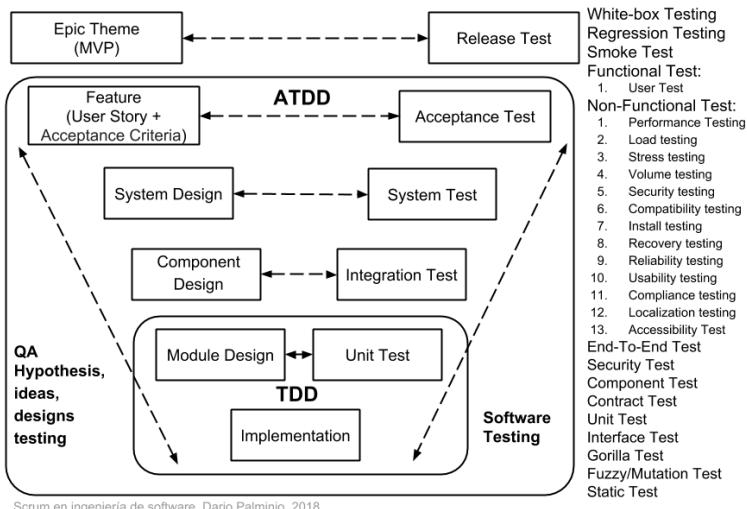


Figura 9.3: Modelo en V de calidad y testing

Aunque considero que lo más importante es incorporar la mentalidad de testing continuo, independientemente de las prácticas o técnicas particulares.

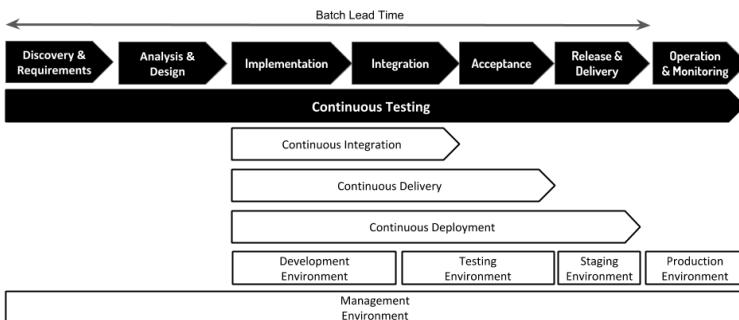


Figura 9.4: Ciclo de vida de una historia y prácticas

Cada fase del ciclo de vida del desarrollo puede contener prácticas de testing, desde las fases más tempranas hasta producción.

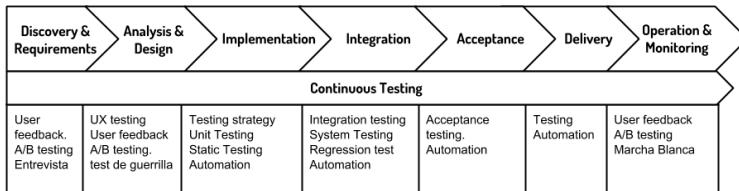


Figura 9.5: Prácticas básicas en continuous testing

Y, si lo pensamos bien, podemos decir que Scrum es una prueba de integración del sistema organizacional para validar continuamente si es posible que una organización entregue valor en un Sprint¹. La prueba de Scrum revela los fallos de la organización, disfuncionalidades o los impedimentos que luego debemos buscar solucionar para lograr la agilidad organizacional. De este modo, cada Sprint que hagamos será un ciclo de testing organizacional, tanto de nuestro equipo como del resto de la organización, para levantar acciones de mejora que podemos

¹Stacia Heimgartner Viscardi, Professional ScrumMaster's Handbook

explicitar, definir y evaluar en las sesiones de retrospectivas que tengamos y que apuntan a mejorar la propuesta de valor y el tiempo entre que se crea una historia y que la misma pueda ser validada por el cliente en forma operativa. Como dijo Jon Kern (co-autor del Manifiesto Ágil), “Ágil es reducir la brecha entre actuar y recibir feedback”, que en el desarrollo de productos se traduce a reducir el Time to Market (desde la perspectiva de negocio) o reducir el Delivery Lead Time (desde la perspectiva de DevOps); y Scrum nos ayuda a testear y medir esa brecha para reducirla. Con Scrum buscamos entregar software de calidad, que aporte valor, lo más pronto que sea posible.

9.2.1 Tamaños de Testing

Y hacia el equipo, el Scrum Master debe ayudarlo a que encuentre su manera de hacer buen testing y de mejorarlo. No existe realmente una forma única. Tampoco un convención de nombres y tipos de prueba claras. Depende de diferentes autores y de cada equipo. El Scrum Master debe ayudar con eso. Un ejemplo de esquema es el que han usado equipos en Google. A los desarrolladores de Google les gusta tomar decisiones basadas en datos, en lugar de confiar en el instinto o en algo que no se puede medir y evaluar. Ellos, llegaron a un acuerdo sobre un conjunto de convenciones de nomenclatura basadas en datos para sus pruebas. Llamaron a sus pruebas: "pequeñas", "medianas" y "grandes" ².

²Test Sizes by Simon Stewart, Google Testing Blog, Monday, December 13, 2010.

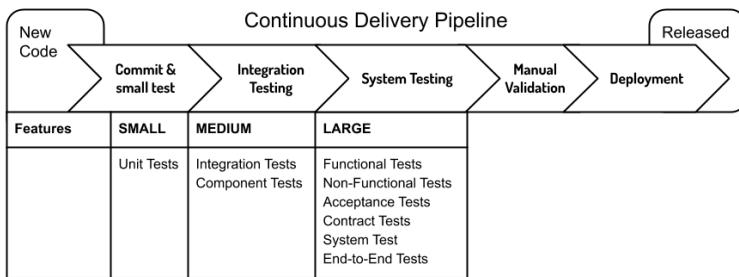


Figura 9.6: Tamaños de testing en el pipeline del Continuous Delivery

Van de la prueba pequeña que equivale a una prueba de unidad, pasan por la prueba mediana que asegura que los niveles en la arquitectura de una aplicación pueden comunicarse correctamente y por una prueba grande que es una de extremo a extremo o de sistema.

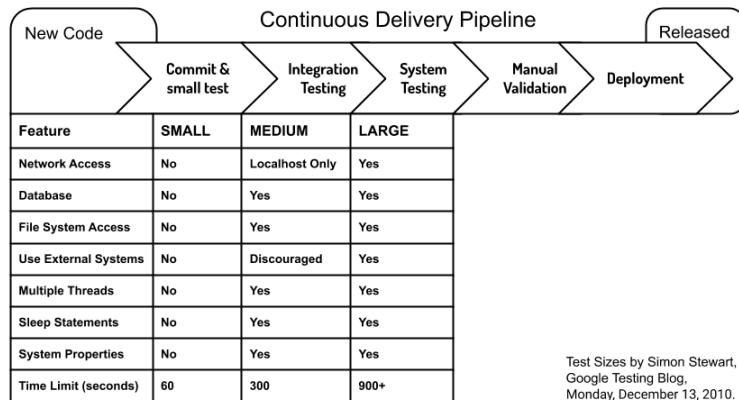


Figura 9.7: Tamaños de testing y tipos en el pipeline del Continuous Delivery

El Scrum Master debe asegurarse que el equipo encuentre la manera más efectiva y eficiente de hacer testing. Puede usar un

esquema como el anterior, la pirámide de testing de Mike Cohn, la de Lisa Crispin o lo que encuentren apropiado. Sin testing de calidad no hay software de calidad.

9.3 Prácticas técnicas

Como ya he mencionado, es deseable que el SM vele por la excelencia técnica de su equipo y del proceso de desarrollo, buscando lograr un flujo limpio, cadenciosos y sostenible. Claro que la responsabilidad es del equipo y tanto un líder técnico como el SM, pueden liderar o guiar al equipo en el proceso de mejora continua del flujo de trabajo. Teniendo en cuenta las fases de desarrollo de los ítems de trabajo como historias, podemos tener dos aspectos en consideración para analizar el estado de nuestro procesos de desarrollo en cuanto a excelencia técnica. Por un lado, el aspecto de prácticas técnicas considerando la información, proceso, métodos, técnicas y organización (aquí se tiene en cuenta XP, DevOps, etc.). Por otro lado el aspecto tecnológico considerando las herramientas empleadas que cubren estas prácticas técnicas. Por ejemplo a continuación muestro un esquema de flujo y sus prácticas técnicas relacionadas.

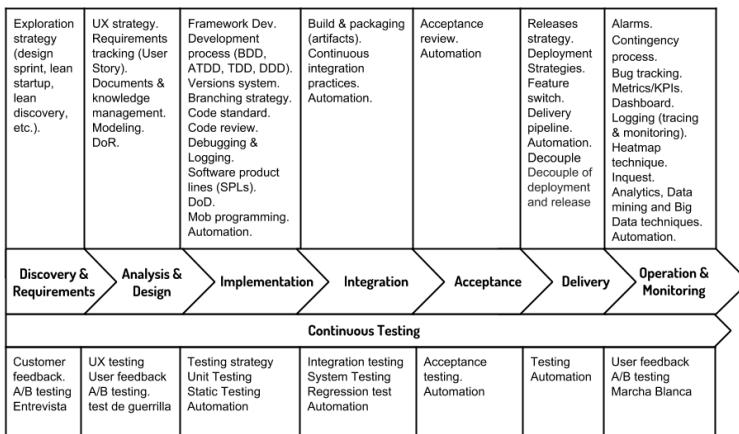


Figura 9.8: Flujo de prácticas técnicas

Se puede hacer un esquema de flujo (el pipeline de producto o de delivery) semejante con los aspectos tecnológicos, de herramientas, correspondientes a cada fase. Si bien hay un flujo genérico, cada equipo determinará cuál es su flujo y, en consecuencia, su pipeline.

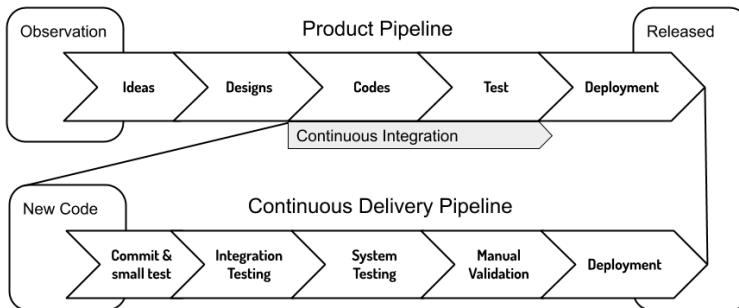


Figura 9.9: Flujo de producto vs. de delivery

Hay que tener en cuenta que para que el flujo de desarrollo de software fluya limpio y cadencioso, la experiencia del

desarrollo debe ser memorable y fluida sobre una plataforma e infraestructura que la facilite. Es por esto que un equipo de desarrollo tiene interdependencia con equipos de operaciones e infraestructura y es parte del desafío de un equipo de alto rendimiento disminuir la brecha, empoderarse y facilitar la integración de ese área de conocimiento.

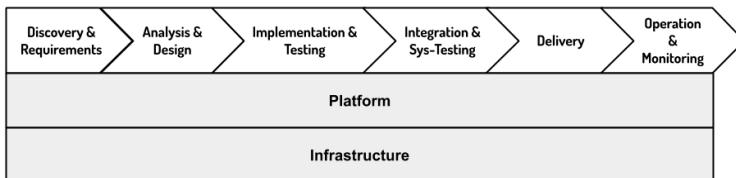


Figura 9.10: Plataforma e infraestructura

9.4 DevOps

Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software útil y el enfoque DevOps nos ayuda a este objetivo. Aclaro aquí que DevOps no es más que una práctica y enfoque dentro de la ingeniería de software. Por este motivo, una manera de deducir el nivel de ingeniería y, en consecuencia, de calidad de trabajo de un equipo es ver cómo se desenvuelven con DevOps en cuanto a cómo manejan la optimización y automatización del flujo de trabajo, la calidad y flexibilidad de su plataforma e infraestructura y como se integran o gestionan operaciones. A continuación veremos algunos temas de plataforma, infraestructura, pipeline, automatización y Feature toggles.

9.4.1 Plataforma

El trabajo de desarrollo de software es una actividad bastante compleja. Ya no se programa en texto plano y se integra enviando archivos por email. El trabajo del Scrum Master es ayudar a

su equipo a que encuentre su stack tecnológico adecuado para el problema que intenta resolver. La complejidad de plataforma justa y mínima que facilite y optimice el flujo de desarrollo. Se debería preguntar: ¿Qué plataforma tiene y debería tener el equipo? ¿Cómo maneja o debería manejar la plataforma? ¿Cuáles son sus herramientas informáticas para idear, crear, integrar, probar, desplegar, monitorear y corregir errores de software de una manera ágil? ¿Cuán rápido permiten llevar una idea a producción?

Por ejemplo, un equipo estándar que desarrolla algún sitio web con una arquitectura orientada a microservicios. ¿Qué necesitaría? A modo de ejemplo podemos pensar en lo siguiente:

- **Análisis y diseño:** El equipo usará, en una de sus primeras fases, herramientas de gestión de proyectos (como Jira de Atlassian, Trello, etc.) para idear, definir, compartir, coordinar y planificar trabajo en forma de historias de usuario, tickets o docs. El equipo necesita comunicarse para lo que usará software de chatting (email, Slack, Rocket.Chat, Fleep, etc.) y conferencias (Google Hangouts, Zoom, skype, etc.). En el proceso de análisis y diseño necesitará compartir documentación y otros archivos necesitando un repositorio de documentos compartidos (Google Drive, Dropbox, OneDrive y SharePoint en Office 36 de Microsoft, etc.). Sumando herramientas de diseño UX (por ejemplo para Wireframes, Visual Design, etc.) y de diseño y diagramado de software (Día, Visio, Google Draw.io, etc.).
- **Implementación:** En programación incluirán lenguajes, entornos IDE de programación (Eclipse, etc.) y librerías. Necesitará un gestor de configuración y versionado de software o Version Control System para trabajar en un código de propiedad compartida (github, Subversion, CVS, etc.).
- **Integración:** Trabajará con un esquema de branching e integración de código para lo cual usará herramientas de compilación, dependencias y building, integración y despliegue (como Gitlab, Maven, etc.) además de

herramientas de automatización de integración y despliegue (como Jenkins, Travis CI, Buildbot, DotCi, etc.).

- **Pruebas:** Junto a esto, para realmente hacer ingeniería, necesita herramientas de gestión, ejecución y automatización de pruebas (como Selenium, Sahi, Source Labs, Testing Bot etc.).
- **Despliegue:** Herramientas para desplegar en producción (Jenkins, Travis CI, Buildbot, DotCi, Ansible, Puppet, Docker, etc.).
- **Monitoreo:** Y, en su etapa de monitorear el software operativo en producción, necesita monitorear el rendimiento y la salud del software y atender las contingencias que ocurran. Aquí necesitará herramientas de Bug-tracking como comunicación y sistema de ticket para identificaciones de errores y bugs (Bugzilla, Mantis Bug Tracker, etc.), herramienta de logs o registros y monitoreo (Splunk, Sumo Logic, Logstash, GrayLog, Loggly, PaperTrails, etc.). Y por último el equipo necesita evaluar los datos del negocio que opera el software, tendencias, comportamientos de los usuarios y clientes, para lo que puede usar herramientas de analítica digital y Business Intelligence (Splunk, Google Analytic, Elasticsearch y Kibana, etc.).

Luego, todas estas herramientas... ¿Sobre qué corren? ¿Sobre qué infraestructura funciona? ¿Qué ambientes de programación, pruebas y producción manejan? ¿Cuán versátil y fácil de configurar es? ¿Qué dominio, propiedad y responsabilidad tiene el equipo sobre esta? ¿Cómo se comunica y se coordina con los proveedores y el soporte?

9.4.2 Infraestructura

Ambientes de software

El flujo de desarrollo de software sucede y fluye por diferentes ambientes de software, que podrían clasificarse como ambiente de

desarrollo local, de desarrollo compartido (o de integración), de prueba, de preparación (Staging) y de producción.

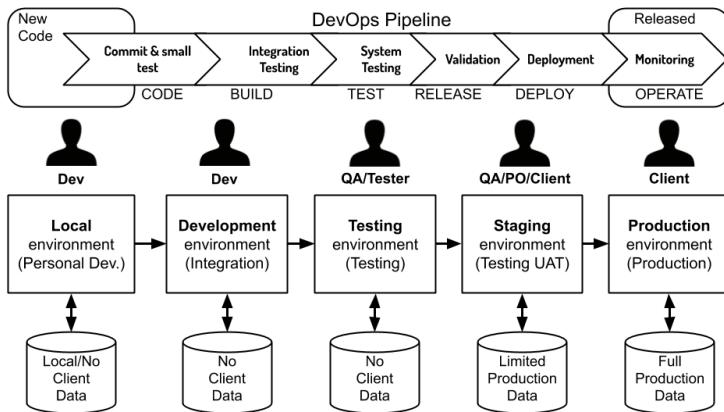


Figura 9.11: Ambientes del flujo de desarrollo

- **Desarrollo:** Un ambiente de desarrollo compartido e integración está configurado para permitir a los desarrolladores escribir código rápidamente, verificarlo creando pruebas básicas (pruebas unitarias) y ser productivo. Este entorno es mucho más pequeño de lo que se necesita para ejecutar una aplicación completa en una implementación de la vida real. También presenta herramientas específicas para desarrolladores que a veces pueden obstaculizar la validación rigurosa del control de calidad. Y lo más importante, el entorno de desarrollo cambia constantemente, con nuevas funciones que se agregan todo el tiempo, lo que dificulta que los ingenieros de control de calidad realicen pruebas que requieren mucho tiempo, pruebas de regresión o integración, sin interrumpir el proceso de desarrollo. Debido a que ejecutar pruebas complejas en el entorno de desarrollo conduciría a una gran pérdida de tiempo de los desarrolladores y, a su vez, por la poca estabilidad del software dificultaría el trabajo de los testers,

es que se hace necesario otro entorno especializado para probar.

- **Prueba:** El ambiente de prueba es donde los ingenieros de control de calidad pueden usar una variedad de herramientas de prueba para ejecutar todas sus diferentes pruebas sobre el código de aplicación tomado del entorno de desarrollo. Mientras los desarrolladores comprueban su código en busca de errores simples antes de pasarlo para garantizar la calidad, los testers ejecutan tipos de pruebas más complejas y que requieren mucho tiempo para verificar la compatibilidad del código nuevo y antiguo, la integración correcta de los diferentes módulos, el rendimiento del sistema, etc.
- **Staging:** Luego se necesitan pruebas de aceptación del usuario en un ambiente Staging, provisional, antes de subir a producción. El Staging es un entorno de ensayo pre-productivo y es una réplica idéntica del entorno de producción del cliente, que también suele contener datos de producción reales que se han desinfectado por motivos de seguridad. Está alojado de la misma manera que los servidores de producción e implica una configuración idéntica y operaciones de actualización. Por lo tanto, las pruebas en un entorno provisional ofrecen la forma más confiable de verificar la calidad del código y garantizar que los servidores de producción tengan éxito.
- **Producción:** Y por último el ambiente productivo donde se explotará el software. Aquí estarán todos los datos reales del cliente y es donde el software, de algún modo, vive.

Infraestructura ágil

La infraestructura es como la autopista por donde circula nuestro código dentro de los ambientes como vehículos. En consecuencia la calidad y tecnología de esta autopista determina la velocidad con que circulará nuestro código y la tasa de accidentes. La infraestructura ha evolucionado y seguirá haciéndolo a un ritmo

acelerado. A continuación cuento una visión general de cómo se ha dado.

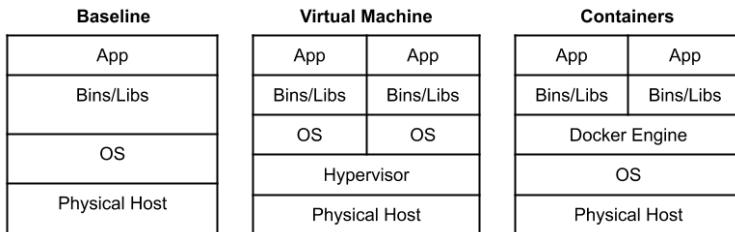


Figura 9.12: Infraestructura

- **Baseline:** La forma más simple de infraestructura de trabajo que tenemos como desarrolladores es el Baseline, en donde el desarrollador instala todo lo necesario para programar y correr la aplicación desarrollada en su máquina física local. Y, además, puede usar servidores dedicados o Datacenters para hacer pruebas y despliegues. El problema de esta estrategia es que cuando estamos desarrollando una aplicación inestable puede desestabilizar todo el sistema, desconfigurar el computador y terminar todo en un caos provocado por la coexistencia de varios lenguajes o plataformas, librerías, servidores de bases de datos locales, espacios de trabajo de nuestro IDE, proyectos de versionado de código, etc... Y esto nos hace perder mucho tiempo en estabilizar todo de nuevo. Y en el caso de servidores compartidos esto se potencia, porque son más personas las que echan manos. Esto no es para nada ágil. Para mejorar esto surgió la posibilidad de trabajar con virtualización.
- **Virtualización:** La virtualización nos permite ejecutar aplicaciones en un entorno controlado, con lo necesario para funcionar, encapsulado en un sistema operativo virtual independientemente de la máquina real que use. La virtualización funciona por medio de la abstracción de recursos, esto es por parte de un Hypervisor o VMM

que corre en la máquina física del desarrollador. Una de las principales ventajas que tiene la virtualización es la capacidad de separar un entorno inestable (el virtual) de otro estable (la máquina física). Esto permite restaurar el sistema a cualquier estado previo correcto en menos tiempo que si se trabajara directamente en la máquina física real como el caso Baseline. Otra ventaja es poder trabajar con múltiples entornos de prueba. Por ejemplo, en la misma máquina podemos correr la aplicación en distintos sistemas operativos. Esto mejora también el trabajo sobre servidores compartidos. Diferentes herramientas nos ofrecen virtualización de sistemas operativos (Docker, Virtuozzo, OpenVZ, Linux-VServer, etc.). Además existe la virtualización de procesos (Java Virtual Machine, etc.) y de sistemas (Oracle VM VirtualBox, MTL Virtual Machine, etc.). Junto a estas tecnologías surge la virtualización de red que nos ayuda a desacoplar completamente los recursos de red del hardware subyacente. El problema de la virtualización es que el proceso de configuración y distribución sigue siendo una tarea manual, repetitiva y en definitiva, poco conveniente. Esta forma no tiene tanta flexibilidad y portabilidad. Además, cuando necesitamos tener máquinas virtuales dedicadas y tenemos un elevado número de servidores en una misma máquina, se ve una clara reducción de recursos. Por eso surgió otra estrategia de virtualización más avanzada, la ‘containerización’.

- **Containerización:** Los contenedores nos proporcionan un entorno aislado e independiente como forma de empaquetar todo lo necesario para ejecutar una aplicación: el código, las herramientas del sistema, librerías, etcétera. Sin usar sistemas operativos virtuales completos. Esto permite la portabilidad, que implica que los desarrolladores puedan crear aplicaciones en una computadora portátil y desplegarlas en los servidores, de forma más rápido, arrancarlas y pararlas más rápido y aprovechar mejor los recursos de hardware. Además facilita trabajar en una arquitectura de microservicios. Existen diferentes

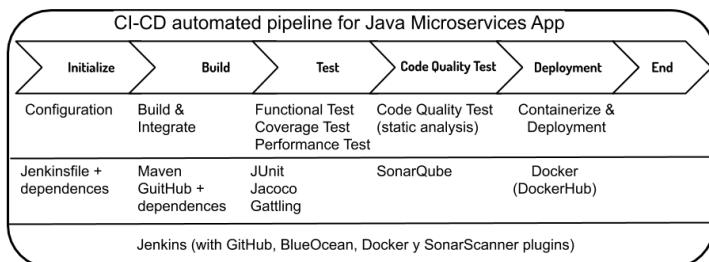
herramientas software que nos provee containerización (Kubernetes, Docker, Solaris Containers, etc.).

- **Infraestructura como código:** Por otra parte, surge la “infraestructura como código”, en alternativa a la infraestructura tradicional que traía procesos manuales, errores humanos, demoras en configuraciones, cuellos de botella por entrega de la infraestructura, documentación incompleta, dificultad de introducir frecuentes cambios, etcétera. Con ella aplicamos herramientas y prácticas de ingeniería de software en nuestra infraestructura. Esto nos permite gestionar y proporcionar infraestructura de forma programática a través de código y automatización, para crear y cambiar servidores, instancias, entornos, contenedores o alguna otra infraestructura de forma ágil. En otras palabras, escribimos y ejecutamos código con diferentes herramientas software (Chef, Ansible de Redhat, Puppet, SaltStack, Terraform de HashiCorp, Vagrant, etc.) que define, despliega y actualiza infraestructuras.
- **Cloud Computing:** Y todo esto nos lleva al Cloud Computing, o computación en la nube. Que es una plataforma que permite ofrecer las TI como servicios en la red. Todo lo que se encuentra en la nube Cloud se ofrece al usuario o programador como servicio, tanto software (SaaS), plataforma (PaaS) e infraestructuras (IaaS). El Cloud está basado en entornos virtualizados de alta disponibilidad y rendimiento sobre la infraestructura de Datacenter del proveedor. Además existe Elastic Computing que se aplica en los Elastic Cloud como tecnología de adaptación de uso de recursos en función de la demanda. Actualmente estas tecnologías son provista por proveedores dedicados al negocio de Clouding (Amazon, Azure, Google, Alibaba, etc.).

9.4.3 Pipeline automatizado

Desde el aspecto técnico, en ingeniería de software buscamos reducir el tiempo entre el inicio del desarrollo de una funcionalidad

y su puesta en operaciones (lead time), la mejora continua de nuestro proceso de desarrollo de software y lograr código de calidad. Es decir acelerar el time-to-market, ser eficientes y entregar software de calidad que sea valioso para el cliente o el usuario. Y DevOps es parte de esta disciplina que se centra en eficientizar todo el proceso desde el desarrollo hasta producción. En este sentido, podemos decir que el santo grial de DevOps es la automatización completa de todo el proceso centrandonos, de principio, desde que el desarrollador hace commit (push del código) hasta que su código llega a producción (Delivery Lead Time). Para automatizar este proceso o flujo de despliegue, se necesita programar la generación de código, integración, empaquetados, testing automáticos, orquestación, empaquetado y/o containerización y despliegue. Una manera de centralizar todo el flujo es en un “pipeline” y lo hacemos mediante la programación de configuraciones como scripts (pipeline as code) usando algún “automation server” como Jenkins.



Bases for Code: Java & JDK (JVM), VCS (Git), Web Application Framework (Angular or React, jHipster), JavaScript Environment (Node.js), JavaScript Packaging (Yarn), Database (PostgreSQL, MongoDB).
 Code Generation & Dependency Manager: Build Automation Tool for Java (Maven) y Package Manager for JavaScript (Npm)
 Monitoring: Suite ELK (Elasticsearch, Logstash y Kibana)
 S.O.: Red Hat Enterprise Linux
 Servers + Cloud: Central Configuration (Spring Cloud Config), Discovery (Netflix Eureka), Load Balancer (Netflix Ribbon), Circuit Breaker (Netflix Hystrix), Edge Server (Netflix Zuul), Authorization Server (Spring Security)

Figura 9.13: Ejemplo de un CI-CD pipeline para una App Java con arquitectura de Microservicios

Un pipeline es una secuencia de eventos o jobs que se pueden configurar y ejecutar como una secuencia de etapas STAGE. Donde cada STAGE es un conjunto de pasos Steps. Y cada Steps

es una tarea que dice qué hacer mediante un bloque de líneas de código script.

```
pipeline { //Job pipeline
    stages {
        stage("Initialize") {
            steps {
                echo "Initializing pipeline"
                // Here Initializing code...
            }
        }
        stage("Build") {
            steps {
                echo "Building!"
                // Here Building code...
            }
        }
        stage("Test") {
            steps{
                echo "Testing!"
                // Here code to throw Test...
            }
        }
    }
    stage("Code Quality Test") {
        steps{
            echo "Code Quality analyzing..."
            // Here code to throw static analysis...
        }
    }
    stage("Deployment") {
        steps{
            echo "Deploying..."
            // Here code to throw Deployment code...
        }
    }
} // End of stages
} // End of pipeline
```

Figura 9.14: Script ejemplo de un pipeline

Cada pipeline que el equipo cree dependerá de su arquitectura de software, plataforma y la propia madurez de ingeniería de software del equipo. Y la arquitectura de software o la plataforma no es excusa para no hacer DevOps ni tener en cuenta un pipeline. Uses Java, Microsoft Visual Studio o SAP tu puedes hacer DevOps. Tal vez un equipo más inmaduro en su desarrollo o al inicio de su proyecto tenga un pipeline pobre o no lo tenga. Y un equipo más desarrollado o avanzado tenga un pipeline más sofisticado (aunque simple de usar), rápido y automatizado. Tu pipeline demuestra tu nivel de DevOps y tu nivel de DevOps demuestra tu nivel de Ingeniería de Software. En particular recomiendo un pipeline basado en pruebas pequeñas (pruebas unitarias), medianas (pruebas de integración) y grandes (pruebas de sistema) y evolucionarlo de algo simple a algo más sofisticado y completamente automatizado.

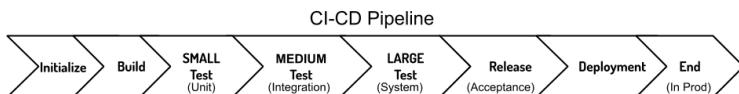


Figura 9.15: Esquema de un CI-CD pipeline tipo small-medium-large test

9.4.4 Activadores de características

Algo contemplado en DevOps es la posibilidad de hacer rollback rápido o desactivar funcionalidades de forma simple y rápida. Algo que ayuda a esto y a desplegar rápidamente pequeñas funcionalidades es una técnica introducida por de "Feature Toggles" (feature switch o feature flag). Esta permite a los equipos modificar el comportamiento del sistema de software sin cambiar el código. Es como tener llaves de encendido de características de nuestra aplicación que corre en producción. Los Feature toggles se implementan con Feature Flags (vanderas de características). Y tenemos tres opciones básicas para implementar:

- **Static Feature Flag:** Esta es la manera más simple, pero menos recomendada, y consiste en usar variables banderas directamente en el código fuente y productivo como "toggle point" para habilitar o deshabilitar funcionalidades.
- **Dynamic Feature Flag:** Otra manera es activar o desactivar flags de una manera dinámica fuera del código fuente core desde un subsistema que centraliza la configuración de flags y que se llama Toggle Configuration. Los Toggle Configuration pueden ser archivos de configuración o config file (flags en archivos) o bases de datos (flags en tablas) con algún software de gestión de Feature Flags. Las aplicaciones tendrán características con Toggle Points que usan un Toggle Router para decidir según un Toggle Configuration externo. El Toggle Configuration y Toggle Router pueden evolucionar a subsistemas de administración más complejos, con interfaces de usuario y sistemas de autenticación.

- **Context Flag:** Otro mecanismo es el que permite desactivar ciertas funciones para nuestra base general de usuarios en producción, pero poder activarla para usuarios internos. Esto se puede implementar usando configuradores de ambientes con Toggle Context. El Toggle Router leerá el ambiente de este Toggle Context y el Toggle Point encenderá o apagará una característica según el contexto y el valor del Toggle Configuration.
- **Canary Releasing:** Esta técnica es una manera de incluir a la anterior. Consiste en permitir ir activando funcionalidades por zonas. Primero las activaríamos en sistemas internos, después en sistemas de clientes con confianza y si todo funciona correcto al final se activaría a todos o los clientes críticos. Si en alguna de estas zonas hay problemas, dejaríamos de avanzar y reestableceríamos el sistema para luego resolver el problema.

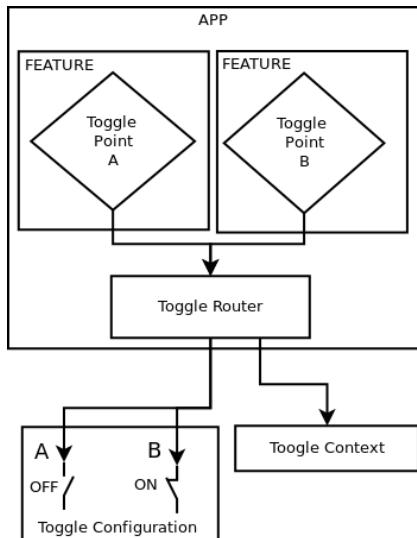


Figura 9.16: Esquema de feature toggle

Los Feature Flags ayudan a desplegar pequeños experimentos

en producción para recibir feedback rápido, recuperar la estabilidad de la aplicación ante fallos fácilmente sin tener que desinstalar (sin hacer rollback), encender releases planificados desplegados apagados con anterioridad (solo encendiendo flags), hacer pruebas por zonas de alcance de usuarios. Los Feature Flags impactan a las métricas de Deployment Frequency y MTTR mejorando la prestancia DevOps. Aunque no se debería abusar del uso de esta técnica. Se recomienda trabajar primero buscando desarrollar features lo más chicas posibles para hacer despliegues continuos de pequeñas funcionalidades. Y usar Feature Toggles solo cuando el equipo considere adecuado.

9.5 En síntesis

Hacer ingeniería de software ágil es desarrollar software de valor, de modo iterativo, incremental y evolutivo, con un producto o servicio que evoluciona, donde los requisitos y soluciones cambian con el tiempo, haciendo entregas frecuentes de calidad y valor, testeando continuamente, con feedback rápido, trabajando con equipos auto-organizados y multidisciplinarios que mejoran, inmersos en un proceso compartido de toma de decisiones interactiva y dinámica y entendiendo el negocio. Y, lo más importante, es considerar que la ingeniería es, además de una práctica científica, una práctica humana. Construir software se trata de relaciones humanas en un sistema social. Por eso, hay que construir relaciones humanas de valor, que habiliten la entrega continua de valor. Según Patrick Lencioni, un equipo humano debería tener ciertas aptitudes funcionales que posibilitan el trabajo de alto rendimiento: confianza, sin miedo al conflicto, compromiso, asunción de responsabilidades y atención a los resultados. Según el proyecto Aristóteles de Google, la clave es la cultura del equipo y lo necesario en ella es la seguridad psicológica entendida como: el sentimiento de confianza de que el equipo no va a avergonzar, rechazar o castigar a alguien por sus opiniones, comportamientos o ideas. Y como un equipo no es una isla, la organización que le da contexto debe reforzar este tipo de cultura, el factor humano, para lograr personas extraordinarias

desarrollando resultados extraordinarios.

Capítulo 10

Malas y buenas prácticas

10.1 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 Maeda¹). El problema no es que no sirva o no funcione, sino

¹Masa 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

que no lo hacemos bien (Masa Maeda 2012). Pues, en el artículo original de 1970 en el que Royce Winston expone el 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 hibridaciones² 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 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

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.

²Metodología híbrida o mezcla con Scrum.

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 inefficiente. No 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 éxito³.

Desde esta perspectiva, hay que tener en cuenta que el

³[Victor Hugo Certuche, 2016] [Mike Hadlow, 2014]

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, 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 profesionales⁴ 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

⁴[SWEBOkV3, 2014]

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 la manera de trabajar y se aplica un “Scrum cosmético”⁵. Esto quiere decir que se agregan los eventos y 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 el enfoque 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

⁵Un modelo de madurez para equipos ágiles, Angel Medinilla, Jan 15 2015.

entrega continua⁶. Operaciones pueden generar bloqueos y cuellos de botella debido a: procesos pesados que 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 minutos, como si se tratara de un "vicio organizacional"⁷. Por otro lado, en las organizaciones donde prima la confianza no es necesario asentar toda reunión en minutos. Las minutos 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 antípatrón que se ha arrastrado por años y que se consideraba como la forma de trabajo eficiente. 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

⁶Disociación 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].

⁷[UNTREF, 2014]

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 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.

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 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 puede 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 organizaciones y sistemas o que tiene la solución al problema o cambio organizacional que estamos necesitando. A veces sucede que los entrenadores ágiles son defensores de lo que saben (Scrum, Lean, Kanban, SAFe, LeSS, Crystall 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 y tratar según su coyuntura. Ninguna metodología resuelve todos los problemas y es necesario integrar diferentes, según el contexto y estado organizacional, según la propia experiencia organizacional, con herramientas de ingeniería de sistemas y enfoques multidisciplinarios.

13. No hacer ingeniería de sistemas cuando se quiere escalar Scrum

Si se deja librado a la suerte el cambio organizacional en escalamiento de Scrum, sin la guía de algún equipo experto en sistemas y agilidad que facilite hacer ingeniería de sistemas en la estructura organizacional, entonces la probabilidad de perder grandes esfuerzos, costos y tiempos por tomar malas decisiones aumenta. A veces el cambio organizacional se deja librado a la pura autodeterminación plena de los equipos empoderados, de abajo hacia arriba, y con el acompañamiento de líderes que no tienen el suficiente conocimiento de agilidad y de sistemas. También se puede dar que se caiga en la mala práctica de “se hace lo que el mesías dice” y resulta que el mesías no sabe hacer ingeniería de sistemas y no tiene el suficiente conocimiento de los modelos diferentes de escalamiento. Existen diferentes

factores que pueden impedir que se madure una evolución organizacional ágil orgánica y sistémica. Los cambios de procesos, sistemas, estructuras organizacionales y roles de trabajo, sin el empleo del pensamiento sistémico en ingeniería, pueden hacer que los cambios no sean evolutivos y las transformaciones sean lentas, caóticas o, en el peor de los casos, fallidas.

10.2 Recomendaciones

Hay recomendaciones que se pueden hacer para que tanto scrum u otra metodología ágil sean exitosas como por ejemplo no incurrir en las malas prácticas, automatizar todo lo posible, tener acuerdos de trabajo consensuados, trabajar en equipo y juntos, generar visión compartida, etcétera. Sin embargo voy a comentar algunos pocos consejos para llevar Scrum con éxito en una organización.

1. **Patrocinio ejecutivo:** según el onceavo reporte de estado de Agile de VersionOne, las compañías que logran implementar agilidad con éxito necesitan patrocinio ejecutivo (48 %).
2. **Soporte de expertos:** según el onceavo reporte de estado de Agile de VersionOne, las compañías que logran implementar agilidad con éxito necesitan Agile Coaches internos (el 52 %) y capacitadores o consultores ágiles (el 36 %).
3. **Capacitaciones:** es importante que todos los roles Scrum reciban capacitación además de mantener alguna frecuencia en las capacitaciones y workshops internos. Lo que se hace es asegurar que todos usan el mismo vocabulario, usan los términos adecuados y se encuentran más o menos en el mismo entendimiento de qué es ser ágil y cómo llevar adelante las prácticas ágiles con Scrum.
4. **Espacio dedicado al trabajo ágil:** es aconsejable diseñar espacios dedicados al trabajo ágil formado por espacios cómodos de trabajo, puestos de trabajo funcionales y donde

todos los integrantes del equipo se puedan ver e interactuar fácilmente, espacios para reunirse con pizarras, espacios para colocar afiches y pegar notas o post-its, etcétera.

5. **Trabajo sobre la cultura:** No se debe llevar un cambio organizacional sin trabajar en la cultura. Para esto los valores son clave. Se debe buscar que cada uno de los miembros de la organización pongan en práctica una serie de valores y principios (playbook), simples y claros, que habiliten el cambio y el nuevo estado deseado en la organización.

Capítulo 11

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 usándolo, mi participación en transformaciones organizacionales hacia la agilidad y de un diálogo constante con compañeros profesionales.

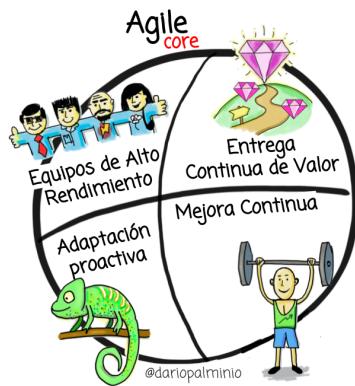
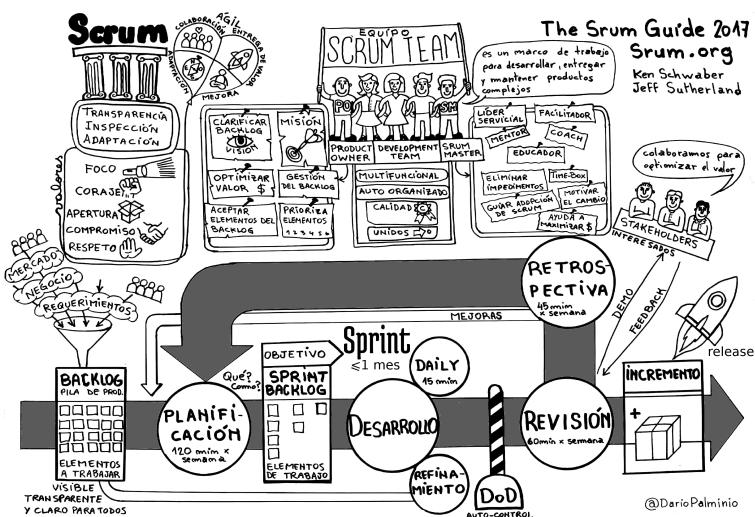


Figura 11.1: Agile Core

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 en un sistema de trabajo fluyente. La habilidad de un PO se muestra en la búsqueda por lograr, con su equipo, el mejor producto posible para el negocio, satisfaciendo al cliente, y en forma temprana. La excelencia en los desarrolladores es entregar iterativamente, en forma temprana y frecuente, el producto de mejor calidad posible y adaptándose a las necesidades del cliente. La magia de Scrum consiste en no acostumbrarse ni conformarse y siempre avanzar mejorando, en un estado de flujo, con armonía y calidez humana. Te ayuda a madurar un equipo de alto rendimiento, a entregar valor en forma continua, adaptarse proactivamente y mejorar continuamente. O sea, a ser ágil. Haz tu propia experiencia fluyendo con Scrum.



Capítulo 12

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. A nivel mental representa un enfoque y a nivel social un movimiento.

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.

ATDD: Acceptance Test Driven Development es un enfoque por el cual las pruebas de aceptación se hacen antes de desarrollar, en forma de escenarios y se automatizan.

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.

BDD: es un enfoque que busca un lenguaje común para unir la parte técnica y la de negocio, y que sea desde ese lenguaje común desde donde arranque el Testing. BDD es como el puente para unir un ATDD con un TDD.

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

Business Owner: Responsable de los resultados comerciales y técnicos del producto o servicio.

Business Sponsor: Es quien financia y participa activamente del desarrollo del producto o servicio.

Cycle time: Tiempo de ciclo es cuando el trabajo real comienza hasta cuándo está listo para entregarse. Tiempo en que dura una fase de trabajo incluyendo su cola de espera. O sea que es la suma del “touch time” o tiempo de trabajo real y el tiempo de cola. Suele ser una columna compuesta en un tablero Kanban.

DDD: El diseño guiado por el dominio (domain-driven design), es un enfoque que pone el foco primario del proyecto en el

núcleo y la lógica del dominio basando la programación y los diseños complejos en el modelado del dominio del problema.

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).

Ingeniería de Sistemas: Actividad profesional con un enfoque científico-técnico, interdisciplinario, organizacional y de pensamiento sistemático para la invención y utilización de técnicas dirigidas a la construcción, operación y mantenimiento de sistemas, incluyendo sistemas sociales empresariales, cibernéticos y sistemas hombre-máquina. Los lineamientos generales de esta disciplina pueden ser encontrados en SEBoK (Guide to the Systems Engineering Body of Knowledge).

Ingeniería de Software: Actividad profesional con enfoque científico-técnico, sistemático, disciplinado y cuantificable para la invención y utilización de técnicas dirigidas a la construcción, operación y mantenimiento de software, de aplicaciones informáticas o a la automatización de información en sistemas hombre-máquina. Los lineamientos

generales de esta disciplina pueden ser encontrados en SWEBOK (Guide to the Software Engineering Body of Knowledge).

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.

Flow Master: es también llamado Service Delivery Manager, Flow Manager o Delivery Manager y es un facilitador del flujo en un equipo Scrumban o Kanban.

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.

KPI Indicador clave de rendimiento (Key Performance Indicators).

Métricas Las métricas son una combinación de atributos cuantificables pertinentes que comunican información relevante acerca de la calidad de nuestros productos y nuestra productividad¹. 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 indirectas². 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

¹[INCOSE, 2005]

²[Roger S. Pressman, 2002]

es tan poderoso en ciencia, en ingeniería de software como en cualquier disciplina.

MMP: Producto comercial 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), código de software funcional entregado o software en funcionamiento.

PMI: Instituto de gestión de proyectos (Project Management Institute).

PMO: Oficina de gestión de proyectos (Project Management Office).

PSPI: Incremento de producto potencialmente entregable (Potentially Shippable Product Increment), código de software funcional entregable o software en funcionamiento listo para entregar.

QA: Aseguramiento de calidad (Quality Assurance) o 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).

OPCON: contingencia operacional o fallo de software en operaciones (Operational Contingency).

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.

SME: Experto en alguna materia o disciplina (Subject Matter Expert).

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

Spt: Iteración (Sprint).

SP: Story Point o puntos de historia.

Throughput: Número medio de unidades procesadas en un tiempo determinado. Tasa de salida del sistema de trabajo.

UI: Se refiere al diseño de interfaz de usuario (user interface) o al rol encargado de analizar y diseñar interfaz de usuario.

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

VoC: Voz del cliente.

VUCA: El entorno VUCA, se caracteriza por la volatilidad, la incertidumbre, la complejidad y la ambigüedad (Volatility, Uncertainty, Complexity and Ambiguity).

WIP: Trabajo en curso (Work In Process). Cantidad de ítems de trabajo desarrollándose en paralelo en una fase de desarrollo o en una columna de tablero Kanban.

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, 2001, como estaba en Octubre de 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.
- [Gantthead-James, 2010] Gantthead-James (2010). *Seven Obstacles to Enterprise Agility. By Gantthead, James.* gantthead.com.
- [Greg Gehrlich, 2012] Greg Gehrlich (2012). *Build It Like A Startup: Lean Product Innovation by Greg Gehrlich.* 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.
- [Jez Humble, 2018] Jez Humble (2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* by Jez Humble. 2018.

[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. Wwww.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.

[Ken/Jeff, 2017] Ken/Jeff (2017). *La Guía de Scrum. La Guía Definitiva de Scrum: Las Reglas del Juego.* por Ken Schwaber y Jeff Sutherland. Scrum.org.

[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, apredizaje, 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).

[Paulo Caroli, 2017] Paulo Caroli (2017). *Directo al Punto: Una guía para la creación de productos Lean.* 2017.

[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 VMEdú, 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).* Scrum-Alliance.
- [Scrum-Institute, 2015] Scrum-Institute (2015). *Scrum revealed: the only book can simply learn scrum! by International Scrum Institute.* 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 Emergence.* By Steven Johnson. Prentice Hall 2002.
- [SWEBOKEv3, 2014] SWEBOKEv3 (2014). *SWEBOKE Guide V3.0, Guide to the Software Engineering Body of Knowledge Version 3.0.* Editors Pierre Bourque, École de technologie supérieure (ETS) 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

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>