Introducción
a la Agilidad y Scrum



kleer™
Agile Coaching & Training.

Web | www.kleer.la
Facebook | facebook.com/kleer.la
Twitter | twitter.com/kleer.la

CC BY NC ND

Índice

Introducción 1
a la Agilidad y Scrum 1
Esta obra fue realizada por Martín Alaimo para KLEER, con aportes de Pablo Tortorella y Daniela Casquero y se encuentra bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported 2
Introducción a las Metodologías Ágiles 3
Del Modelo en Cascada a las Metodologías Ágiles 3
El origen de las Metodologías Ágiles 7
Manifiesto Ágil 7
Valores 7
Principios 8
Metodologías Ágiles existentes 8
Releases 9
Dynamic Systems Development Method (DSDM) 9
Essential Unified Process (EssUP) 11
Extreme Programming (XP) 11
Valores 12
Características 13
Lean Software Development 14
La filosofía Lean 16
Kanban 16
Open Unified Process (OpenUP) 17
Principios 17
Fases 18
Áreas de interés 18
Scrum 19
Introducción a Scrum 20
¿Qué es Scrum? 20
Cambio organizacional 21
Roles de Scrum 22
Equipo de Desarrollo 22
Product Owner 22
ScrumMaster 23
El ScrumMaster es un Líder Facilitador 24
Elementos de Scrum 25
Product Backlog 25
Priorización por valor de negocio de cada PBI 25
Priorización por retorno de la inversión (return on investment, ROI) de cada PBI 25
Prioridades según la importancia y el riesgo de cada PBI 26
Backlog de Impedimentos 26
Dinámica (flujo del trabajo) 27
Iteraciones (Sprints) 27
Retrasos y adelantos en un Sprint 27
Incremento funcional potencialmente entregable 27
Sprint Planning Meeting (Reunión de Planificación de Sprint) 28
Parte estratégica: ¿qué vamos a hacer? 28
Parte táctica: ¿cómo lo vamos a hacer? 29
Reuniones diarias 29
Reunión de revisión del producto 30
Reunión de retrospectiva 31
Reunión de refinamiento del Backlog 31



Esta obra fue realizada por Martín Alaimo para KLEER, con aportes de Pablo Tortorella y Daniela Casquero y se encuentra bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported.

Introducción a las Metodologías Ágiles

Del Modelo en Cascada a las Metodologías Ágiles

El desarrollo de software no es una disciplina sencilla. En las últimas décadas los lenguajes de modelado (UML)[1] y posteriormente varias herramientas[2] intentaron sin éxito posicionarse como las "balas de plata"[3] para resolver algunos de sus problemas. Incluso se había llegado a contar con herramientas poderosas y procesos de modelado y diseño sin tener muy en claro cómo aplicarlos a la hora de construir el software.

No fue sino hasta la adopción amplia y consciente de un tercer elemento injustamente relegado a un papel secundario, las metodologías de desarrollo, que se encontraron soluciones adecuadas a muchos problemas. La mayoría de estas metodologías fueron introducidas desde la ingeniería civil, lo que resultó en un exhaustivo control sobre los procesos y las tareas.

El Modelo Secuencial de Procesos, también conocido como Waterfall Model o Modelo en Cascada, se convirtió en el modelo metodológico más utilizado dentro de la industria. Data de principios de los años setenta y tiene sus orígenes en los ámbitos de la manufactura y la construcción, ambientes físicos altamente rígidos donde los cambios se vuelven prohibitivos desde el punto de vista de los costos, sino prácticamente imposibles. Como no existía proceso alguno en la industria del software, esta condición no impidió su adopción.

La primera mención pública (reconocida) de este tipo de metodologías fue realizada en un artículo que data de 1970 donde el Dr. Winston W. Royce[4] presenta -sin mencionar la palabra "Waterfall"- un modelo secuencial para el desarrollo de software que comprendía las siguientes fases:

- Especificación de requerimientos
- Diseño
- Construcción (también conocida como implementación o codificación)
- Integración
- Verificación o prueba y debugging
- Instalación
- Mantenimiento

---

1 Edward Yourdon, *Análisis Estructurado Moderno*, Prentice-Hall, 1993
Unified Modeling Language (UML), Information technology, ISO/IEC 19501:2005
http://www.uml.org

2 Herramientas CASE (U-CASE, M-CASE, L-CASE): http://es.wikipedia.org/wiki/Herramienta_CASE

3 El término ha sido adoptado como metáfora referida a cualquier solución sencilla que tiene una eficacia extrema. Suele aparecer con la expectativa de que algún nuevo desarrollo tecnológico o la práctica fácil de implementar resuelva alguno de los problemas vigentes principales. Fuente: Wikipedia

4 Winston Royce, *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26, 1970: 1–9

El proceso Waterfall sugiere una evolución secuencial. Por ejemplo: primero se realiza la fase de especificación de requerimientos. Una vez que se encuentra completa se procede a un "sign-off" (firma/aprobación) que *congela* dichos requerimientos, y es recién aquí cuando se inicia la fase de diseño del software, fase donde se produce un plano o "blueprint" del mismo para que los codificadores/programadores lo implementen.

Hacia el final de la fase de implementación, diferentes componentes desarrollados son integrados con el fin de pulir las interfaces entre ellos. El siguiente paso es la fase de verificación en la que los testers someten el sistema a diferentes tipos de pruebas funcionales mientras los programadores corrigen el código donde sea necesario. Una vez que el sistema responde satisfactoriamente a la totalidad de las pruebas, se inicia una etapa de instalación y mantenimiento posterior.

Los problemas detectados en los modelos tradicionales o de tipo Waterfall se fundamentan, por un lado, en el entorno altamente cambiante propio de la industria, y por el otro, en 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 empleados.

A medida que han pasado los años, y con el advenimiento de las economías globalizadas y los entornos web, el contexto de negocio de los sistemas ha pasado de ser relativamente estable a convertirse en un contexto altamente volátil, donde los requerimientos expresados hoy, en muy pocas oportunidades son válidos unos meses más tarde. Bajo esta nueva realidad, las metodologías Waterfall resultaron muy “pesadas” y prohibitivas para responder satisfactoriamente a los cambios de negocio.

En el año 1994 el Standish Group publicó un estudio conocido como el "CHAOS Report"[5] donde se encontró la siguiente tasa de éxito en los proyectos de desarrollo de software en general:

- 31.1% es cancelado en algún punto durante el desarrollo del mismo
- 52.7% es entregado con sobrecostos, en forma tardía o con menos funcionalidades de las inicialmente acordadas
- 16.2% es entregado en tiempo, dentro de los costos y con las funcionalidades comprometidas

Los datos publicados, entre otros, mostraron estos índices:

Sobrecostos	% de Respuestas
Menos del 20%	15.5%
21 - 50%	31.5%
51 - 100%	29.6%
101 - 200%	10.2%
201 - 400%	8.8%
Mayor al 400%	4.4%

5 The CHAOS Report (1994), Standish Group - http://www.standishgroup.com/sample_research/chaos_1994_1.php

Funcionalidad entregada	% de Respuestas
Menos del 25%	4.6%
25 - 49%	27.2%
50 - 74%	21.8%
75 - 99%	39.1%
100%	7.3%

Factores mas importantes para el éxito de un proyecto	% de Respuestas
Involucramiento del usuario	15.9%
Apoyo de la gerencia	13.9%
Claridad en los requerimientos	13.0%
Planificación apropiada	9.6%
Expectativas realistas	8.2%
Hitos más acotados	7.7%
Staff competente	7.2%
Compromiso	5.3%
Objetivos y visión claros	2.9%
Staff enfocado y dedicado	2.4%
Otros	13.9%

Factores mas comunes de cancelación de proyectos	% de Respuestas
Requerimientos incompletos	13.1%
Falta de involucramiento del usuario	12.4%
Falta de recursos	10.6%
Expectativas irreales	9.9%
Falta de soporte gerencial	9.3%
Requerimientos y especificaciones cambiantes	8.7%
Falta de planificación	8.1%
No se necesitaba más	7.5%
Falta de gestión IT	6.2%
Analfabetismo técnico	4.3%
Otros	9.9%

Factores mas importantes de desafío para los proyectos	% de Respuestas
Falta de input del usuario	12.8%
Requerimientos y especificaciones incompletas	12.3%
Requerimientos y especificaciones cambiantes	11.8%
Falta de apoyo gerencial	7.5%
Falta de conocimientos técnicos	7.0%
Falta de recursos	6.4%
Expectativas irreales	5.9%
Objetivos poco claros	5.3%
Calendario poco realista	4.3%
Nuevas tecnologías	3.7%
Otros	23.0%

Demora	% de Respuestas
Menos del 20%	13.9%
21 - 50%	18.3%
51 - 100%	20.0%
101 - 200%	35.5%
201 - 400%	11.2%
Mayor al 400%	1.1%

Las conclusiones de la investigación sugieren que el involucramiento del usuario y el empleo de periodos de tiempo más cortos son claves para incrementar las tasas de proyectos exitosos

Bajo esta realidad surgieron nuevas metodologías, como por ejemplo:

- Metodologías en Espiral
- Metodologías Iterativas
- Metodologías Ágiles

Tanto las Metodologías en Espiral como las Metodologías Iterativas se encuentran fuera del alcance de este manual.

El origen de las Metodologías Ágiles

En febrero de 2001 se reunieron en Utah (EEUU) un grupo de diecisiete profesionales reconocidos del desarrollo de software con el objetivo de determinar los valores y principios que les permitirían a los equipos desarrollar software de forma más rápida y responder mejor a los cambios que pudieran surgir a lo largo de un proyecto de desarrollo. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por la rigidez y dominados por la documentación.

En esta reunión se creó la Agile Alliance[6], una organización sin fines de lucro cuyo objetivo es el de promover los valores y principios de la filosofía ágil y ayudar a las organizaciones en su adopción. La piedra angular del movimiento ágil es conocida como Manifiesto Ágil (Agile Manifesto[7]).

Manifiesto Ágil

El Manifiesto Ágil se compone de 4 valores y 12 principios.

Valores

Valorar a las personas y las interacciones entre ellas por sobre los procesos y las herramientas

Las personas son el principal factor de éxito de un proyecto de software. Es más importante construir un buen equipo que construir el contexto. Muchas veces se comete el error de construir primero el entorno de trabajo y esperar que el equipo se adapte automáticamente. Por el contrario, Scrum propone crear el equipo y que éste construya su propio entorno y procesos en base a sus necesidades.

Valorar el software funcionando por sobre la documentación detallada

La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar un decisión importante". Estos documentos deben ser cortos y centrarse en lo esencial. La documentación (diseño, especificación técnica de un sistema) no es más que un resultado intermedio y su finalidad no es dar valor en forma directa al usuario o cliente del proyecto. Medir avance en función de resultados intermedios se convierte en una simple "ilusión de progreso".

Valorar la colaboración con el cliente por sobre la negociación de contratos

Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta mutua colaboración será la que dicte la marcha del proyecto y asegure su éxito.

Valorar la respuesta a los cambios por sobre el seguimiento estricto de los planes

La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también su éxito o fracaso. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

6 http://www.agilealliance.org
7 http://www.agilemanifesto.org

Principios

Los valores anteriores son los pilares sobre los cuales se construyen los doce principios del Manifiesto Ágil. De estos doce principios, los dos primeros son generales y resumen gran parte del espíritu ágil del desarrollo de software, mientras que los siguientes son más específicos y orientados al proceso o al equipo de desarrollo:

1. Nuestra mayor prioridad es satisfacer al cliente a través de entregas tempranas y frecuentes de software con valor.
2. Aceptar el cambio incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan los cambios para darle al cliente ventajas competitivas.
3. Entregar software funcionando en forma *frecuente*, desde un par de semanas a un par de meses, prefiriendo el periodo de tiempo más corto.
4. Expertos del negocio y desarrolladores deben trabajar juntos diariamente durante la ejecución del proyecto.
5. Construir proyectos en torno a personas motivadas, generándoles el ambiente necesario, atendiendo sus necesidades y confiando en que ellos van a poder hacer el trabajo.
6. La manera más eficiente y efectiva de compartir la información dentro de un equipo de desarrollo es la conversación cara a cara.
7. El *software funcionando* es la principal métrica de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los sponsors, desarrolladores y usuarios deben poder mantener un ritmo constante indefinidamente.
9. La atención continua a la excelencia técnica y buenos diseños incrementan la agilidad.
10. La simplicidad –el arte de maximizar la cantidad de trabajo no hecho- es esencial.
11. Las mejores arquitecturas, requerimientos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares, el equipo reflexiona acerca de cómo convertirse en más efectivos, luego mejora y ajusta su comportamiento adecuadamente.

Metodologías Ágiles existentes

A continuación se desarrollarán las siguientes metodologías: Agile Unified Process (AUP), Dynamic Systems Development Method (DSDM), Essential Unified Process (EssUP), Extreme Programming (XP), Lean Software Development, Kanban, Open Unified Process (OpenUP), Projects in Controlled Environments (PRINCE2) y por último, Scrum.

Agile Unified Process (AUP)

El Agile Unified Process (AUP) es una versión simplificada del IBM Rational Unified Process (RUP)[8]. Describe un enfoque simple y comprensible para la construcción de sistemas mediante la utilización de prácticas y técnicas ágiles sin dejar de cumplir con RUP.
A diferencia de este último, AUP cuenta con 7 disciplinas:

1. **Modelado**: entender el modelo de negocio de la organización, comprender el modelo de dominio sobre el que trata el proyecto e identificar una solución viable que lo resuelva.
2. **Implementación**: transformación del modelo en código y realización de un nivel básico de pruebas, en particular pruebas unitarias (Unit Testing[9]).
3. **Prueba**: realización de una evaluación objetiva para garantizar la calidad del sistema construido. Esto incluye encontrar defectos, validar que el sistema se comporte según se ha diseñado y que los requerimientos sean cumplidos.
4. **Despliegue**: planificación para la entrega del sistema y ejecución del plan para disponibilizar el sistema a los usuarios finales.
5. **Gestión de la configuración**: gestión del acceso a los diferentes artefactos del proyecto. No solo gestiona las versiones de éstos a través del tiempo, sino además sus cambios.
6. **Gestión del proyecto**: dirige las actividades que se realizan como parte de la gestión del proyecto en sí. Esto incluye la gestión de riesgos, alcance, recursos humanos, tiempo, costos, proveedores, contrataciones, etc.
7. **Ambiente**: soporta el resto del proyecto asegurando que los procesos adecuados, estándares, guías y herramientas estén a disposición del equipo cuando éste los necesita.

Releases

El proceso AUP distingue entre dos tipos diferentes de iteraciones:

- **Iteración de desarrollo**: realiza el despliegue del producto terminado en un ambiente de QA o Demo.
- **Iteración de entrega**: realiza el despliegue del producto en producción.

Dynamic Systems Development Method (DSDM)

Se trata de un proceso de desarrollo ágil que divide el proyecto en tres fases: pre-proyecto, ciclo de vida y post-proyecto. La fase de ciclo de vida, a su vez, se divide en en cinco instancias: estudio de factibilidad, estudio de negocio, iteración del modelo funcional, iteración de diseño y construcción, e implementación.

8 http://www-01.ibm.com/software/awdtools/rup/?S_TACT=105AGY59&S_CMP=WIKI&ca=dtl-08rupsite
9 http://en.wikipedia.org/wiki/Unit_testing

Pre-proyecto

Durante esta fase se identifican los proyectos candidatos, se asegura la financiación del proyecto y se realizan los compromisos referidos a él.

Ciclo de vida del proyecto

- **Estudio de factibilidad**

  En esta instancia se estudia la factibilidad de la utilización de DSDM para la gestión del proyecto. Esta etapa produce cuatro entregables: el reporte de factibilidad, el prototipo de factibilidad, el plan global para el resto del proyecto y la bitácora de riesgos identificados.

- **Estudio de negocio**

  Se realizan actividades con los stakeholders[10] del proyecto, generalmente workshops/talleres, para identificar los requerimientos que luego serán volcados a una lista priorizada de requerimientos. Esta asignación de prioridades se realiza con la técnica MoSCoW[11]. Los entregables principales de esta etapa son la lista priorizada de requerimientos, una definición de área de negocios que describe el proyecto y su entorno de negocio, una definición global inicial de arquitectura y un plan de desarrollo. A partir de este punto el proyecto se torna iterativo.

- **Iteración del modelo funcional**

  En esta etapa iterativa los requerimientos son convertidos en un prototipo funcional contra el cual se ejecuta una serie de verificaciones (testing) funcionales. Habitualmente divisible en cuatro sub-etapas:

  - **Identificación del prototipo funcional:** determinación de las funcionalidades a ser implementadas como parte de esta iteración.
  - **Acuerdo del cronograma**: se acuerdan las fechas y forma de desarrollo de las funcionalidades.
  - **Creación del prototipo funcional**: es aquí donde se realiza la investigación, refinamiento y consolidación con la funcionalidad construida en iteraciones anteriores.
  - **Revisión funcional**: mediante esta sub-etapa se valida la funcionalidad construida.

- **Iteración de diseño y construcción**

  El interés principal es la integración de la funcionalidad construida durante la etapa previa en un sistema mayor que satisfaga las necesidades del usuario. El testing también constituye una parte importante, donde se atienden también requerimientos no-funcionales. También se puede dividir en cuatro sub-etapas:

10 Se llama stakeholder a todo aquel interesado o que se vea afectado por las actividades de un proyecto. R. E. Freeman, *Strategic Management: A Stakeholder Approach,* 1984.

11 http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method#moscow

- **Identificación del prototipo de diseño:** identificación de los requerimientos funcionales y no funcionales que deben ser incorporados.
- **Acuerdo del cronograma**: acuerdo sobre la forma y fechas del desarrollo de los requerimientos identificados en la sub-etapa anterior.
- **Creación del prototipo de diseño**: creación de un sistema que puede ser entregado a los usuarios finales para su utilización.
- **Revisión del prototipo de diseño**: verificación (testing) del sistema construido. Tanto los resultados del testing como el feedback del usuario se transformarán en la documentación de usuario.

- **Implementación**

  En este período, el sistema y la documentación son entregados a los usuarios finales y se realizan entrenamientos de futuros usuarios. Por lo general, esta etapa también se puede dividir en cuatro sub-etapas:

  - **Aprobación y guías de usuario**: los usuarios finales aprueban el sistema para su pasaje a producción y se crean las guías de usuario referentes al uso del sistema.
  - **Entrenamiento de usuarios**: se entrenan usuarios finales y futuros usuarios en la utilización del sistema.
  - **Implementación**: instalación del sistema en el lugar donde los usuarios finales lo utilizarán.
  - **Revisión de negocio**: se realiza una revisión del impacto que la instalación del sistema tiene en el negocio. Desde este punto el ciclo de vida de desarrollo puede moverse a la próxima etapa (post-proyecto) o hacia una nueva iteración si se necesita desarrollo adicional.

Post-proyecto

El foco de esta fase se encuentra en lograr que el sistema opere de forma eficiente. La actividad que caracteriza esta etapa es el mantenimiento evolutivo y/o correctivo.

Essential Unified Process (EssUP)

Es un conjunto de prácticas que conforma el conocimiento esencial de un ciclo de vida de desarrollo de software. Este enfoque de la práctica centrada en el desarrollo de software incorpora y se basa en prácticas establecidas de la industria. Las prácticas en EssUP integran los principios del éxito del proceso unificado, la agilidad y los conceptos de Madurez de Procesos o Capability Maturity Model Integration(CMMI), aprovechando sus diferentes capacidades: estructura, agilidad y mejora de procesos.

Extreme Programming (XP)[12]

También conocido como programación extrema o XP, este conjunto de metodologías enfatiza

12 Kent Beck, *Extreme Programming Explained: Embrace Change,* 1999.

las prácticas de ingeniería de software. La programación extrema se diferencia de las metodologías tradicionales, al igual que las metodologías ágiles en general, por ser un enfoque basado en la adaptabilidad más que en la previsibilidad.

XP considera que los cambios de requerimientos durante el ciclo de vida de un proyecto son algo natural y hasta deseable en el desarrollo de software: poder incorporar cambios en cualquier momento durante el desarrollo de un sistema es una aproximación más realista que aquellas que intentan definir todo desde un principio y condensar el esfuerzo en controlar y evitar los cambios.

Valores

Los valores originales de la programación extrema son: simplicidad, comunicación, retroalimentación (feedback) y coraje. Un quinto valor, el respeto, fue añadido en la segunda edición del libro Extreme Programming Explained (Kent Beck). Los cinco valores[13] se explican así:

Simplicidad

Descripto por la agilidad como "el arte de maximizar la cantidad de trabajo no realizado", la simplicidad es el valor principal de la programación extrema. Simplificar los diseños y arquitecturas agiliza el desarrollo y facilita el mantenimiento. Un diseño complejo del código, junto con sucesivas modificaciones realizadas por diferentes desarrolladores, aumenta la complejidad en forma exponencial. Una de las prácticas fundamentales a la hora de mantener la simplicidad en el diseño es la llamada refactorización (sobre la que se tratará más adelante).

Otro punto importante sobre la simplicidad es la documentación, donde se busca que el código se comente lo justo y necesario, y que por medio de técnicas de nomenclatura de variables, métodos y clases, el código se transforme en un activo autodocumentado.

Comunicación

La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo legible. El código autodocumentado es más fiable que los comentarios ya que estos últimos pronto quedan desfasados con el código a medida que es modificado. Por ello debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. Los programadores se comunican constantemente gracias a la programación por parejas y la comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente es quien decide qué características tienen prioridad y él siempre debe estar disponible para despejar dudas.

Feedback

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se

13 http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema

conoce en tiempo real. Como los ciclos son muy cortos, los resultados se muestran con frecuencia y se minimiza la necesidad de rehacer partes que no cumplan con los requisitos, ayudando a los programadores a centrarse en lo que es más importante.

Considérense los problemas derivados de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el "estado de salud" del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallas debido a cambios recientes producidos en el código.

Coraje

Los puntos anteriores parecen tener sentido común, entonces ¿por qué coraje? Para la alta gerencia, la programación en parejas puede ser difícil de aceptar porque a primera vista pareciera que la productividad disminuye a la mitad ya que solo la mitad de los programadores está escribiendo código. Hay que ser valiente para confiar en que la programación por parejas beneficia la calidad del código sin repercutir negativamente en la productividad.

La simplicidad es uno de los principios más difíciles de adoptar. Se requiere coraje para implementar las características que el cliente quiere ahora sin caer en la tentación de optar por un enfoque más flexible que permita futuras modificaciones. No se debe emprender el desarrollo de grandes marcos de trabajo (frameworks) mientras el cliente espera. En ese tiempo el cliente no recibe noticias sobre los avances del proyecto y el equipo de desarrollo no recibe retroalimentación para saber si va en la dirección correcta. La forma de construir marcos de trabajo es mediante la refactorización del código en sucesivas aproximaciones.

Respeto

El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a los otros porque los programadores no pueden realizar cambios que hagan que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la refactorización del código. Finalmente, respetan el trabajo del resto no menospreciando a otros, sino orientándolos a mejorar, obteniendo como resultado un mayor autoestima en el equipo y elevando su ritmo de producción.

Características

- **Simplicidad**: mantener los diseños y arquitecturas simples. No sobredimensionar y evitar el hecho de caer en la predicción de funcionalidades.
- **Estándares de codificación**: el objetivo de un estándar de codificación es que todos los programadores escriban el código siguiendo una serie común de parámetros para que parezca escrito en su totalidad por una única persona.
- **Propiedad colectiva**: el principio de la propiedad colectiva intenta evitar que ciertas porciones de código o del sistema en sí pertenezcan en la práctica a un programador o

grupo de programadores específicos sin dejar a otros acceder y/o modificar dicho código. En XP, todos son dueños de todo y todos están autorizados a revisar y cambiar el código de las aplicaciones cuando lo crean conveniente, sin importar quien lo haya escrito. Este principio debe estar reforzado por medio de la simplicidad y los estándares de codificación.

- **Pruebas unitarias**: las pruebas unitarias son fragmentos de código destinados a evaluar el comportamiento de los diferentes módulos o componentes en forma aislada del resto de la solución.
- **Pruebas automatizadas**: debería haber tantas pruebas unitarias, de integración y aceptación automáticas como sea posible. En la programación extrema existe la idea de que cualquier acción que se repita más de tres veces es candidata a ser automatizada.
- **Integración continua**: este principio determina que el sistema debe estar integrado todo el tiempo, y funcionando. El hecho de que esté funcionando hace referencia a que la totalidad de las pruebas automatizadas debe ejecutarse de forma exitosa. Que sea continuamente tiene como finalidad que tan pronto como se detecte un cambio en la base de código, se deban ejecutar las pruebas.
- **Programación de a pares**: está demostrado que la productividad y la calidad del código producidos por una pareja de programadores es mucho mayor al resultado obtenido por la suma de los logros de los programadores en forma aislada.
- **Desarrollo iterativo e incremental**: la solución o aplicación de software debe ser construida en ciclos cortos, entregando software frecuentemente y logrando así feedback temprano y continuo.

Lean Software Development[14]

La filosofía Lean de desarrollo de software surge de la mano de Mary y Tom Poppendieck, quienes basados en el enfoque de Lean Manufacturing, adaptaron sus principios al desarrollo de software. Los principios Lean resultantes son:

Eliminar los desperdicios

Mediante una técnica conocida como Value Stream Mapping se debe analizar los procesos de desarrollo e identificar los desperdicios y eliminarlos, en forma iterativa, hasta llegar a incluso eliminar aquellos que se creían esenciales.

Se considera desperdicio todo aquello que no aporta valor al cliente, como por ejemplo:

- Código y funcionalidades innecesarias
- Retraso en el proceso de desarrollo de software (cuellos de botella)
- Requisitos poco claros
- Burocracia (*micromanagement*, gestión sobrante)

14 http://en.wikipedia.org/wiki/Lean_software_development

- Comunicación interna lenta

Amplificar el aprendizaje

El desarrollo de software es una disciplina que exige un continuo proceso de aprendizaje, especialmente el entendimiento de los requerimientos de negocio como la mitigación de riesgos técnicos.

El proceso de aprendizaje es potenciado con el uso de iteraciones cortas, cada una de ellas acompañada con refactorización y sus pruebas de integración, incrementando el feedback mediante reuniones frecuentes y ciclos de entrega cortos con el usuario.

Decidir lo más tarde posible

El desarrollo de software está siempre asociado con cierto grado de incertidumbre, los mejores resultados se alcanzan con un enfoque basado en opciones y retrasando las decisiones tanto como sea posible hasta que se basen en hechos y no en suposiciones y pronósticos inciertos. Esto también permite la adaptación tardía a los cambios y previene de las costosas decisiones delimitadas por la tecnología.

Reaccionar tan rápido como sea posible

Cuanto antes el producto final se entrega sin defectos considerables, más pronto se puede recibir feedback del cliente e incorporarse en la siguiente iteración. Cuanto más cortas sean las iteraciones, mejor es el aprendizaje y la comunicación dentro del equipo.

Potenciar el equipo

Uno de los pilares de Lean establece: “encontrar buena gente y dejarle hacer su trabajo”. Ha crecido una creencia en la mayoría de las empresas tradicionales acerca de la toma de decisiones: los gerentes deben decirle a los trabajadores cómo hacer su propio trabajo. En cambio, en una aproximación work-out se enseña a los gerentes a escuchar a los desarrolladores, de manera que ellos puedan explicar mejor qué acciones podrían tomarse, así como también ofrecer sugerencias para mejoras.

Integridad innata

El cliente debe contar permanentemente con una experiencia general del sistema: a esto se llama percepción de integridad. La integridad conceptual significa que los componentes separados del sistema funcionan bien juntos, como en un todo, logrando equilibrio entre la flexibilidad, mantenibilidad, eficiencia y capacidad de respuesta. Esto podría lograrse a través de la comprensión del dominio del problema y resolviéndolo en forma evolutiva, todo junto al mismo tiempo, y no secuencialmente.

Una de las maneras más saludables de transitar hacia una integridad innata es la refactorización. Cuanto más funcionalidades se añadan a las del sistema, más se perderá del código base para futuras mejoras. Mediante la refactorización se intenta mantener la sencillez, la claridad y la cantidad mínima de funcionalidades en el código. Las duplicaciones en el código son signo de un mal diseño y deben evitarse.

El proceso de construcción debe ir acompañado de una suite completa y automatizada de pruebas, tanto para desarrolladores como clientes que tengan la misma versión, sincronización y semántica que el sistema actual; las cuales se mantienen en constante ejecución, garantizando así la integración continua de todos los componentes.

Al final, la integridad debe ser verificada con una prueba global, asegurando que el sistema haga lo que el cliente espera. Las pruebas automatizadas también son consideradas como parte del proceso de producción y, por tanto, si no agregan valor deben considerarse residuos. Las pruebas automatizadas no deberían ser un objetivo, sino más bien un medio para lograr un fin, específicamente para la reducción de defectos.

Ver todo el conjunto

Los sistemas de software hoy en día no son simplemente la suma de sus partes, sino también el producto de sus interacciones. Los defectos en el software tienden a acumularse durante el proceso de desarrollo, a causa de la descomposición de las grandes tareas en pequeñas tareas y la normalización de las diferentes etapas de desarrollo. Las causas reales de los defectos deben ser encontradas y eliminadas.

Cuanto mayor sea el sistema, más serán las organizaciones que participen en su desarrollo y más partes serán las desarrolladas por diferentes equipos. Por eso también es mayor la importancia de tener bien definidas las relaciones entre los diferentes proveedores con el fin de producir una buena interacción entre los componentes del sistema.

La filosofía Lean

La filosofía Lean tiene que ser bien entendida por todos los miembros de un proyecto antes de aplicar Lean de manera concreta en una situación de la vida real. "Piensa en grande, actúa en pequeño, equivócate rápido; aprende con rapidez". Esta consigna resume la importancia de comprender el terreno y la idoneidad de implementar los principios Lean a lo largo del proceso de desarrollo de software. Solo cuando todos los principios de Lean se apliquen al mismo tiempo, combinado con un fuerte "sentido común" en relación con el ambiente de trabajo, habrá una base para el éxito en el desarrollo de software.

Kanban

Kanban es un enfoque Lean de desarrollo de software ágil. Literalmente, Kanban es una palabra japonesa que significa "tarjeta visual". En Toyota, Kanban es el término que se utiliza para el sistema de señalización visual y física que une todo el sistema de producción Lean. Curiosamente, mientras Kanban en su aplicación al desarrollo de software es nuevo, Kanban en la producción Lean tiene más de medio siglo de edad.

La mayor parte de los métodos ágiles como Scrum y XP se encuentran alineados con los principios Lean. Sin embargo, en 2004 David Anderson fue pionero en una aplicación más directa de Lean Thinking y Teoría de las Restricciones al Desarrollo de Software. Bajo la guía de expertos como Don Reinertsen, se convirtió en lo que David llama un "Sistema Kanban para el Desarrollo de Software", al que la mayoría de la gente ahora se refiere simplemente como "Kanban" y que a grandes rasgos significa:

- **Visualizar el flujo de trabajo:** dividir el trabajo en pequeñas porciones, escribir cada ítem en una tarjeta y ponerla en la pared. Utilizar el nombre de las columnas para ilustrar donde está cada elemento dentro del flujo de trabajo.
- **Límite WIP (work in progress):** asignar límites explícitos a la cantidad de ítems que podrían estar en marcha en cada uno de los estados del flujo de trabajo.
- **Medir el tiempo de espera (lead time):** tiempo promedio para completar un ítem, a veces llamado tiempo de ciclo o cicle time. Optimizar el proceso para que el tiempo de entrega sea lo más pequeño y predecible posible.

Open Unified Process (OpenUP)

OpenUP[15] es parte del Eclipse Process Framework (EPF)[16], un proceso de desarrollo de software open source. EPF nació bajo la tutoría de varias empresas[17], entre ellas IBM, que luego lo donaron a la Fundación Eclipse[18].

OpenUP solo incorpora las partes fundamentales del Rational Unified Process (RUP), su metodología madre, y no provee lineamientos para todos los artefactos y procesos.

Principios

OpenUP se basa en los siguientes principios[19]:

- **Colaborar** para sincronizar intereses y compartir conocimiento, buscando así un ambiente de equipo saludable donde se facilite la colaboración y se desarrolle un conocimiento compartido del proyecto.
- **Equilibrar** las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto.
- **Promover** el desarrollo de una solución que maximice los beneficios obtenidos por los participantes y que cumpla con los requisitos y restricciones del proyecto.
- **Centrarse** en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
- **Desarrollo evolutivo** para obtener retroalimentación y mejora continua.
- Obtener **retroalimentación temprana** y continua de los participantes del proyecto mediante entregas frecuentes de incrementos progresivos en la funcionalidad.

15 http://epf.eclipse.org/wikis/openup/index.htm
16 http://www.eclipse.org/epf/
17 Derechos de Autor de OpenUP: http://epf.eclipse.org/wikis/openup/core.default.release_copyright.base/guidances/supportingmaterials/openup_copyright_C3031062.html
18 http://es.wikipedia.org/wiki/Eclipse_(software)
19 http://es.wikipedia.org/wiki/OpenUP

Fases

OpenUP divide el proyecto en cuatro fases orientadas a diferentes disciplinas cada una:

Concepción (inception)

Fase compuesta por "n" iteraciones de concepción destinadas a:

- Dar inicio y panificar el proyecto
- Elaborar la visión técnica
- Identificar requerimientos
- Elaborar los casos de uso detallados

Elaboración

Fase compuesta por "n" iteraciones destinadas a:

- Identificar y refinar requerimientos
- Identificar, desarrollar y entregar incrementos de la arquitectura base de la solución
- Desarrollar, probar y entregar incrementos funcionales de la solución

Construcción

Fase compuesta por "n" iteraciones destinadas a:

- Identificar y refinar requerimientos
- Desarrollar, probar y entregar incrementos funcionales de la solución

Transición

Fase compuesta por "n" iteraciones destinadas a:

- Identificar y refinar requerimientos de mantenimiento evolutivo y/o correctivo
- Desarrollar, probar y entregar incrementos funcionales de mantenimiento evolutivo y/o correctivo

Áreas de interés

Las áreas de interés de la metodología OpenUP hacen énfasis en tres niveles diferentes del contexto:

- El nivel personal
- El nivel del equipo

- El nivel de proyecto

PRINCE2

PRINCE2 (PRojects IN Controlled Environments o Proyectos en Entornos Controlados) es un método basado en procesos para la gestión eficaz de proyectos. PRINCE2 es un estándar de facto utilizado ampliamente por el gobierno del Reino Unido y reconocido y utilizado en el sector privado de múltiples países.

El método PRINCE2 es de dominio público, ofreciendo guías de mejores prácticas no propietarias en gestión de proyectos. PRINCE2 es una marca registrada de la OGC[20]. Las características principales de PRINCE2 son:

- Se centra en la justificación de negocios
- Su planificación basada en el producto enfoque
- Su énfasis en la división del proyecto en fases manejables y controlables
- Una organización define la estructura para el equipo de gestión de proyectos
- La flexibilidad que se aplicará a un nivel apropiado para el proyecto

Scrum

Scrum más que una metodología es un framework (marco de trabajo) para la gestión de proyectos complejos. Se basa principalmente en la premisa de ejecutar un proyecto en iteraciones de entre dos y cuatro semanas, llamadas Sprints, y de duración fija (timebox). Esto quiere decir que las fechas de finalización de cada iteración no pueden ser pospuestas.

El desarrollo del producto se realiza en forma incremental y evolutiva, teniendo como base la priorización de características según el valor de negocio que las mismas representan, entregando las características de mayor valor al comienzo y dejando para las iteraciones finales las características de menor valor de negocio.

Desarrollaremos Scrum en detalle en el próximo capítulo: Introducción a Scrum.

20 OGC: Office of Government Commerce http://www.ogc.gov.uk/

Introducción a Scrum

¿Qué es Scrum?

Scrum es un enfoque ágil para el desarrollo de software. No es un proceso completo o una metodología, sino un marco de trabajo. En lugar de proporcionar una descripción completa y detallada de cómo deben realizarse las tareas de un proyecto, deja mucho en manos del equipo de desarrollo. Esto ocurre debido a que es el equipo quien conocerá la mejor manera de resolver las problemáticas que se presenten.

El equipo de desarrollo se encuentra apoyado en dos roles: el ScrumMaster y el Product Owner. El ScrumMaster es quien vela por la productividad del equipo de desarrollo. Puede ser considerado un coach o líder servil/facilitador encargado de acompañar al equipo de desarrollo de forma tal que encuentre su punto de mayor eficiencia. El Product Owner es quien representa al negocio, stakeholders, cliente y usuarios finales. Tiene la responsabilidad de conducir al equipo de desarrollo hacia el producto adecuado.

El progreso de los proyectos que utilizan Scrum se realiza y verifica en una serie de iteraciones llamadas Sprints. Estos Sprints tienen una duración fija, pre-establecida de no más de un mes. Al comienzo de cada Sprint el equipo de desarrollo realiza un compromiso de entrega de una serie de funcionalidades o características del producto en cuestión.

Al finalizar el Sprint se espera que estas características comprometidas estén terminadas, lo que implica su análisis, diseño, desarrollo, prueba e integración al producto en desarrollo. En este momento es cuando se realiza una reunión de revisión del producto construido durante el Sprint, donde el equipo de desarrollo muestra lo construido al Product Owner y a cualquier stakeholder interesado en participar. El feedback obtenido en esta reunión puede ser incluido entre las funcionalidades a construir en futuros Sprints.

Principios de Scrum

Esta información fue obtenida desde el Blog de Tobias Mayer: "Essence of Scrum"[21] con aportes propios del autor.

Scrum se considera "una manera simple de manejar problemas complejos". Proporciona un marco de trabajo para soportar la innovación y permitir que equipos auto-organizados entreguen resultados de alta calidad en tiempos cortos. Scrum es un *estado de la mente*; es una manera de pensar que libera el espíritu creativo mientras se sostiene firmemente en principios sólidos y largamente respetados, incluyendo el empirismo, los elementos emergentes y la auto-organización.

Empirismo

Se refiere al proceso continuo de inspección y adaptación que permite tomar decisiones en tiempo real y en base a datos reales. Como resultado, los decisores pueden responder

21 http://agilethinking.net/essence-of-scrum

rápidamente en contextos cambiantes, como ocurre en la industria del desarrollo de software.

Elementos emergentes

Son consecuencia de una aproximación empírica. Implica que todas las soluciones a todos los problemas emergerán y se volverán visibles a medida que avancemos en los proyectos. No se volverán visibles si simplemente hablamos de ellos. El "big up front design" (gran diseño anticipado) sólo producirá un "big wrong design" (gran diseño erróneo) o a lo sumo un "big working but totally inflexible design" (gran diseño que funciona pero totalmente inflexible).

Cuando permitimos que las soluciones emerjan estamos frente a la solución más simple y apropiada para el contexto actual. Este surgimiento, junto con el empirismo, nos guiarán a la solución más apropiada y flexible (es decir que podremos cambiar).

Auto-organización

Se refiere a la estructura de los equipos que crean el producto. Se otorga autonomía a pequeños equipos multidisciplinarios para que puedan tomar decisiones importantes, necesarias para 1) crear un producto de alta calidad y 2) manejar su propio proceso. La razón de este enfoque se fundamenta en que aquellos que hacen el trabajo son quienes mejor conocen cómo hacerlo. Estos equipos trabajan de una manera altamente interactiva y generativa, donde el producto emerge del diálogo continuo, de la exploración y de la iteración. La auto-organización funciona cuando hay objetivos y límites claros.

Además de estos principios, Scrum se apoya en dos mecanismos principales: priorización y timeboxing.

Priorización

Simplemente significa que siempre hay cuestiones que son más importantes que otras. Esto es tan obvio que muchas veces se olvida cuando pensamos "necesitamos TODO ESTO AHORA". Scrum nos ayuda a volver a poner el foco en seleccionar cuáles son las cosas más importantes y hacerlas primero. Tomarse el tiempo para priorizar y ser riguroso sobre eso es esencial para el éxito de Scrum.

Timeboxing

Es un mecanismo simple para manejar la complejidad que consiste en poner límites de tiempo a una actividad. No podemos imaginar el sistema completo de una vez y todo junto. Entonces, tomamos un pequeño problema y en un corto espacio de tiempo, digamos una semana o un mes, trabajamos en solucionarlo . Los resultados de esa acción nos guiarán hacia una solución para el próximo problema y nos darán más conocimiento sobre las necesidades del sistema en conjunto. El concepto de timeboxing refuerza el hecho de cumplir nuestros compromisos dentro de los tiempos prometidos. Un timebox no se puede achicar o agrandar, esto es, la entrega del resultado no se debe adelantar ni retrasar. La variable de ajuste es el alcance del producto comprometido en un determinado timebox.

Cambio organizacional

Con Scrum, las jerarquías gerenciales de las organizaciones tienden a ser niveladas y los equipos de desarrollo tienen más contacto directo e inmediato con los clientes. El estilo de liderazgo y el ambiente de trabajo se aparta del "comando y control" y transita hacia un estilo más colaborativo. Scrum promueve el diálogo regular y abierto por sobre la documentación

extensiva, así como el acuerdo negociado es preferido a los contratos de trabajo formales e impersonales.

Las cualidades de **apertura**, **honestidad** y **coraje** son fomentadas en todos los niveles, y el beneficio individual se vuelve secundario ante el avance colectivo. Un ambiente Scrum es aquel que prioriza a la gente, donde las personas de todos los niveles muestran **respeto** y **confianza** entre ellos. Las decisiones se toman por **consenso**, antes que por imposición de alguien de mayor jerarquía y todo el conocimiento es **compartido** de una manera **transparente** y sin recelos.

Roles de Scrum

En un equipo Scrum se espera que intervengan tres roles:

Equipo de Desarrollo

El equipo de desarrollo está formado por todos los individuos necesarios para la construcción del producto en cuestión. Es el único responsable por la construcción y calidad del producto.

El equipo de desarrollo es auto-organizado. Esto significa que no existe un líder externo que asigne las tareas ni que determine la forma en la que serán resueltos los problemas. Es el mismo equipo quien determina la forma en que realizará el trabajo y cómo resolverá cada problemática que se presente. La contención de esta auto-organización está dada por el objetivo a cumplir: transformar las funcionalidades comprometidas en software funcionando y con calidad productiva, o en otras palabras, producir un incremento funcional potencialmente entregable.

Es recomendable que un equipo de desarrollo se componga de hasta nueve personas[22]. Cada una de ellas debe poseer todas las habilidades necesarias para realizar el trabajo requerido. Esta característica se conoce como multi-funcionalidad y significa que dentro del equipo de desarrollo no existen especialistas exclusivos, sino más bien individuos generalistas con capacidades especiales. Lo que se espera de un miembro de un equipo de desarrollo es que no solo realice las tareas en las cuales se especializa sino también todo lo que esté a su alcance para colaborar con el éxito del equipo.

El equipo de desarrollo tiene tres responsabilidades tan fundamentales como indelegabes. La primera es proveer las estimaciones de cuánto esfuerzo será requerido para cada una de las características del producto. La segunda responsabilidad es comprometerse al comienzo de cada Sprint a construir un conjunto determinado de características en el tiempo que dura el mismo. Y finalmente, también es responsable por la entrega del producto terminado al finalizar cada Sprint.

Product Owner

El Product Owner es la persona responsable del éxito del producto desde el punto de vista de los stakeholders.Sus principales responsabilidades son:

22 http://blogs.collab.net/agile/2009/01/17/why-are-scrum-teams-supposed-to-be-small/

- Determinar la **visión** del producto, hacia dónde va el equipo de desarrollo
- Gestionar las **expectativas de los stakeholders**
- Recolectar los **requerimientos**
- Determinar y conocer en detalle las **características funcionales** de alto y de bajo nivel
- Generar y mantener el **release plan**: fechas de **entrega** y **contenidos** de cada una
- Maximizar la **rentabilidad** del producto
- Determinar las **prioridades** de cada una de las características por sobre el resto
- Cambiar las prioridades de las características según avanza el proyecto, acompañando así los cambios en el negocio
- Aceptar/rechazar el producto construido al final de cada Sprint y proveer **feedback** valioso para su evolución

El Product Owner se focaliza en maximizar la rentabilidad del producto. La principal herramienta con la que cuenta para poder realizar esta tarea es la priorización. De esta manera puede reordenar la cola de trabajo del equipo de desarrollo para que éste construya con mayor anticipación las características o funcionalidades más requeridas por el mercado o la competitividad comercial.

Otra responsabilidad importante del Product Owner es la gestión de las expectativas de los stakeholders mediante la comprensión completa de la problemática de negocio y su descomposición hasta llegar al nivel de requerimientos funcionales.

ScrumMaster

El ScrumMaster es el Coach del equipo y es quien lo ayuda a alcanzar su máximo nivel de productividad.

Se espera que el ScrumMaster sea un líder servil, facilitador, que acompañe al equipo de trabajo en su día a día y garantice que todos, incluyendo al Product Owner, comprendan y utilicen Scrum de forma correcta.

Las responsabilidades principales del ScrumMaster son:

- Velar por el **correcto empleo** y **evolución** de Scrum
- Facilitar el **uso de Scrum** a medida que avanza el tiempo. Esto incluye la responsabilidad de que todos asistan a tiempo a las daily meetings, reviews y retrospectivas
- Asegurar que el equipo de desarrollo sea **multi-funcional** y eficiente
- **Proteger** al equipo de desarrollo de distracciones y trabas externas al proyecto
- Detectar, monitorear y **facilitar la remoción de los impedimentos** que puedan surgir con respecto al proyecto y a la metodología[23]

23 Estos impedimentos podrán ser resueltos dentro del equipo de desarrollo (usualmente), entre diferentes equipos (*Scrum*

- Asegurar la **cooperación** y **comunicación** dentro del equipo
- Estar al corriente del **progreso de las actividades** del equipo de desarrollo, de las nuevas tareas que hayan surgido como consecuencia del trabajo que el equipo de realiza y de los cambios en las estimaciones
- Mantener actualizadas las métricas que denotan el avance del Sprint

Además de estas cuestiones, el ScrumMaster debe detectar **problemas y conflictos interpersonales** dentro del equipo de trabajo. Para respetar la filosofía auto-organizativa del equipo, lo ideal es que el equipo mismo sea quien resuelva estas cuestiones. En el caso de no poder hacerlo, deberá involucrarse al ScrumMaster y eventualmente a niveles más altos de la gerencia.

El ScrumMaster es un Líder Facilitador

No es casualidad la aparición de un nuevo nombre o rol. Este nuevo concepto del enfoque ágil representa el cambio respecto de las responsabilidades y el modelo de gestión de los gerentes de proyectos tradicionales en relación al equipo de trabajo.

El ScrumMaster puede ser visto como un Facilitador, incluso muchas veces se lo referencia así en lugar de ScrumMaster. Su responsabilidad es asegurar que se cumpla con el proceso de Scrum sin interferir directamente en el desarrollo del producto final. Es importante establecer que el equipo de Scrum elige la forma de trabajo que más prefiera, siempre que se cumplan las pautas básicas de Scrum, por ello mientras lo hagan no existe una forma "errónea" de trabajar.

El rol del ScrumMaster también incluye asegurar que el desarrollo del producto tenga la mayor probabilidad de ser completado de forma exitosa. Para lograr este cometido, trabaja de cerca con el Product Owner asegurando una correcta priorización de los requerimientos, por un lado, y con el equipo de desarrollo para convertir los requerimientos en un producto funcionando, por el otro.

Por lo que hemos visto, el ScrumMaster tiene un rol más indirecto que un Gerente de Proyectos tradicional, a pesar de esto es un rol vital para el éxito de Scrum. Para todo Gerente de Proyectos tradicional, el cambio hacia esta nueva filosofía de gestión es desafiante. Se dice que "Scrum es fácil, hacer Scrum es difícil[24]". Esta afirmación tiene sus fundamentos en la idea de que una cosa es aprender Scrum y otra muy diferente es aplicar Scrum exitosamente. Emprender este camino significa adoptar una filosofía de liderazgo servil por sobre el comando y control.

Finalmente, cuando un ScrumMaster logra cubrir exitosamente su rol, la implementación de Scrum sucede sin sobresaltos. Las responsabilidades del ScrumMaster deberían cubrir la totalidad de su tiempo. Si bien hay casos en los que el ScrumMaster cumple, además de su rol, el rol de desarrollador, no siempre es la mejor de las situaciones ya que ambas responsabilidades podrían llegar a exceder la disponibilidad de una sola persona, y así alguno de ambos roles no estaría siendo cubierto satisfactoriamente.

---

de Scrums) o con la intervención de la gerencia.

24 "Scrum is simple, doing Scrum is hard" - Jim York, CST.

Elementos de Scrum

El proceso de Scrum posee una mínima cantidad necesaria de elementos formales para poder llevar adelante un proyecto de desarrollo. A continuación describiremos cada uno de ellos.

Product Backlog

El primero de los elementos, y principal de Scrum, es el Backlog del Producto o también conocido como Pila del Producto o Product Backlog.

El Backlog del Producto es básicamente un listado de ítems (Product Backlog Items, PBIs) o características del producto a construir, mantenido y priorizado por el Product Owner. Es importante que exista una clara priorización, ya que es esta priorización la que determinará el orden en el que el equipo de desarrollo transformará las características (ítems) en un producto funcional acabado.

Esta prioridad es responsabilidad exclusiva del Product Owner y, aunque el equipo de desarrollo pueda hacer sugerencias o recomendaciones, es el Product Owner quien tiene la última palabra sobre la prioridad final de los ítems del Product Backlog, teniendo en cuenta el contexto de negocio, el producto mismo y el mercado en el que está inserto.

Priorización por valor de negocio de cada PBI

Una forma de priorizar los ítems del Product Backlog es según su valor de negocio. Podemos entender el valor de negocio como la relevancia que un ítem tiene para el cumplimiento del objetivo de negocio que estamos buscando.

Si planteáramos un ejemplo que ilustre el valor de negocio de los PBIs podríamos decir: en un proyecto cuyo objetivo es aumentar la afluencia de alumnos y facilitar la comunicación de los contenidos de las diferentes carreras de una universidad, se ha decidido crear un sitio web con diferentes características que se encuentran listadas en el Product Backlog. Dos de ellas son 1) que el alumno pueda acceder a los programas de estudios de las diferentes carreras y sus contenidos y 2) que el alumno pueda efectuar el pago en línea de su matrícula y cuotas utilizando una tarjeta de crédito.

En esta situación, muchos podríamos pensar que el requerimiento que implica el pago online con tarjeta de crédito representará un mayor valor de negocio que darle acceso a los alumnos a los contenidos de los programas de estudio, cuando la realidad es a la inversa: 1) el hecho de que un alumno pueda acceder a los contenidos de los programas de las diferentes carreras aporta un mayor valor hacia el cumplimiento del objetivo del producto (aumentar la afluencia de alumnos e incrementar a comunicación de los programas) que lo que el pago online podría hacer y 2) un alumno podría seguir abonando con tarjeta de crédito telefónicamente.

Priorización por retorno de la inversión (return on investment, ROI) de cada PBI

Un enfoque diferente de medir la prioridad de un determinado ítem del Backlog es calcular el beneficio económico que se obtedrá en función de la inversión que se deba realizar. Esto, si bien es una simple fórmula matemática, tiene implícita la problemática de encontrar o conocer el valor económico ganado por la incorporación de una determinada característica a un producto. Una vez identificada, el cálculo es relativamente simple:

ROI = valor de negocio/costo

Donde el costo representa el esfuerzo necesario para la construcción de una determinada característica de un producto y el valor de negocio es el rédito económico obtenido por su incorporación.

Prioridades según la importancia y el riesgo de cada PBI

Ya sea que los ítems del Backlog se prioricen por valor de negocio o por ROI, en cualquier caso llamémosle "priorizar por importancia", éstos pueden verse complementariamente afectados por el nivel de riesgo asociado a cada uno de ellos.

De esta manera, deberíamos aprovechar la construcción iterativa y evolutiva de Scrum para mitigar riesgos en forma implícita: construyendo primero aquellas características con mayor riesgo asociado y dejando las que poseen menor riesgo para etapas posteriores.

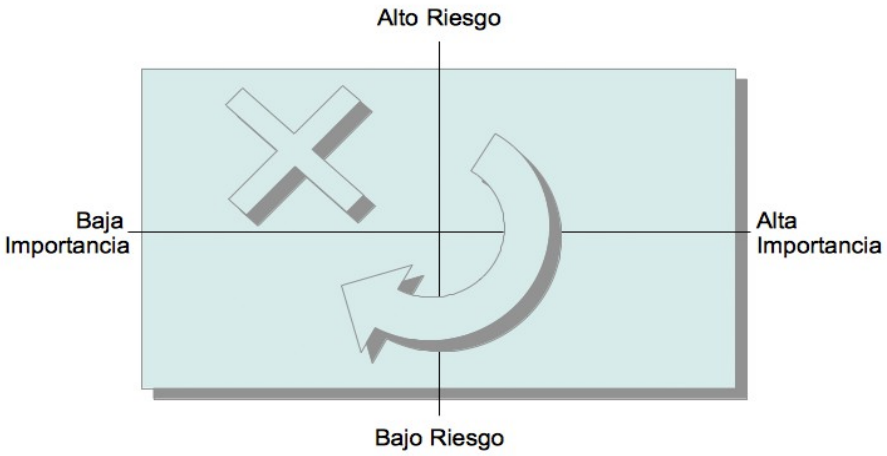


Figura 1: priorización ponderada por riesgos

Se recomienda que los PBIs de baja importancia y alto riesgo sean evitados, por ejemplo, transfiriéndolos o eliminándolos del alcance.

Backlog de Impedimentos

Como hemos comentado al describir el rol del ScrumMaster, una de sus principales responsabilidades es asegurar la remoción de impedimentos mediante facilitación o acción concreta. El Backlog de Impedimentos es un elemento opcional que puede ayudar al ScrumMaster y al equipo de desarrollo a llevar un registro de impedimentos ordenados por prioridad.

Si bien el Backlog de Impedimentos se debería mantener actualizado permanentemente, es especialmente utilizado en la Daily Standup Meeting, que se describirá en la próxima sección.

Dinámica (flujo del trabajo)

Antes de describir en detalle la dinámica de Scrum, recordemos el mecanismo de timeboxing[25] promovido por Scrum y los principios de ritmo sostenible, entrega frecuente de software valioso y adaptación constante que encontramos en el Manifiesto Ágil[26]. La razón es que en conjunto constituyen la piedra angular de la dinámica de Scrum: aprendizaje, inspección y adaptación.

Iteraciones (Sprints)

Las iteraciones en Scrum se conocen como Sprints. Scrum, como todos los enfoques ágiles, es un proceso de desarrollo incremental e iterativo. Esto significa que el producto se construye en incrementos funcionales entregados en periodos cortos para obtener feedback frecuente.

En general, Scrum recomienda una duración de Sprint de entre 1 y 4 semanas, siendo 2 o 3 semanas lo más habitual que encontraremos en la industria. Una de las decisiones que debemos tomar al comenzar un proyecto o al adoptar Scrum es justamente la duración de los Sprints. Luego, el objetivo será mantener esta duración constante a lo largo del desarrollo del producto, lo que implicará que la duración de una iteración no cambie una vez que sea establecida.

Como excepción podemos mencionar aquellas situaciones donde el equipo mismo decida probar con iteraciones más largas o más cortas. Esta decisión se basa principalmente en la volatilidad del contexto: mientras más volátil sea (negocio cambiante, requerimientos desconocidos, etc.) más corta será la duración del Sprint. Lo importante es recordar que se logra mayor ritmo y previsibilidad teniendo Sprints de duración constante.

Retrasos y adelantos en un Sprint

Muchas veces podremos encontrar situaciones en donde el equipo de desarrollo se atrase o se adelante. En estos casos, la regla del "timeboxing" no nos permitirá modificar (adelantar o postergar) la fecha de entrega o finalización del Sprint. La variable de ajuste en estos casos será el alcance del Sprint, esto es, en el caso de adelantarnos deberemos incrementar el alcance del Sprint agregando nuevos PBIs y reducirlo en el caso de retrasarnos.

Incremento funcional potencialmente entregable

El resultado de cada Sprint debe ser un incremento funcional potencialmente entregable.

Incremento funcional porque es una característica funcional nueva (o modificada) de un producto que está siendo construido de manera evolutiva. El producto crece con cada Sprint.

Potencialmente entregable porque cada una de estas características se encuentra lo suficientemente validada y verificada como para poder ser desplegada en producción (o entregada a usuarios finales) si así el negocio lo permite o el cliente lo desea.

25 Ver página 21.
26 Ver página 8.

Sprint Planning Meeting (Reunión de Planificación de Sprint)

Al comienzo de cada Sprint se realiza una reunión de planificación del Sprint donde serán generados los acuerdos y compromisos entre el equipo de desarrollo y el Product Owner sobre el alcance del Sprint.

Esta reunión de planificación habitualmente se divide en dos partes con finalidades diferentes: una primera parte estratégica y enfocada en el "qué", y una segunda parte táctica cuyo hilo conductor principal es el "cómo".

Parte estratégica: ¿qué vamos a hacer?

Podríamos decir que se trata de un taller donde el Product Owner expone todos y cada uno de los PBIs que podrían formar parte del Sprint, mientras que el equipo de desarrollo realiza todas las preguntas que crea necesarias para conocer sus detalles y así corroborar o ajustar sus estimaciones.

Aún asumiendo que los PBIs ya han sido estimados con anterioridad[27], debido al principio de "aceptar los cambios aun en etapas avanzadas del proyecto", es posible que en esta reunión aparezcan PBIs que no habían sido estimados anteriormente. Frente a esta situación, el equipo de desarrollo indagará y estimará esos PBIs de inmediato.

El objetivo buscado durante esta parte de la reunión es identificar "qué" es lo que el equipo de desarrollo va a realizar durante el Sprint, es decir, todos aquellos PBIs que el equipo se comprometerá a transformar en un producto funcionando y utilizable o en otras palabras: incremento funcional potencialmente entregable.

El Product Owner y el equipo de desarrollo deben participar de esta parte de la reunión como protagonistas principales. El ScrumMaster, al tiempo que facilita la reunión, también debe asegurar que cualquier stakeholder del proyecto que sea requerido para profundizar en detalles esté presente o sea contactado.

El equipo de desarrollo utiliza su capacidad productiva (también conocida como Velocidad o *Velocity*), obtenida de los Sprints pasados, para conocer hasta cuánto trabajo podría comprometerse a realizar. Esto determinaría en un principio cuáles serían los PBIs comprometidos en este Sprint.

Como se ha visto, hemos hablado en potencial*:* el equipo de desarrollo "podría", esto "determinaría". La razón es que cada uno de los ítems del Product Backlog debe ser discutido para entender cuáles son sus criterios de aceptación y así conocer en detalle qué se esperará de cada uno. De esta manera, el equipo de desarrollo discutirá con el Product Owner sobre cada PBI y generará un compromiso de entrega para aquellos que considera suficientemente claros como para comenzar a trabajar y que además podrían formar parte del alcance del Sprint que está comenzado. A esto se lo conoce como planificación basada en compromisos o Commitment-based Planning.

Al finalizar esta primera parte de la reunión, tanto el Product Owner como los stakeholders involucrados (si los hubiese) se retirarán, dejando así al ScrumMaster y al equipo de desarrollo para que den comienzo a la segunda parte de esta reunión, que se describe a continuación.

27 La mecánica de esta estimación será explicada más adelante.

Parte táctica: ¿cómo lo vamos a hacer?

Durante este espacio de tiempo el equipo de desarrollo determinará la forma en la que llevará adelante el trabajo. Esto implica la definición inicial de un diseño de alto nivel, el cual será refinado durante el Sprint mismo y la identificación de las actividades que el equipo en su conjunto tendrá que llevar a cabo.

Se espera que el diseño sea emergente, es decir, que surja de la necesidad del equipo de desarrollo a medida que éste avance en el conocimiento del negocio. Por esta misma razón es que indicamos la no necesidad de realizar un diseño completo y acabado de lo que será realizado durante el Sprint. En cambio, se buscará un acuerdo de alto nivel que será bajado a detalle durante la ejecución de la iteración.

Esto mismo sucede con las actividades del Sprint, es decir que no es estrictamente necesario enumerar por completo todas las actividades que serán realizadas durante la iteración ya que muchas aparecerán a medida que avancemos. Recordemos que a esta altura los PBIs ya han sido estimados y el surgimiento de actividades durante el Sprint no habilita a incrementar las estimaciones de los PBIs, salvo excepciones donde la estimación inicial no había considerado la totalidad del esfuerzo necesario. Adicionalmente, es recomendable que las actividades duren idealmente menos de un día. Esto permitirá detectar bloqueos o retrasos durante las reuniones diarias (ver siguiente).

Si bien el Product Owner no participa de esta reunión, debería ser contactado en el caso de que el equipo de desarrollo necesite respuestas a nuevas preguntas con la finalidad de clarificar su entendimiento de las necesidades.

Al finalizar esta reunión, el equipo habrá arribado a un Sprint Backlog o Commited Backlog que representa el alcance del Sprint en cuestión. Este Sprint Backlog es el que se coloca en el taskboard (pizarra de actividades) del equipo. Se dará comienzo al desarrollo del producto para este Sprint.

Reuniones diarias

Uno de los beneficios de Scrum está dado por el **incremento de la comunicación** dentro del equipo de proyecto. Esto facilita la coordinación de acciones entre los miembros del equipo de desarrollo y el conocimiento “en vivo” de las dependencias de las actividades que realizan.

Por otro lado, se requiere además **aumentar y explicitar los compromisos** asumidos entre los miembros del equipo de desarrollo y **dar visibilidad a los impedimentos** que surjan del trabajo qu está siendo realizando y que muchas veces nos impiden lograr los objetivos.

Estos tres objetivos: 1) incrementar la comunicación 2) explicitar los compromisos y 3) dar visibilidad a los impedimentos, son logrados mediante las reuniones diarias o Daily Standup Meetings. Estas reuniones tienen, como su nombre lo indica, una frecuencia diaria y no deberían llevar más de 15 minutos. Estos 15 minutos son un timebox, es decir, que no se pueden superar.

A la reunión diaria acude el ScrumMaster y el equipo de trabajo. En el caso de que sea necesario, se podrá requerir la presencia del Product Owner y de los stakeholders. De todas maneras, se intenta que sea una reunión abierta donde cualquier interesado en escuchar lo que sucede pueda participar en calidad de observador. Se recomienda que los observadores no participen activamente en la reunión, y mucho menos, que soliciten a los miembros del

equipo justificación del progreso y explicación de los problemas.

Esta reunión es facilitada por el ScrumMaster. Todos y cada uno de los miembros toman turnos para responder las siguientes tres preguntas, y de esa manera comunicarse entre ellos:

1. ¿Qué hice desde la última reunión diaria hasta ahora?
2. ¿En qué voy a estar trabajando desde ahora hasta la próxima reunión diaria?
3. ¿Qué problemas o impedimentos tengo?

Es importante destacar que en ningún momento se trata de una reunión de reporte de avance o status al ScrumMaster ni a otras personas. Por el contrario, es un espacio de estricta comunicación entre los miembros del equipo de desarrollo.

El objetivo de la primera pregunta (¿qué hice...?) es verificar el cumplimiento de los compromisos contraídos por los miembros del equipo en función del cumplimiento del objetivo del Sprint. La finalidad de la segunda pregunta (¿qué voy a hacer...?) es generar nuevos compromisos hacia el futuro. Cuando hablamos de compromisos, hacemos referencia a aquéllos que los los miembros del equipo asumen ante sus compañeros.

La última pregunta (¿qué problemas...?) apunta a detectar y dar visibilidad a los impedimentos. Estos impedimentos no se resuelven en esta reunión, sino en posteriores. Es responsabilidad del ScrumMaster que se resuelvan lo antes posible, generando las reuniones que sean necesarias e involucrando a las personas correctas.

En el caso de que los PBIs del Sprint se hubiesen podido dividir en actividades de menos de un día: si una de estas actividades se encuentra en progreso durante dos reuniones diarias seguidas (con 24hs de separación) claramente se advierte un retraso.

Reunión de revisión del producto

Al finalizar cada Sprint se realiza una reunión de revisión (review/demo) del incremento funcional potencialmente entregable construido por el equipo de desarrollo (el "qué"). En esta reunión el equipo expondrá el resultado del Sprint frente al Product Owner. Cuando decimos "resultado" hablamos de "producto utilizable" y "potencialmente entregable" que el Product Owner utilizará y evaluará durante esta misma reunión, aceptando o rechazando así las funcionalidades construidas.

El Product Owner evalúa en tiempo real las funcionalidades construidas y provee su feedback. Los stakeholders del proyecto también pueden participar en esta reunión para aportar sus impresiones, que pueden ser acerca de cambios en la funcionalidad construida o bien nuevas funcionalidades que surjan de ver el producto en acción.

Toda la retroalimentación que el Product Owner y los stakeholders aporten debe ser ingresada como PBIs en el Product Backlog. Para esto, los PBIs nuevos deben ser priorizados con respecto a todos los ya existentes en el Product Backlog. También es necesario que estos nuevos PBIs sean estimados antes de incluirlos como parte del Product Backlog ya que el Product Owner deberá decidir cuáles de los PBIs existentes cuya estimación de costo es similar a la de los nuevos PBIs deben ser eliminados para no incurrir en el incremento desmedido del alcance (Scope Creeping): si se agrega trabajo entonces debemos quitar trabajo de otro lado. El Product Owner cuenta para esto con la priorización de los ítems del Backlog como herramienta para la toma de este tipo de decisiones.

En el caso de que una funcionalidad sea rechazada, el PBI correspondiente reingresa al Product Backlog con máxima prioridad, para ser tratado en el siguiente Sprint. La única excepción a esta regla es que el Product Owner, por decisión propia, prefiera dar mayor prioridad a otros. En este caso, nada debe salir del Backlog ya que esto no sería considerado como un incremento en el alcance.

Al finalizar la revisión del producto, es recomendable definir la fecha de la próxima reunión de revisión, que corresponderá al final del Sprint siguiente. De este modo ya se tendrán las agendas bloqueadas a tal fin.

Reunión de retrospectiva

En un método empírico como Scrum, la retrospección del equipo es el corazón de la mejora continua. Mediante el mecanismo de retrospección, el equipo reflexiona sobre la forma en la que realizó su trabajo y los acontecimientos que sucedieron en el Sprint que acaba de concluir para mejorar sus prácticas. Todo esto sucede durante la reunión de retrospectiva.

Esta reunión tiene lugar inmediatamente después de la reunión de revisión. Mientras que la reunión de revisión se destina a revisar el producto (el “qué”), la retrospectiva se centra en el proceso (el “cómo”).

Este tipo de actividad necesita un ambiente seguro donde el equipo de desarrollo pueda expresarse libremente, sin censura ni temores, por lo cual se restringe solo al equipo de desarrollo y al ScrumMaster. En el caso de que se requiera la participación del Product Owner, stakeholders o gerentes, estos podrán ser convocados.

Valiéndose de técnicas de facilitación y análisis de causas raíces, se buscan tanto fortalezas como oportunidades de mejora. Luego, el equipo de desarrollo decide por consenso cuáles serán las acciones de mejora a llevar a cabo el el siguiente Sprint. Estas acciones y sus impactos se revisarán en la próxima reunión de retrospectiva.

Reunión de refinamiento del Backlog

La reunión de refinamiento del Backlog se realiza durante el Sprint y en función de las necesidades. Su objetivo es profundizar en el entendimiento de los PBIs que se encuentran más allá del Sprint actual y así dividirlos en PBIs más pequeños, si lo requieren, y estimarlos. Idealmente se revisan y detallan aquellos que potencialmente se encuentren involucrados en los próximos dos o tres Sprints.

Otro objetivo importante que se debe perseguir en esta reunión es la detección de riesgos implícitos en los PBIs que se estén analizando, y en función de ellos revisar y ajustar las prioridades del Product Backlog.

La participación de todo el equipo de desarrollo y del Product Owner es esencial para el éxito de esta reunión. Sin ellos la reunión no tendría sentido. La responsabilidad de convocarla es del Product Owner y se realiza entre una y dos veces por Sprint, facilitadas por el ScrumMaster.

Acerca de Kleer

Somos una empresa de capacitación y coaching ágil donde creemos en una forma de trabajo clara, centrada en las personas y orientada hacia las necesidades específicas de cada contexto.

Nuestras premisas son:

- **Mantenerlo simple**. Las metodologías y prácticas de trabajo en las que confiamos aportan claridad a los proyectos. Los proyectos claros se vuelven más previsibles y con menor incidencia de errores evitables.
- **Las personas son todo**. No acompañamos proyectos sino equipos y personas. Nuestro propósito es transmitirles nuestro conocimiento y experiencia para que logren resultados de los que se sientan orgullosos.
- **Cada contexto es un mundo**. Estudiamos las características de cada proyecto u organización para entender cuáles son las prácticas y metodologías que mejor responden a sus necesidades y objetivos de negocio.

Contacto: entrenamos@kleer.la | http://www.kleer.la

Acerca del Autor

Martín Alaimo

Certified Scrum Coach (CSC) y entrenador en Metodologías Ágiles, con más de 15 años de experiencia en proyectos de desarrollo de software. Martín acompaña a diferentes empresas, muchas de primera línea, en su camino de adopción de Scrum y Agilidad. También dicta entrenamientos sobre Scrum, tanto desde una perspectiva de gestión y liderazgo como de un punto de vista de ingeniería y desarrollo de software.

Comenzó su carrera en la industria de desarrollo de software hace más de 15 años como desarrollador de software centrado siempre en implementaciones web, convirtiéndose con el tiempo en gerente de proyecto de Java / .NET / RoR y tecnologías de SAP, con varios años de experiencia liderando proyectos de software en organizaciones multinacionales.

Estudió Analista de Sistemas Informáticos y se especializó en Gestión de Proyectos, principalmente Agiles, obteniendo certificaciones como PMP (PMI), PMI-ACP, Certified ScrumMaster, Certified Scrum Professional y Certified Scrum Coach.

Blog:http://www.martinalaimo.com/es/blog/

Contacto: martin.alaimo@kleer.la

Esta obra fue realizada por Martín Alaimo para KLEER, con aportes de Pablo Tortorella y Daniela Casquero y se encuentra bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported.