Unidad 2 / Escenario 3

Lectura fundamental

Metodologías ágiles

Contenido

Palabras clave: *Software*, procesos, modelos, desarrollo de *software*, ingeniería del *software*, procesos del *software*, cascada, espiral

1. Metodologías de desarrollo de *software*

Antes de hablar de las metodologías ágiles de desarrollo de *software*, se verá cuáles son las características de las metodologías de desarrollo de *software* y en qué difieren de los modelos de procesos de *software*. Como se ha visto, un modelo de procesos de *software* define las actividades que se van a desarrollar para construir un *software*, las metodologías por su parte brindan métodos y herramientas específicas para desarrollar esas actividades cuidando el proceso. Dicho de otra manera, el modelo de procesos se enfoca en definir qué actividades desarrollar y en qué orden, mientras que las metodologías ofrecen herramientas que orientan el cómo desarrollar y gestionar estas actividades.

Para cumplir con su propósito una metodología de desarrollo de *software* debe cumplir con unas características fundamentales:

- Debe ajustarse a los objetivos y contexto del proyecto a realizar.
- Debe cubrir todas las fases del ciclo del proceso de desarrollo de *software*. Como hemos visto hasta ahora cada etapa tiene un objetivo específico que se debe satisfacer para garantizar que el producto de *software* obtenido es un producto de calidad.
- Debe además facilitar la integración de las distintas fases del ciclo de desarrollo permitiendo.
    - Rastreabilidad: es decir que es posible moverse no solo hacia adelante en el ciclo de vida, sino hacia atrás de forma que se pueda comprobar el trabajo realizado y se puedan efectuar correcciones.
    - Fácil interacción entre etapas del ciclo de desarrollo: es necesaria una validación formal de cada fase antes de pasar a la siguiente. La información que se pierde en una fase determinada queda perdida para siempre, con un impacto en el sistema resultante.
- Debe facilitar el entendimiento del problema a desarrollar por parte de todos los interesados, de manera que la metodología de alguna manera es también una herramienta que facilita la comunicación entre todos los involucrados. Esto significa que las herramientas de especificación y análisis debe ser amigables y sencillas.
- La metodología debe ser la base de una comunicación efectiva, no solo entre el analista y el usuario sino en todos los implicados en el proyecto, es decir, no sólo en la determinación de los requisitos sino en la comunicación de los avances y decisiones tomadas en la ejecución del proyecto.

- La clave del éxito es que todas las partes implicadas han de intercambiar información libremente.
- La metodología debe poder emplearse en un entorno amplio de proyectos de *software*, es decir que:
  - Una empresa deberá adoptar una metodología que sea útil para un gran número de sistemas que vaya a construir. Por esta razón no es práctico adoptar varias metodologías en una misma empresa. Sin embargo, si es posible definir una metodología propia que tome las ventajas de otras metodologías, siempre y cuando se cumplan las características que se están describiendo.
  - Las metodologías deberán ser capaces de abordar sistemas de distintos tamaños y rangos de vida.
  - La metodología debe servir para sistemas de distinta complejidad, es decir puede abarcar un departamento, varios departamentos o varias empresas.
  - Entorno, la metodología debe servir con independencia de la tecnología disponible en la empresa.

2. Metodologías tradicionales versus metodologías ágiles

Los ingenieros firmantes del manifiesto ágil, los cuáles se pueden consultar el página oficial del manifiesto ***http://agilemanifesto.org***, en la sección: Ver firmantes, consideran que las metodologías de desarrollo de *software* conocidas como tradicionales, son rígidas y burocráticas, es decir, que se invierte demasiado tiempo en el desarrollo de actividades que no ofrecen verdadero valor. Estas personas consideran que:

- Las metodologías tradicionales ponen mucho énfasis sobre el plan de proyecto. En tenerlo todo bien especificado antes de comenzar, en seguir fielmente el camino planificado y en documentar exhaustivamente todo lo realizado.
- Las metodologías ágiles, por el contrario, ponen el énfasis en entregar buen código al cliente. En obtener resultados que satisfagan al cliente, adaptándose a sus necesidades cambiantes.

Cabe aclarar que esta es la justificación de los firmantes del manifiesto, para quienes, desde su mirada crítica, en las metodologías tradicionales el marcado énfasis en la documentación y en el plan del

proyecto está en detrimento de obtener atención a la calidad del código y las necesidades del cliente. Estas son las razones que ellos postulan para proponer la alianza ágil. Por lo anterior, en 2001, Kent Beck lideró un grupo conocido como "La alianza ágil" quienes firmaron el "Manifiesto por el desarrollo ágil de *software*".

En el que se establece como premisas valorar:

- "Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto *software*. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que este configure su propio entorno de desarrollo en base a sus necesidades.
- Desarrollar *software* que funciona, más que conseguir una buena documentación. La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental.
- La colaboración con el cliente más que la negociación de un contrato. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- Responder a los cambios más que seguir estrictamente un plan. 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 el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta." (Manifiesto para el Desarrollo ágil de Software, 2001).

2.1. Doce principios del Manifiesto ágil

- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de *software* que le aporte un valor.
- Dar la bienvenida a los cambios: se capturan los cambios para que el cliente tenga una ventaja competitiva.
- Entregar frecuentemente *software* que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

- La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
- Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
- El *software* que funciona es la medida principal de progreso.
- Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.
- Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
- En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

La intención de los defensores del desarrollo de *software* ágil es que "el desarrollo ágil se centre en los talentos y habilidades de los individuos, y adapta el proceso a personas y equipos específicos." Es decir, que el proceso se adapta a las necesidades de las personas y del equipo, no al revés.

Por esta razón, los miembros de un equipo de *software* deben compartir las siguientes características clave:

- Competencia: en un contexto de desarrollo ágil (así como en la ingeniería de *software*), la "competencia" incluye el talento innato, las habilidades específicas relacionadas con el *software* y el conocimiento general del proceso que el equipo haya elegido aplicar. La habilidad y el conocimiento del proceso pueden y deben considerarse para todas las personas que sean miembros ágiles del equipo.
- Enfoque común: aunque los miembros del equipo ágil realicen diferentes tareas y aporten habilidades distintas al proyecto, todos deben centrarse en una meta: entregar al cliente en la fecha prometida un incremento de *software* que funcione. Para lograrlo, el equipo también se centrará en adaptaciones continuas (pequeñas y grandes) que hagan que el proceso se ajuste a las necesidades del equipo.

- Colaboración. La ingeniería de *software* (sin importar el proceso) trata de evaluar, analizar y usar la información que se comunica al equipo de *software*; crear información que ayudará a todos los participantes a entender el trabajo del equipo; y generar información (*software* de cómputo y bases de datos relevantes) que aporten al cliente valor del negocio. Para efectuar estas tareas, los miembros del equipo deben colaborar, entre sí y con todos los participantes.
- Habilidad para tomar decisiones. Cualquier equipo bueno de *software* (incluso los equipos ágiles) debe tener libertad para controlar su destino. Esto implica que se de autonomía al equipo (autoridad para tomar decisiones sobre asuntos tanto técnicos como del proyecto).
- Capacidad para resolver problemas difusos. Los gerentes de *software* deben reconocer que el equipo ágil tendrá que tratar en forma continua con la ambigüedad y que será sacudido de manera permanente por el cambio. En ciertos casos, el equipo debe aceptar el hecho de que el problema que resuelven ahora tal vez no sea el que se necesite resolver mañana. Sin embargo, las lecciones aprendidas de cualquier actividad relacionada con la solución de problemas (incluso aquellas que resuelven el problema equivocado) serán benéficas para el equipo en una etapa posterior del proyecto.
- Confianza y respeto mutuos. El equipo ágil debe tener tanta y confianza para lograr "un tejido tan fuerte que el todo sea más que la suma de sus partes"
- Organización propia. En el contexto del desarrollo ágil, la organización propia implica tres cosas: 1) el equipo ágil se organiza a sí mismo para hacer el trabajo, 2) el equipo organiza el proceso que se adapte mejor a su ambiente local, 3) el equipo organiza la programación del trabajo a fin de que se logre del mejor modo posible la entrega del incremento

A continuación, verá las características de las principales metodologías ágiles según (Orjuela, 2008, pp. 159-171)

2.2. Programación Extrema (*Extreme Programming, XP*)

Se basa en:

- Retroalimentación continua entre el cliente y el equipo de desarrollo,
- Comunicación fluida entre todos los participantes,

- Simplicidad en las soluciones implementadas y agilidad para enfrentar los cambios.
- XP se define como especialmente adecuada para proyectos con requisitos imprecisos y cambiantes, y donde existe un alto riesgo técnico.

2.3. Crystal Methodologies

Se trata de un conjunto de metodologías para el desarrollo de *software* caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos.

El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo, *Crystal Clear* (3 a 8 miembros) y *Crystal Orange* (25 a 50 miembros).

2.4. Dynamic Systems Development Method (DSDM)

Es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos.

Propone cinco fases:

1. Estudio de viabilidad: se elabora un informe en el que se evalúa la conveniencia de desarrollar el proyecto.
2. Estudio del negocio: en esta fase se identifican las características esenciales del negocio y de la tecnología listando y priorizando los requerimientos del negocio.
3. Modelado funcional: se determinan las principales funcionalidades a través de un prototipo que debe ser aprobado para continuar con la siguiente fase.
4. Diseño y construcción: el prototipo funcional de la fase anterior se transforma en un prototipo de diseño más detallado que se debe implementar o construir.
5. Implantación: se verifica que el sistema cumple con lo deseado y se hace el entrenamiento de los usuarios.

Las tres últimas son iterativas, además de existir retroalimentación a todas las fases.

2.5. *Adaptive Software Development* (ASD)

Es un proceso Iterativo, orientado a los componentes *software* más que a las tareas y tolerante a los cambios.

El ciclo de vida que propone tiene tres fases esenciales:

1. Especulación, inicia el proyecto y se planifica las características del *software*.
2. Colaboración, se desarrollan las características del *software*
3. Aprendizaje, se revisa la calidad del *software* y se entrega al cliente.

2.6. *TSP Team Software Process*

Provee un equilibrio entre el proceso, el producto y la gestión del equipo de trabajo.

Se basa en los siguientes principios:

- Aprender es más efectivo que seguir procesos definidos, se basa en la evaluación y retroalimentación constante de los resultados de cada iteración.
- El éxito del equipo se basa en la definición de objetivos claros, liderazgo y entrenamiento constante.

2.7. SCRUM

- Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos años. Está especialmente indicada para proyectos con un rápido cambio de requisitos.
- El desarrollo de *software* se realiza mediante iteraciones, denominadas *sprints*, con una duración de alrededor de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente.

- La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Este es la metodología ágil más popular en este momento, no sólo en la industria del *software*, sino en cualquier escenario en el cual se desarrollen proyectos en los que se considera que las condiciones del proyecto pueden variar rápida y frecuentemente y en que se puede generar una dinámica más ágil de trabajo en equipo.

- Para mejorar puede optar por el curso y examen de certificación en SCRUM para lo cual puede seguir la guía oficial la cual encuentra en el siguiente enlace: crumguide.org

Los tres conceptos fundamentales que propone SCRUM son roles, artefactos y actividades. En la siguiente tabla se muestra un resumen de estos conceptos.

Tabla 1. Roles, Artefactos y Actividades de SCRUM

Roles	Artefactos	Actividades
Scrum team (ST) formado por: • *Product owner* (PO) • *Scrum master* (SM) • *Development team* (DT) • *Stakeholders*	Del proyecto • *Product backlog* (PB) • *Sprint backlog* (SB) • *Graphs* • *Impediments backlog* (IB)	Del *sprint* • *Scrum board* • *Incidence backlog* • *Parking backlog* • *Sprint 0 o first sprint* • *Sprint* • *Sprint planning* (planificación del *sprint*) • *Daily meeting* (reunión diaria) • *Sprint review* (revisión del *sprint*) • *Sprint retrospective* (retrospectiva del *sprint*)

Fuente: elaboración propia basada en Monte (2016)

En la siguiente tabla se presentan las características fundamentales de los diferentes roles de acuerdo con Monte (2016).

Tabla 2. Roles de *SCRUM*

Rol	Funciones	Responsabilidad
***Product owner* (PO)** El *product owner* es el enlace entre el cliente y el equipo de desarrollo. Debe conocer el negocio; ha de saber de qué se habla en el ámbito funcional; por qué una historia del *product backlog* es más importante que otra; y por qué la historia dice lo que dice.	• Definir la estrategia • Definir los objetivos • Mantener el *product backlog* • Negociar el alcance con el cliente • Definir, junto con el *Scrum* master, los criterios de aceptación del proyecto y de cada *sprint* • Mantener el presupuesto • Participar en los *sprint* reviews • Ayudar al SM y al DT a resolver cualquier cuestión en lo referente al proyecto, la funcionalidad y los productos.	Evaluación del curso del proyecto
Scrum master (SM) Generalmente se relaciona con el rol del project manager, pero no es su equivalente, puede ser cualquier miembro del development team. El SM y el PO no pueden ser la misma persona.	• Es un mentor para los componentes del *development team* (DT). • Es quien proporciona suporte al DT y ayuda a resolver los problemas. • Es el enlace entre el DT y el P O. • Es quien reporta, archiva y lleva registro.	• De la gestión, junto con el D T, del curso del *sprint*, mediante el *Scrum* board. • De la evaluación del grado de avance y éxito del *sprint*, mediante el gráfico *sprint* burn-down.

	• Es quien propone, promueve y potencia mejoras sobre el proceso y sobre el *Scrum team*. • De la gestión, junto con el DT, del alcance del *sprint* actual, mediante el *sprint backlog*.	• De la gestión de los problemas del *sprint* y del equipo, y de buscar soluciones mediante el incidence *backlog* y el *impediments backlog*. • De la promoción de la mejora continua, mediante la reunión de *Scrum retrospective*.
***Development team* (DT)** El *development team* se caracteriza por ser flexible, autoorganizado y multidisciplinario. Se recomienda que este conformado entre tres y nueve integrantes.		• De la estimación del esfuerzo, tanto de la funcionalidad descrita al *product backlog* como de cada tarea del *sprint*. • De la gestión del *sprint backlog*. • De la determinación del detalle de cada funcionalidad descrita en el *product backlog* y la subdivisión en tareas. • De la entrega del producto acabado y convenientemente depurado, siguiendo los criterios de aceptación marcados. • De la ejecución diaria del *daily meeting* y de cumplir las normas de esta actividad.

Stakeholder Los *stakeholder* son los receptores del producto acabado y, por lo tanto, son quienes hacen la aceptación. Para hacer esto, están obligados a asistir a los *sprint reviews*.		• De definir los criterios de aceptación de cada funcionalidad del *product backlog* y de proporcionar y resolver las dudas que le presente el PO, que puede acudir tanto por cuenta propia como por encargos del DT.

Fuente: elaboración propia

En la siguiente tabla se presentan las características fundamentales de los artefactos de acuerdo con Monte (2016).

Tabla 3. Artefactos de SCRUM

Artefacto	Descripción
***Product backlog* (PB)**	El *product backlog* es la lista de funcionalidades, productos o acciones que conforman el producto que se ha de construir. El *product backlog* se escribe en «el idioma» del cliente y se compone de user stories (historias de usuario) Cada historia se va completando y detallando a medida que se necesita o debe información. Ha de tener suficiente información para permitir que el equipo pueda hacer una estimación de lo que costaría hacerla realidad. El PO es responsable de mantener la lista, y de encargarse de que los usuarios proporcionan suficiente información útil y de priorizarla.
***Sprint backlog* (SB)**	El *sprint backlog* es la lista de funcionalidades extraídas del product *backlog* que se incorporan al *sprint* en curso. Como cada funcionalidad está tasada en un valor de *story points*, el PO, en función de la velocidad del equipo (*team velocity*) puede asignar las funcionalidades más prioritarias que cubran la capacidad de trabajo del DT en el *sprint*.

Graph Release burn-down	El *release burn-down* es responsabilidad del PO. Con este gráfico el PO puede ver la evolución del proyecto y tomar medidas a cada *sprint* review. El gráfico presenta dos métricas: la evolución del proyecto y la velocidad del equipo. La línea correspondiente a la velocidad del equipo muestra el número de story points que el equipo ha resuelto en cada *sprint*. Este gráfico muestra en el eje X los *sprints* y en el eje Y, los *story points* resueltos en cada momento.
Graph Sprint burn-down	El *sprint burn-down* es responsabilidad del SM. Este gráfico se mantiene actualizado gracias a la consulta diaria del *Scrum board*. Y permite al SM conocer la evolución del *sprint* en curso. El gráfico presenta dos métricas: la previsión, que corresponde a la valoración de horas que se hizo para cada tarea, y la evolución real, que corresponde al consumo de horas real de cada tarea. Este gráfico presenta la suma de horas en tareas que se van sacando del estado pendiente. Muestra en el eje X los días de que se compone el *sprint*, y en el eje Y, las horas invertidas en las tareas que se consiguen resolver cada día.
Impediments backlog e incidence backlog (IB)	La *impediments backlog* es una lista de problemas que sirve como registro para que el *Scrum master* pueda buscar soluciones. La *incidence backlog* es una lista de problemas referentes a las tareas de un *sprint*. Cualquier cambio no previsto sobre una tarea se anota en esta lista para que pueda ser tratado en la reunión de *sprint retrospective.* Los problemas que se incluyen en el *impediments backlog* se pueden catalogar como de «difícil solución», puesto que explican situaciones que sobrepasan la capacidad de resolución del propio equipo. Por ejemplo, un problema de infraestructura grave que impide la adopción de una solución viable a una funcionalidad concreta. En este escenario, el equipo puede tomar nota del problema, registrarlo, para exponer de forma justificada donde corresponda el motivo del retraso de una tarea. La *incidence backlog*, en cambio, registra problemas que pueden ser resueltos por el equipo, como por ejemplo, cambios en la estimación, problemas técnicos o problemas de coordinación (cuellos de botella) en el tratamiento de la tarea.

Parking backlog	El *parking backlog* es la lista de tareas que se encuentran paradas en un *sprint* al detectarse algún problema que impide acabarlas, por ejemplo, porque se espera un resultado intermedio. Usualmente, cuando un impediment explicado en la sección anterior no se resuelve y la tarea queda parada, dicha tarea pasa a estar en la lista de *parking*. El *sprint* no puede considerarse acabado sin que todas sus tareas estén acabadas. Por tanto, el *parking backlog* tiene que quedar vacío a la finalización de cada *sprint*, o bien hace falta negociar con el PO el traslado de las tareas pendientes a otro *sprint*, siempre que esto no afecte los criterios de aceptación del propio *sprint*.
Scrum board	El *Scrum board* es una herramienta opcional en el estándar *Scrum*, a pesar de que está ampliamente aceptado por la comunidad *Scrum* y es la principal (y para muchos, imprescindible) herramienta visual del estado del *sprint* para el *Scrum team*. Debe mostrar la situación del *sprint* en tiempo real y, por lo tanto, hace falta actualizarlo a cada daily meeting, pero también bajo demanda de cualquier componente del DT, siempre con la coordinación del SM. En el *Scrum board* se muestra toda la información del *sprint* en curso; es decir: • Las user stories del *sprint* colocadas por orden de prioridad (arriba las más prioritarias) para que el equipo pueda ver con un simple vistazo la importancia de cada tarea. • Las tareas de cada *user story* y su situación. • Las listas de incidencias e impedimentos y el *parking backlog*.

Fuente: elaboración propia

En síntesis...

el manifiesto ágil es un marco incluyente que define las premisas para el desarrollo de *software* ágil, no es una metodología en sí mismo. Cada metodología ágil implementa las premisas del manifiesto con enfoques o aproximaciones diferentes.

Referencias

Doce Principios del manifiesto ágil. (2001) Recuperado de: http://agilemanifesto.org/iso/es/principles.html

Jacobson, I., Booch,G., y Rumbaugh, J. (2000). *El proceso unificado de desarrollo de software*. Madrid, España: Adisson Wesley.

Manifesto for Agile Software Development. (2001) Recuperado de: http://agilemanifesto.org

Martínez, A., Martinez, Raul. (2000). *Guía a Rational Unified Process.*

Monte, J.L. (2016). *Implantar SCRUM con éxito*. Barcelona, España: Editorial UOC.

Orjuela Duarte, A., & Rojas C., M. (2008). *Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo*. Revista Avances en Sistemas e Informática, 5 (2), 159-171.

INFORMACIÓN TÉCNICA

FACULTAD DE
INGENIERÍA, DISEÑO
E INNOVACIÓN

Módulo: Ingeniería del *Software* I

Unidad 2: Metodologías de desarrollo de *software*

Escenario 3: Metodologías ágiles

Autor: Diana Angélica Cruz Ortega

Asesor Pedagógico: Luisa Esperanza Rincón Jiménez

Diseñador Gráfico: Cristian Navarro

Asistente: Ginna Quiroga