

A faded, grayscale background image of a person's face, likely a man, looking directly at the camera. The image is positioned on the left side of the slide, with the right side being a solid light gray.

Introducción a las metodologías ágiles

Introducción a las metodologías ágiles

Contenidos

- Introducción y objetivos
- Modelos de proceso y metodologías
- Agilidad y procesos ágiles
- Manifiesto ágil
- Algunas metodologías ágiles
- Ágil vs. tradicional
- Referencias

1. Introducción y objetivos

En este tema vamos a comenzar el estudio de las metodologías ágiles de desarrollo de software, que reciben este nombre genérico en oposición a las metodologías tradicionales, o «pesadas».

Metodologías tradicionales o «pesadas»: Siguen, en general, una serie de fases estrictamente definidas, donde la introducción de cambios durante el desarrollo del proyecto puede resultar muy costosa y que requieren de la generación de grandes volúmenes de documentación.

Objetivos:

- Repasar el concepto de modelo de proceso, su relación con las metodologías y la importancia que tiene seleccionar el modelo más adecuado para cada proyecto.
- Comprender el concepto de «agilidad» en el contexto del desarrollo de software, así como los principios del «Manifiesto ágil».

1. Introducción y objetivos

Objetivos:

- Conocer las principales características de las metodologías ágiles y sus ventajas con respecto a las metodologías tradicionales.
- Comprender los fundamentos de algunas de las metodologías ágiles más populares, como: XP, Scrum, Crystal o Feature-Driven Development.
- Saber valorar de manera crítica tanto las metodologías tradicionales como las ágiles.
- Reflexionar sobre la importancia de seleccionar la metodología más adecuada en función de las características de cada proyecto.

2. Modelos de proceso y metodologías

Para empezar, vamos a definir proceso de software y modelo de proceso de software:

- Proceso de software:

*Es «una serie de actividades relacionadas que conducen a la elaboración de un producto de software»
(Sommerville, 2011, p. 28).*

- Modelo de proceso de software:

Es, como todo modelo, una representación simplificada y abstracta de ese proceso (la realidad) que nos ayuda a comprenderlo.

2. Modelos de proceso y metodologías

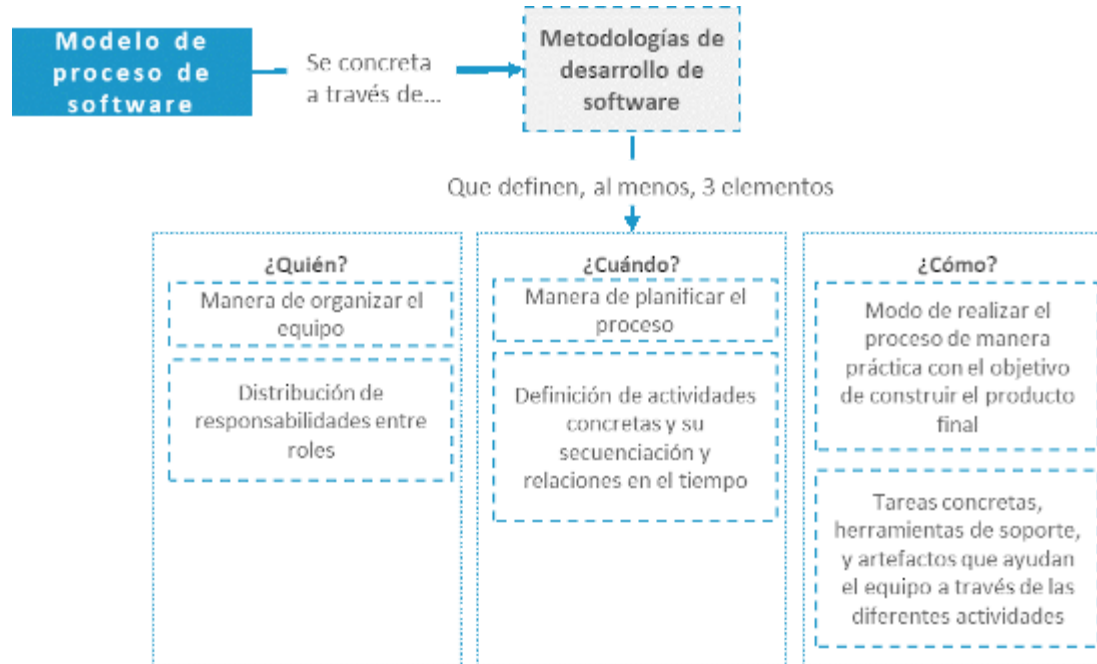
Se han desarrollado numerosos modelos de proceso, pero no hay que entenderlos como representaciones estáticas y rígidas de la manera óptima de elaborar un producto software, sino como enfoques adaptables a las necesidades de cada desarrollo

Los diferentes modelos de proceso pueden concretarse mediante un conjunto específico de actividades, personas y herramientas.

A estas concreciones, de carácter práctico y que proporcionan marcos de trabajo para el desarrollo, las conocemos como metodologías de desarrollo de software.

2. Modelos de proceso y metodologías

En general, cualquier metodología debería definir al menos tres elementos



2. Modelos de proceso y metodologías

Volviendo a los procesos del software, Pressman (2010) nos ofrece una definición más precisa:

«Una estructura para las **actividades, acciones y tareas** que se requieren a fin de construir un software de alta calidad» (p.26).

Actividades

Buscan lograr metas de carácter general (por ejemplo, comprender los objetivos del sistema, o ponerlo en producción).

Acciones

Son conjuntos de tareas que producen resultados importantes (el diseño arquitectónico del sistema, el documento de especificación de requisitos, etc.).

Tareas

Se centran en objetivos pequeños y muy bien definidos (la construcción de una prueba unitaria, el diseño de una pantalla de la aplicación, etc.).

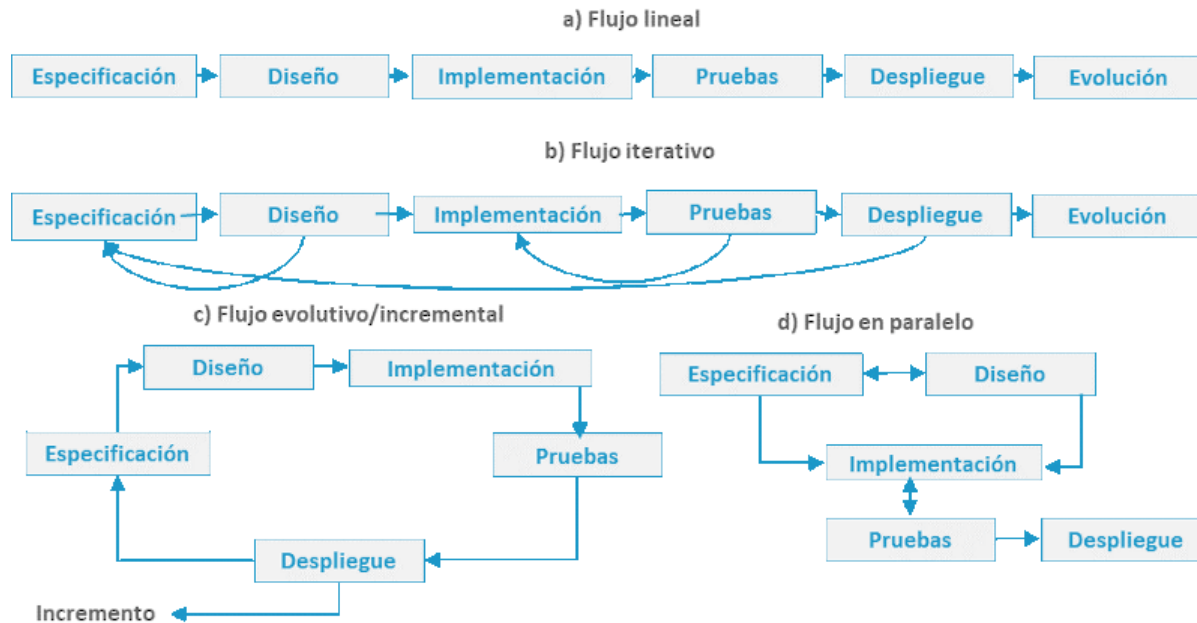
2. Modelos de proceso y metodologías

La experiencia ha permitido identificar un conjunto de actividades básicas que comparten todos los procesos de software. Estas actividades aparecen resumidas en la siguiente tabla según dos autores diferentes, pero si nos fijamos detenidamente, en el fondo comparten una estructura similar.

Actividades de los procesos de software	
Según Sommerville	Según Pressman
<ol style="list-style-type: none">1. Especificación del software. Definición de la funcionalidad esperada y sus restricciones.2. Diseño e implementación. Son las labores de desarrollo propiamente dichas, que buscan alcanzar las especificaciones definidas.3. Validación del software. Tareas relacionadas con la comprobación de que lo desarrollado cumple con lo que espera el cliente.4. Evolución del software. Tiene en cuenta las necesidades cambiantes del cliente, y la evolución del entorno.	<ol style="list-style-type: none">1. Comunicación. Diálogo con el cliente y otros interesados para comprender los objetivos y requisitos.2. Planificación. Consiste en programar todas las actividades necesarias teniendo en cuenta los objetivos, los riesgos, los recursos, etc.3. Modelado. Labores que persiguen analizar el problema para comprenderlo mejor y plantear las líneas generales de la solución.4. Construcción. Se trata de la generación del código y las pruebas asociadas.5. Despliegue. Es la entrega al cliente o usuario final para su evaluación.

2. Modelos de proceso y metodologías

Escojamos un conjunto de actividades básicas u otro, todas ellas se deben combinar y secuenciar en lo que conocemos como flujo de proceso, que puede adoptar distintas configuraciones básicas.



2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos:

- Serie de configuraciones canónicas, que podemos denominar como las más antiguas históricamente, y trataron de poner orden en los procesos de desarrollo de software adoptando un enfoque muy formal.
- Alrededor de estos modelos se desarrollaron un número de metodologías tradicionales, entre las que podemos citar el Proceso Unificado Racional (Kruchten, 2003), Microsoft Solutions Framework (Turner, 2006) o Métrica v.3 (Consejo Superior de Informática, 2001).
- Recientemente han ido apareciendo nuevos modelos, que relajan las imposiciones y simplificaciones de otros modelos tradicionales, con el fin de permitir que los procesos de desarrollo se adapten con mayor velocidad y menor costo a los posibles cambios e imprevistos de un proyecto.

2. Modelos de proceso y metodologías

Modelos de proceso ágiles:

- Siguen una serie de principios generales, pero dejan gran libertad en cuanto a las particularidades de su implementación en cada proyecto concreto.
- Por este motivo, muchos autores no consideran las metodologías ágiles como auténticas metodologías, pues dejan muchos aspectos sin concretar y no siempre responden a las preguntas ¿Quién?, ¿Cuándo? y ¿Cómo?
- Pero este es, precisamente, el fundamento y origen de los métodos ágiles, tratar de ofrecer herramientas para facilitar la adaptación al cambio en los proyectos de desarrollo.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

Estos modelos, y las metodologías que en ellos se apoyan, se preocupan por definir de manera precisa y detallada las distintas actividades y tareas, con el objetivo de desarrollar software de calidad estableciendo fechas y presupuestos concretos.

En todo momento se busca minimizar la incertidumbre asociada al proyecto.

Modelo en cascada

El modelo en cascada, o ciclo de vida clásico, es el más antiguo tiene sus orígenes en la década de 1970 (Royce, 1970).

Representa el proceso como una secuencia de actividades claramente definidas, donde es necesario finalizar por completo una fase, antes de continuar con la siguiente.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

Modelo en cascada

Este modelo solo debería aplicarse en procesos donde los requisitos están claramente definidos desde un comienzo y es poco probable que cambien.

Su poca flexibilidad hace que plantee muchos problemas en determinados escenarios (Pressman, 2010):

- Los proyectos reales rara vez siguen un flujo puramente lineal, siendo necesarias las iteraciones y revisiones de fases previas, lo cual es difícil de gestionar según este modelo.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

Modelo en cascada

- ▶ Muchos proyectos no están claramente definidos desde un comienzo, y solo se dispone de una especificación de requisitos muy genérica y parcial.
- ▶ Se debe esperar hasta la finalización del proyecto para que el cliente disponga de una versión funcional del producto.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

Modelo incremental

Para resolver algunos de los problemas que plantea el modelo en cascada, surge este modelo.

Está orientado a construir versiones incrementales del producto, de manera que el software funcional llegue cuanto antes al cliente, y este pueda dar una retroalimentación.

El primer incremento puede contener solo un conjunto de funcionalidades básicas, mientras que a medida que progresa el proyecto se va aumentando la funcionalidad y complejidad del producto.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

Modelo incremental

Dentro de cada incremento se puede seguir un flujo puramente lineal, iterativo o incluso paralelo, pero el objetivo fundamental es liberar versiones del producto que resulten útiles.

Este modelo tiene una serie de ventajas con respecto al modelo en cascada (Sommerville, 2011) y también presenta algunos problemas.

Modelo incremental

Ventajas	Problemas
<ul style="list-style-type: none">▶ Facilita la retroalimentación del cliente.▶ Reduce el costo de adaptación de requisitos.▶ Reduce el esfuerzo de análisis y documentación o, al menos, lo distribuye.▶ Se mejora el TTM (<i>Time To Market</i>) y el ROI (<i>Return Of Investment</i>), puesto que el cliente dispone antes de versiones funcionales de las que puede comenzar a obtener un rendimiento.	<ul style="list-style-type: none">▶ Se pierde visibilidad y control sobre el proceso global.▶ Las continuas modificaciones sobre el sistema pueden hacer que la estructura general del software tienda a degradarse, si no se aplican técnicas específicas de ingeniería, como la refactorización, lo cual implica tiempo y recursos adicionales.▶ Ciertas organizaciones de gran tamaño requieren de procedimientos burocráticos más estrictos, y en ocasiones están sujetas a regulaciones externas.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

El modelo evolutivo

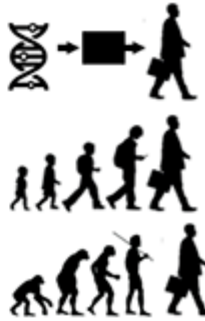
Este modelo es similar al incremental, pero se centra más en la evolución del sistema, tanto en su implementación como en la elaboración de los requisitos cuando en un comienzo no resultan del todo claros.

Se trata de ir mejorando el software de manera progresiva e iterativa, al mismo tiempo que se profundiza en su comprensión, empleando herramientas como la construcción de prototipos.

En la siguiente figura podemos observar un resumen conceptual de los anteriores modelos.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

Modelo en cascada	Fuerte planificación orientada a la construcción del sistema final.	
Modelo incremental	Planificación por iteraciones orientadas a producir incrementos funcionales.	
Modelo evolutivo	Planificación por iteraciones orientadas a mejorar progresivamente el sistema y su comprensión.	

El **modelo en cascada** requiere como información de partida una descripción detallada y precisa del sistema que permita su completa planificación, comportándose para el cliente como una caja negra que ofrece finalmente un producto terminado.

2. Modelos de proceso y metodologías

Modelos de proceso prescriptivos

El **modelo incremental** puede no tener una especificación del todo precisa, pero a través de las sucesivas iteraciones se van generando nuevas versiones funcionales del sistema, que crece a medida que avanza el proyecto.

El **modelo evolutivo** trabaja con versiones parciales, incompletas, a menudo simples prototipos, de manera que el diseño y el sistema se van refinando progresivamente hasta la finalización del proyecto.

2. Modelos de proceso y metodologías

Modelos de proceso especializados

Estos modelos comparten muchas características con los modelos anteriores, o combinaciones de estos, pero se orientan a enfoques del proceso de ingeniería mucho más específicos:

Desarrollo basado en componentes (orientado a la reutilización)

En todo proceso de desarrollo se realiza algún tipo de reutilización, aunque sea de manera informal. Hoy en día, cada vez es más necesaria la aplicación de técnicas de reutilización —y posible, dada la gran variedad de código disponible—, sobre todo cuando pensamos en sistemas grandes y complejos.

2. Modelos de proceso y metodologías

Modelos de proceso especializados

Desarrollo basado en componentes (orientado a la reutilización)

Es posible reducir el tiempo de desarrollo y el coste del proyecto, y en general aumentar la fiabilidad y seguridad al emplear componentes ya probados.

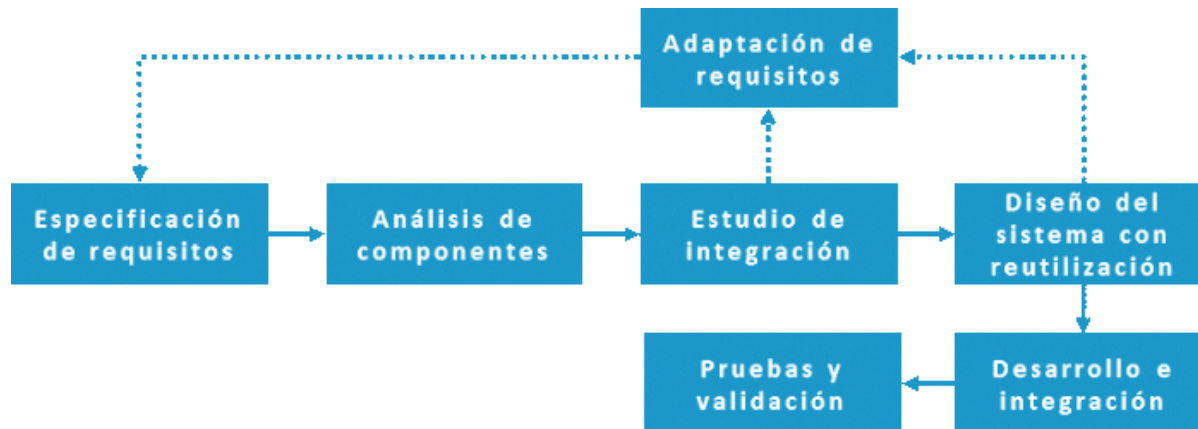
Cuando este enfoque se sistematiza y guía rigurosamente el proceso de desarrollo, aparecen actividades específicas.

2. Modelos de proceso y metodologías

Modelos de proceso especializados

Desarrollo basado en componentes (orientado a la reutilización)

Como podemos ver, el análisis de los componentes disponibles y su posible integración en el sistema puede obligar a la modificación de requisitos iniciales, si con ello se alcanzan los objetivos de ahorro mencionados.



2. Modelos de proceso y metodologías

Modelos de proceso especializados

Métodos formales

Se trata de una variación del modelo en cascada que parte de una representación y modelado muy precisos, en lenguaje matemático, del sistema que se va a desarrollar.

Según Sommerville (2011) «son apropiados en sistemas con fuertes requisitos de seguridad, fiabilidad o protección» (p. 32). El objetivo es lograr un producto libre de defectos, precisando para ello de un gran consumo de tiempo y recursos y unas capacidades específicas de los desarrolladores. Además, esta especificación formal dificulta la comunicación con el cliente (Pressman, 2010).

2. Modelos de proceso y metodologías

Modelos de proceso ágiles

Estos modelos enfatizan la «agilidad» como característica deseable de cualquier proceso de desarrollo, y más cuanto mayor es la incertidumbre a la que nos enfrentamos.

Proporcionan un enfoque mucho más informal que los anteriores, y tienen asociada, en general, mucha menor carga burocrática, lo cual ofrece grandes ventajas en muchos escenarios y tipologías de proyecto.

3. Agilidad y procesos ágiles

A menudo, durante el desarrollo de los proyectos de software debemos enfrentarnos con un número de incertidumbres que a priori pueden resultar difíciles de valorar.

Esto ha sido así desde los comienzos de la ingeniería del software, pero durante las últimas décadas —con la popularización del uso de Internet y el auge de las aplicaciones web y móviles— los siguientes temas están aún más de relieve:

- La dificultad para anticipar la volatilidad de los requisitos o pronosticar las prioridades del cliente respecto a los requisitos existentes a medida que el proyecto avanza.
- Muchos productos software están en continua evolución, de manera que las actividades de análisis, diseño e implementación se suceden de manera muy rápida, a menudo solapada y muy acoplada. Ello obliga a desarrollar estas actividades de manera simultánea en períodos de tiempo reducidos.

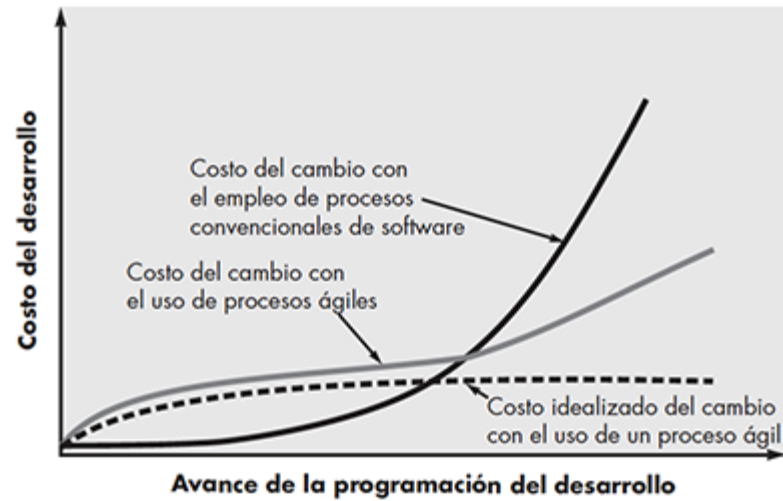
3. Agilidad y procesos ágiles

Se trata, pues, de desarrollar metodologías capaces de afrontar «lo impredecible», manteniendo al mismo tiempo el grado de avance del proyecto.

Por otro lado, sucede que el costo de los proyectos aumenta exponencialmente con el tiempo, a medida que los cambios se introducen de manera más tardía.

Bajo un enfoque tradicional, los cambios en las primeras fases de elaboración de requisitos son sencillos, pero un cambio introducido hacia la finalización del proyecto puede obligar a revisar todas las fases de elaboración previas.

3. Agilidad y procesos ágiles



3. Agilidad y procesos ágiles

Los partidarios de los métodos ágiles defienden que este enfoque es capaz de suavizar esta curva de costos. El motivo es que esta nueva familia de métodos se apoya fundamentalmente en modelos incrementales, de manera que no es necesario un esfuerzo de planificación previo del sistema en su totalidad y un diseño e implementación monolíticos, sino que el proceso se realiza en pequeños incrementos, de manera que en cada uno se desarrollan únicamente las actividades estrictamente necesarias.

La agilidad es mucho más que la capacidad de responder de manera rápida y efectiva a los cambios; supone una filosofía completa y general del proceso de desarrollo. Durante las dos últimas décadas del siglo XX diferentes equipos de desarrollo adoptaron nuevos métodos y prácticas que buscaban este nuevo enfoque general; en 2001 mantuvieron una reunión en la que formalizaron estas ideas y las materializaron en lo que conocemos como el «Manifiesto ágil».

4. Manifiesto ágil

Entre los días 11 y 13 de febrero de 2001, un grupo de diecisiete expertos desarrolladores y gestores de proyectos se reunieron en Utah para compartir experiencias y reflexionar sobre nuevas experiencias en los procesos de desarrollo de software.

El resultado fue el «Manifiesto ágil» que se concreta en cuatro valores y doce principios fundamentales.

Valores fundamentales:

- **Los individuos y sus interacciones**, sobre los procesos y las herramientas.
- **El software funcional**, sobre la documentación extensa.
- **La colaboración con el cliente**, sobre la negociación contractual.
- **La respuesta al cambio**, sobre el seguimiento estricto de un plan.

4. Manifiesto ágil

Valores fundamentales:

Valoración de individuos e interacciones

El software está construido por personas. Un factor clave del éxito de los proyectos es el equipo, las relaciones entre sus miembros y la habilidad para construir el propio entorno de trabajo. Es un error operar a la inversa; construyendo primero el entorno de trabajo esperando que la gente se adapte a él (Fernández, 2012).

Importan tanto las interacciones entre los miembros del equipo como el flujo de información entre cliente o usuario final y desarrolladores. Idealmente, el cliente debe encontrarse próximo al equipo, o formando parte de este, de manera que los canales de comunicación sean los adecuados. Si se consigue un nivel de comunicación eficiente, los procesos y herramientas concretos pasan a un segundo plano, o en todo caso deben ser elegidos y configurados por los propios actores del proceso de desarrollo.

4. Manifiesto ágil

Valores fundamentales:

Valoración del software en funcionamiento

La documentación de un proyecto es una herramienta fundamental para comprenderlo. Resulta especialmente útil en la fase de mantenimiento y siempre que haya que modificar o mejorar el sistema (especialmente, cuando el equipo de desarrollo y el de mantenimiento no coincidan). Pero generar y mantener la documentación actualizada es costoso, y en muchos casos inútil.

Los firmantes del manifiesto, por su experiencia, descubrieron que, en la mayoría de los casos, cuando hay que resolver un problema lo importante es hacer que el software funcione, atender a los usuarios y conseguir que el sistema avance. La documentación pasa a un segundo plano, bien por falta de tiempo o de otros recursos.

4. Manifiesto ágil

Valores fundamentales:

Valoración del software en funcionamiento

La documentación es necesaria, pero su desarrollo debe mantenerse en los niveles estrictamente necesarios para que cumpla su misión, siendo breve y centrándose en lo fundamental.

Los recursos deben orientarse hacia tareas que verdaderamente aportan valor al proyecto, centradas en el desarrollo del propio sistema. A partir de esta idea, las metodologías ágiles proponen artefactos que vienen a sustituir a la documentación tradicional.

4. Manifiesto ágil

Valores fundamentales:

La colaboración con el cliente

Uno de los principales puntos de fricción en los proyectos de desarrollo de software es la negociación contractual entre cliente y desarrollador y, especialmente, las posibles modificaciones que surgen a lo largo del desarrollo del proyecto. Cambios de alcance, modificaciones de requisitos o variaciones de plazos, suponen escollos para el proyecto, y obligan a los desarrolladores a adoptar posiciones a caballo entre las labores de analista y abogado.

El «Manifiesto ágil» parte de la base de que tanto cliente como equipo de desarrollo tienen el objetivo común de desarrollar un sistema funcional que satisfaga a los usuarios finales. Para ello, es importante conseguir los canales de comunicación adecuados y la complicidad con el cliente, que es considerado como uno más del equipo.

4. Manifiesto ágil

Valores fundamentales:

La respuesta al cambio

El cambio resulta inevitable. Las necesidades de las complejas organizaciones de hoy en día son cambiantes, y el propio entorno tecnológico, la competencia, los consumidores, etc., evolucionan a un ritmo cada vez más vertiginoso. Cualquier plan de proyecto puede verse afectado por múltiples imprevistos, y una clave del éxito es la adaptación rápida y la replanificación ágil.

No tiene sentido elaborar grandes planes de proyecto, consumiendo un número de recursos elevado, cuando el futuro puede resultar incierto o cambiante.

4. Manifiesto ágil

Los doce principios

Asociados a estos cuatro valores, los firmantes del manifiesto elaboraron también un listado de doce principios, que siguen todos los proyectos desarrollados bajo las denominadas metodologías ágiles:

- La mayor prioridad es satisfacer al cliente con la entrega temprana y continua de software valioso.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Se aprovecha el cambio para proporcionar ventaja competitiva al cliente.
- Entrega frecuente de software funcional, preferiblemente en períodos de tiempo cortos.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

4. Manifiesto ágil

Los doce principios

- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo necesarios, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo, y entre sus miembros, es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos ágiles promueven el desarrollo sostenible, manteniendo un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
- La simplicidad —el arte de maximizar el trabajo no realizado— es esencial.

4. Manifiesto ágil

Los doce principios

- ▶ Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
- ▶ A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

5. Algunas metodologías ágiles

A continuación, analizaremos algunas de las características principales de una muestra de metodologías ágiles. Será interesante comprobar como todas ellas, a pesar de sus diferencias, siguen de manera general los valores y principios formulados en el «Manifiesto ágil».

Programación extrema (XP)

La programación extrema (eXtreme Programming, XP) es una de las metodologías ágiles más utilizadas y estudiadas. Aunque no se utilice en su totalidad, muchas de las prácticas que propone son aprovechadas por otros marcos metodológicos.

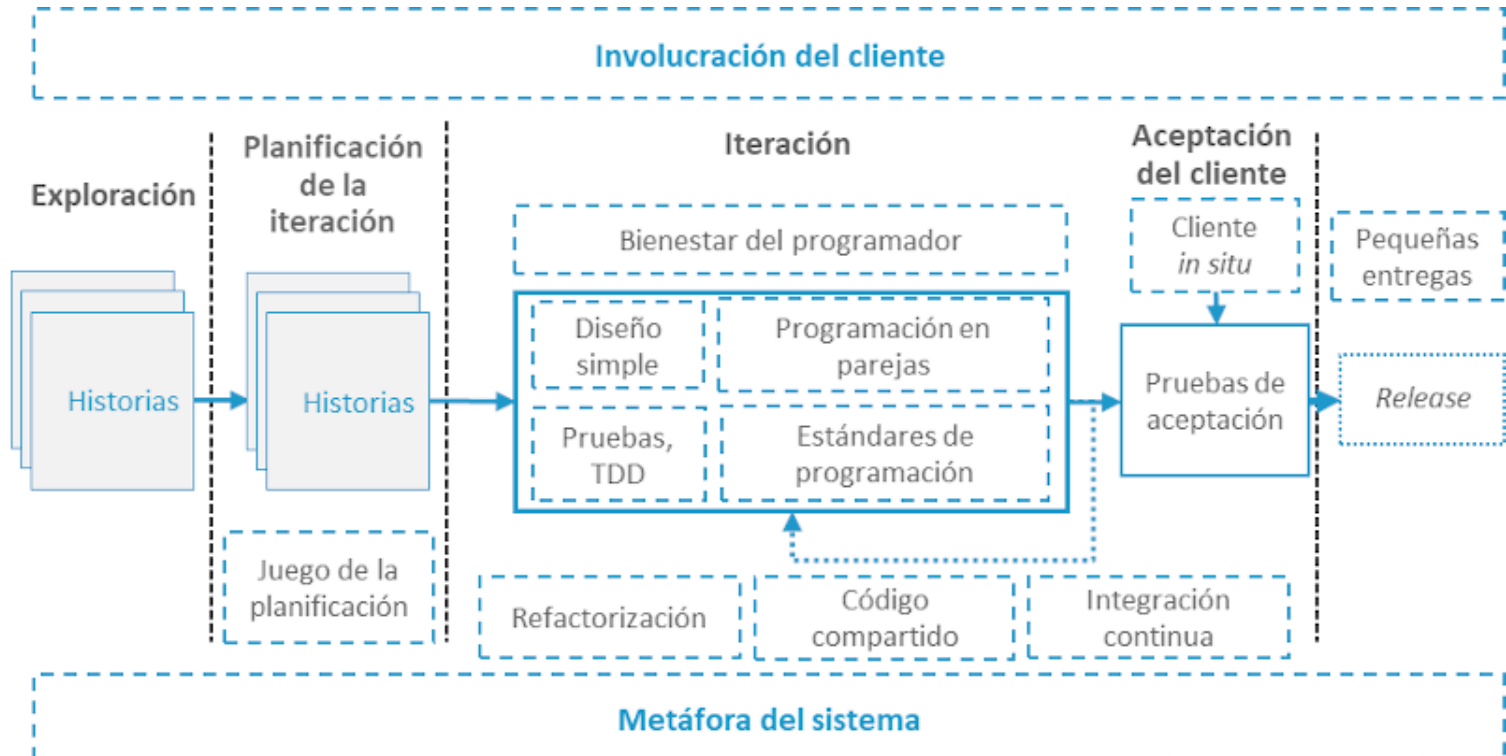
Fue desarrollada por Kent Beck, Ron Jeffries y Ward Cunningham a finales de la década de 1990, y propone soluciones prácticas en tres niveles (Fernández, 2012), todos ellos relacionados.

Programación extrema (XP)

SOLUCIONES APORTADAS POR LA PROGRAMACIÓN EXTREMA

ÁMBITO DE PROGRAMACIÓN	ÁMBITO DE GESTIÓN DEL EQUIPO	ÁMBITO DE LOS PROCESOS
Diseño simple, desarrollo dirigido por pruebas, refactorización y codificación siguiendo estándares de programación.	Propiedad colectiva del código, programación en parejas, integración continua, bienestar del programador y metáfora del sistema.	Cliente <i>in situ</i> , entregas frecuentes y juego de la planificación.

Programación extrema (XP)



5. Algunas metodologías ágiles

Scrum

Scrum como metodología no nació en el contexto del desarrollo del software, sino en el del desarrollo de producto (Takeuchi y Nonaka, 1986). Está especialmente pensada para equipos de tamaño pequeño (de tres a nueve desarrolladores), cuyas actividades se descomponen en iteraciones de duración fija, o sprints.

En Scrum se establecen algunos mecanismos para mantener bajo control la incertidumbre asociada a los proyectos y, a grandes rasgos, podemos descomponer esta metodología en tres grandes fases (Fernández, 2012):



FASE 1

Se definen y analizan las funcionalidades esperadas del sistema. Los documentos fundamentales en esta fase son el *Product Backlog* (o pila de producto) y el *Sprint Backlog*, que son los artefactos que permiten mantener los requisitos del sistema en su conjunto, y los asociados a cada iteración.

FASE 2

Esta es la fase en la que las tareas de desarrollo se distribuyen entre los miembros del equipo. Durante cada iteración se realiza un seguimiento del proceso de desarrollo mediante reuniones diarias de corta duración.

FASE 3

Se evalúa la entrega al final de cada iteración, analizando el trabajo realizado, el progreso general del proyecto y también el propio proceso de desarrollo, adoptando el equipo una actitud reflexiva.

5. Algunas metodologías ágiles

Crystal Methodologies

Se trata de una familia de metodologías creadas por Alistair Cockburn. Se centran en el equipo, entendido como el conjunto de personas que deben desarrollar el software.

Se parte de la base de que el desarrollo de aplicaciones es una serie de juegos de cooperación, con recursos limitados, donde la comunicación juega un papel fundamental.

En este sentido, es fundamental el tamaño del equipo, puesto que no es lo mismo organizar un grupo pequeño, de tres desarrolladores, que uno más grande, de veinte.

5. Algunas metodologías ágiles

Crystal Methodologies

Por tanto, en función del tamaño del equipo aparecen las diferentes variantes de Crystal (Fernández, 2012):

- Crystal Clear, para equipos de menos de ocho personas.
- Crystal Yellow, para equipos de diez a veinte personas.
- Crystal Orange, para equipos de veinticinco a cincuenta personas, etc.

5. Algunas metodologías ágiles

Crystal Methodologies

Las distintas metodologías comparten unos fundamentos comunes, mientras que los roles, patrones de proceso y prácticas son particulares en cada variante.

Las prioridades comunes a todas las metodologías son las siguientes:

Seguridad en el resultado del proyecto

- busca obtener un resultado razonable en función de las prioridades particulares de cada proyecto y los recursos disponibles.

Eficiencia en el proceso de desarrollo

- está estrechamente relacionado con el aprovechamiento de los recursos disponibles.

5. Algunas metodologías ágiles

Crystal Methodologies

Aceptación de las convenciones

- hace referencia a la comodidad que sienten los desarrolladores al adoptar las normas impuestas por la metodología.
- esta es una característica importante, que surge de la comprobación empírica de que en muchas organizaciones las metodologías impuestas no siempre son aceptadas con gusto y muchas veces son obviadas.

5. Algunas metodologías ágiles

Crystal Methodologies

Además, se definen siete propiedades fundamentales que cualquier proyecto de desarrollo debería poseer para tener unas ciertas garantías de éxito (Lowe, 2016).

Todas ellas se aplican a proyectos de cualquier tamaño, pero las tres primeras se consideran fundamentales e ineludibles:

- **Entregas frecuentes.** Para Cockburn (2004) la propiedad más importante de cualquier proyecto es la entrega de productos funcionales a los usuarios finales cada poco tiempo. En proyectos de carácter general se establece un intervalo máximo de cuatro meses (y preferiblemente dos), mientras que en desarrollos web el espacio entre entregas se reduce a una semana.
- **Mejora reflexiva.** El equipo debe dialogar y reflexionar sobre su propia práctica para proponer acciones de mejora.

5. Algunas metodologías ágiles

Crystal Methodologies

- **Comunicación osmótica.** El concepto de ósmosis hace referencia a la absorción de información en segundo plano, aunque no se produzca una comunicación formal, por la simple escucha de lo que sucede en el espacio de trabajo. Por este motivo los equipos deben estar localizados físicamente en el mismo lugar.
- **Seguridad personal.** Se debe facilitar que todos los integrantes del equipo tengan libertad para expresar sus opiniones sin temor a represalias.
- **Focus.** Cada miembro del equipo debe contar con la capacidad, tiempo y herramientas para concentrarse en lo que debe y puede hacer en cada momento, sin interrupciones ni distracciones.

5. Algunas metodologías ágiles

Crystal Methodologies

- ▶ **Acceso directo a usuarios expertos.** Se trata de obtener retroalimentación temprana y de calidad por parte de los usuarios finales del sistema.
- ▶ **Un entorno tecnológicamente apropiado.** El equipo debe contar con herramientas que permitan las pruebas automatizadas o la integración frecuente.

5. Algunas metodologías ágiles

Feature-Driven Development (FDD)

Esta metodología se basa en iteraciones de duración muy corta, siempre inferior a dos semanas. Todas las tareas de análisis, diseño e implementación se centran en cumplir una serie de características (features) del software, que deben cumplir las siguientes propiedades (Fernández, 2012):

- Ser sencillas y poder ser desarrolladas en poco tiempo (entre uno y diez días).
- Representar una aportación para el cliente y su negocio.
- Se deben expresar según el patrón "acción", "resultado", "objeto" (por ejemplo, «calcular el importe total de un pedido»).

5. Algunas metodologías ágiles

Feature-Driven Development (FDD)

FDD hace énfasis en las tareas de diseño, y considera un modelo de proceso con cinco pasos (Palmer, S. 2009; Palmer y Felsing, 2001):

Desarrollo de un modelo de dominio general.

- El equipo, en colaboración con el cliente, desarrolla un modelo general del sistema de manera iterativa bajo la supervisión de un arquitecto jefe.
- Este proceso permite a los expertos en el dominio de aplicación y al equipo de desarrolladores compartir información y establecer un lenguaje común.
- El modelo inicial da una representación amplia del problema, sus objetos e interacciones, y va adquiriendo profundidad y refinamiento a través de las sucesivas iteraciones.

5. Algunas metodologías ágiles

Feature-Driven Development (FDD)

Construcción una jerarquía de características.

- Se descompone la funcionalidad en características sencillas. Cada una de ellas debe poder ser implementada en menos de dos semanas (y preferiblemente entre uno y tres días).
- No es el modelo el que guía el desarrollo, sino estas características. Las características se organizan en una jerarquía de tres niveles (áreas del dominio, actividades de cada área, y características concretas).

5. Algunas metodologías ágiles

Feature-Driven Development (FDD)

Planificación por característica.

- Se analizan las características en función de su prioridad, dependencias y complejidad, y se define un orden de implementación.
- Se agrupan en paquetes, y cada uno es asignado a un programador jefe, que se responsabiliza y especializa en ese conjunto de características.

5. Algunas metodologías ágiles

Feature-Driven Development (FDD)

Diseño por característica.

- Tras la planificación inicial, comienza el proceso iterativo.
- En cada iteración, el programador jefe selecciona un conjunto de características de su paquete, y se elabora un diseño más detallado y una descomposición en clases que se distribuyen entre los programadores.
- Así, la responsabilidad sobre cada clase es individual y la propiedad del código no es compartida, como sucede en XP, buscándose la especialización.

5. Algunas metodologías ágiles

Feature-Driven Development (FDD)

Desarrollo por característica.

- Durante el resto de la iteración cada programador implementa y prueba el código, y tras un proceso de inspección se produce la integración en el repositorio general.

Feature-Driven Development (FDD)



5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Esta metodología, propuesta por Jim Highsmith (2000), parte del convencimiento de que las necesidades del cliente van a cambiar siempre, tanto durante el proyecto como tras su finalización.

Más que de una metodología, estamos hablando de un conjunto de procedimientos que permiten a las empresas alcanzar una cultura de adaptación.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

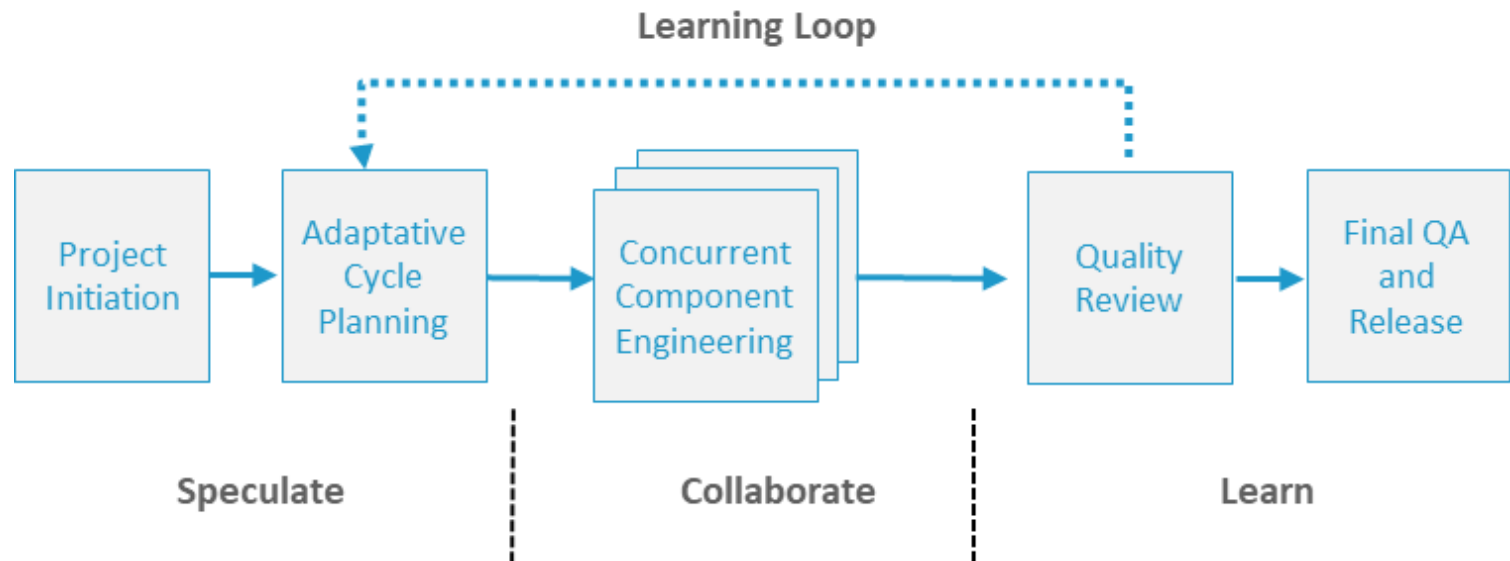
Sus objetivos son los siguientes (Fernández, 2012; Highsmith, 2002):

Adaptive Software Development (ASD)

OBJETIVOS		
Concienciar a la organización sobre la necesidad de trabajar con el cambio, y no contra el cambio.	Desarrollar proyectos iterativos de gestión del cambio y facilitar la colaboración entre las personas.	Marcar una estrategia de desarrollo rápido sin olvidar el rigor y la disciplina necesarios.

Adaptive Software Development (ASD)

ASD plantea un modelo de proceso con las tradicionales fases (planificación, diseño y construcción) sustituidas por un modelo más dinámico y abstracto: especulación, colaboración y aprendizaje (Highsmith, 2002).



5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Fase de especulación.

El término ***especulación*** reconoce la naturaleza incierta de los sistemas complejos, y anima a la exploración y la experimentación.

Asume la existencia de incertidumbres, y aunque la planificación no se abandona, debemos estar dispuestos a apartarnos en cualquier momento de planes predefinidos.

Aquí encontramos dos prácticas:

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Iniciación del proyecto.

- Se trata, según Highsmith (2002), de «definir la misión del proyecto y sus objetivos, entendiendo las restricciones, estableciendo la organización del proyecto, identificando y esbozando requisitos, estimando el tamaño y el alcance, e identificando los riesgos principales».
- Esta fase dura entre dos y cinco días para proyectos pequeños y medianos, y puede alargarse hasta las tres semanas en proyectos grandes.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Planificación adaptativa.

- Con la información de alcance y los recursos disponibles se define un marco temporal para el proyecto.
- Dentro de esta duración total, se establece el número de iteraciones y la duración de cada una.
- Cada una de ellas queda asociada a un tema general u objetivo, de manera que pueda suministrar un conjunto de características relacionadas demostrables al cliente para su revisión, al finalizar.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Fase de colaboración.

En ASD se parte de que los sistemas complejos no se construyen; evolucionan.

Además, estas aplicaciones requieren de la adquisición, análisis y utilización de grandes cantidades de información, no manejables por una única persona, y cuya comprensión requiere a menudo de capacidades y conocimientos específicos.

En este contexto las habilidades comunicativas del equipo son fundamentales.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Fase de colaboración.

- Desarrollo concurrente del sistema, aplicando técnicas de ingeniería orientada a componentes.
- El equipo técnico, de carácter multidisciplinar, se concentra en la producción del software, mientras que los jefes de proyecto se encargan de facilitar la colaboración y el desarrollo de actividades concurrentes.
- En equipos pequeños esta colaboración es sencilla y emplea técnicas como la simple charla en pasillos. En proyectos y equipos grandes la participación del mánager es fundamental.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Fase de aprendizaje.

En proyectos complejos, más que en ningún otro caso, se pueden cometer equivocaciones.

Es necesario aprender del proceso y de nosotros mismos como equipo, empleando técnicas como las retrospectivas y focus groups con clientes.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Aquí encontramos dos prácticas fundamentales:

La revisión de calidad

- desde la perspectiva del cliente **empleando focus groups**
- desde un punto de vista técnico **empleando la programación en parejas**
- las revisiones técnicas o la revisión de la arquitectura al final de cada iteració.

5. Algunas metodologías ágiles

Adaptive Software Development (ASD)

Preguntas, respuestas y release

- Dentro de la fase de aprendizaje es vital dedicar un tiempo a que el equipo reflexione y analice sus propias prácticas para identificar fallos y puntos de mejora.
- También se debe analizar el estado global del proyecto, evaluando el grado de ajuste a la planificación inicial y adoptando las acciones correctivas necesarias.

6. Ágil vs. tradicional

Las metodologías ágiles introducen beneficios indudables. Sin embargo, en ocasiones muchos de sus principios son difíciles de cumplir. En esta sección consideraremos dos aspectos clave de esta problemática: la posibilidad de escalar los métodos ágiles a grandes proyectos y los factores a considerar a la hora de elegir una metodología.

La escalabilidad de los métodos ágiles

Todas las metodologías ágiles nacen para ser empleadas por equipos de desarrollo pequeños, donde es posible la misma localización física de los participantes y el desarrollo de una comunicación oral. Sin embargo, las necesidades que cubren son comunes a los sistemas de grandes dimensiones, y por tanto resulta interesante plantear posibilidades de escalado.

6. Ágil vs. tradicional

Los sistemas de grandes dimensiones presentan en general una serie de particularidades (Sommerville, 2011):

PARTICULARIDADES	
Formados por otros subsistemas comunicados entre sí, asignados a distintos equipos distribuidos geográficamente	Es difícil que los equipos tengan una visión completa del sistema en conjunto y se comuniquen entre sí de manera fluida.
La conexión de diferentes sistemas	Dificulta el enfoque de desarrollo incremental o la integración frecuente del sistema en su conjunto.
Suelen estar sujetos a reglas y regulaciones externas	Estas imponen restricciones adicionales sobre los procesos permitidos y el tipo de documentación generada.
Su desarrollo suele abarcar períodos dilatados de tiempo	Durante este tiempo aumenta la probabilidad de que aparezcan cambios imprevistos.
El número de <i>stakeholders</i> (clientes, usuarios finales, administradores, etc.) suele ser elevado	Lo que dificulta su participación e implicación en el proceso.

6. Ágil vs. tradicional

Para poder mantener los principios fundamentales que sustentan las metodologías ágiles —flexibilidad en la planificación, integración continua y liberación frecuente, o la buena comunicación entre los participantes— es necesario resolver, al menos, las siguientes cuestiones:

HACER ÉNFASIS EN EL DISEÑO Y LA DOCUMENTACIÓN	FAVORECER LA COMUNICACIÓN ENTRE EQUIPOS	USAR HERRAMIENTAS DE GESTIÓN DE CONFIGURACIÓN
Describiendo claramente aspectos como la arquitectura del sistema o la división de trabajo entre equipos.	Utilizando los medios tecnológicos necesarios.	Que faciliten la integración continua, cuando sea necesario, tras un cambio realizado.

6. Ágil vs. tradicional

Finalmente, las grandes organizaciones presentan una cierta oposición a este tipo de metodologías, cuyas causas son las siguientes:

- ▶ La reticencia por desconocimiento y falta de experiencia, al ser métodos no empleados en el pasado.
- ▶ La existencia de normas transversales internas, sobre procedimientos y estándares de calidad, que por su carga burocrática resultan difícilmente compatibles con los métodos ágiles.

La elección de una metodología ágil

Además de las metodologías ágiles que hemos repasado, existen otras muchas. Pero ¿cuándo conviene emplear una de estas metodologías en nuestro proyecto, y qué factores debemos considerar?

6. Ágil vs. tradicional

El hecho de que los métodos ágiles presenten una serie de ventajas no significa que sea obligada o siempre preferible su utilización. Debemos considerar las características del proyecto, el perfil de los miembros del equipo, los recursos disponibles, los plazos de entrega, el entorno empresarial y social del desarrollo...

Algunos aspectos que deberíamos considerar son los siguientes (Fernández, 2012):

El tamaño del proyecto.

- ▶ Muchas de las metodologías ágiles fueron inicialmente ideadas para equipos de tamaño reducido. Aunque la mayoría disponen de extensiones a proyectos grandes, esta adaptación no resulta siempre sencilla.
- ▶ Cuando el proyecto implica la coordinación de muchas personas y agentes, puede ser preferible emplear métodos de planificación y control más rigurosos.

6. Ágil vs. tradicional

La volatilidad de los requisitos.

- El gran punto fuerte de los métodos ágiles es su capacidad de adaptación al cambio, sea cual sea su origen.
- Cuando los requisitos del proyecto están claramente definidos desde un comienzo, y los cambios son previsibles, puede resultar más práctica una metodología tradicional.

La criticidad del sistema.

- Ciertos sistemas, como los sistemas de control de vehículos, los relacionados con aplicaciones biomédicas, o los sistemas bancarios, se prestan poco al enfoque de entregas tempranas y frecuentes que proponen los métodos ágiles.
- Se desea que el sistema funcione perfectamente desde su primera puesta en producción, y se dispone en general de una especificación de requisitos inicial muy detallada, que hace prever pocos cambios durante el desarrollo.

6. Ágil vs. tradicional

La disponibilidad del cliente.

- Los métodos ágiles hacen énfasis en la involucración del cliente en todas las fases del desarrollo.
- Esto no siempre es posible, y en muchos casos es preferible negociar una participación inicial intensiva, de manera que los requisitos queden claramente definidos y formalizados, y el proyecto se ajuste a este compromiso en lo sucesivo.

La experiencia del equipo de desarrollo.

- Las metodologías ágiles son en gran medida muy «elitistas». Asumen una capacidad y nivel de conocimientos por parte de los miembros del equipo que no siempre resulta asequible para las organizaciones.
- A pesar de que son siempre los miembros más jóvenes los que se muestran más partidarios y menos amedrentados por el empleo de estas técnicas, su nivel de madurez como programadores puede resultar un inconveniente.

6. Ágil vs. tradicional

La cultura organizacional del entorno.

- Las metodologías ágiles introducen un grado de aparente informalidad que no es compatible con todas las organizaciones.
- La escasez de documentación formal, la manera en que se gestionan las relaciones interpersonales entre los diferentes roles, o el grado de comunicación con el cliente no siempre son viables.

6. Ágil vs. tradicional

Resumen

- Podríamos decir que las metodologías ágiles son especialmente apropiadas para proyectos dinámicos, con requisitos cambiantes, donde el cliente y los usuarios finales están disponibles de manera cercana para obtener retroalimentación y se dispone de un equipo pequeño y muy capacitado.
- Esto no quiere decir que en otro tipo de contextos las metodologías ágiles resulten inaplicables, pero es necesario valorar cada proyecto concreto con una mentalidad abierta.

7. Referencias

- Beck, K. (2000). Extreme Programming Explained. Embrace Change. Boston: Addison-Wesley.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). Manifiesto por el Desarrollo Ágil de Software. Recuperado de <http://agilemanifesto.org/iso/es/manifesto.html>
- Cockburn, A. (2004). Crystal clear: a human-powered methodology for small teams (1.a ed.). Boston: Addison-Wesley.
- Consejo Superior de Informática (2001). Métrica versión 3. Recuperado de <https://goo.gl/wtmEcH>
- Fernández, J. (2012). Introducción a las metodologías ágiles: otras formas de analizar y desarrollar [Material didáctico]. Barcelona: Universitat Oberta de Catalunya. Recuperado de <http://hdl.handle.net/10609/63466>
- Highsmith, J. (2002). Agile Software Development Ecosystems. Boston: Addison-Wesley Longman Publishing Co., Inc.

7. Referencias

- Highsmith, J. A., (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York: Dorset House Publishing Co., Inc.
- Kruchten, P. (2003). The Rational Unified Process: An Introduction (3^a ed.). Boston: Addison-Wesley.
- Lowe, D. (2016). What is Crystal Clear? [web]. Recuperado de <https://scrumandkanban.co.uk/what-is-crystal-clear/>
- Meier, J. D. (2014). Extreme Programming (XP) at a Glance (Visual) [Blog]. Recuperado de <https://blogs.msdn.microsoft.com/jmeier/2014/06/06/extreme-programming-xp-at-a-glance-visual/>
- Palmer, S. (2009). An Introduction to Feature-Driven Development [web]. Recuperado de <https://dzone.com/articles/introduction-feature-driven>
- Palmer, S. & Felsing, J. (2001). A Practical Guide to Feature-Driven Development (1^a ed.). Londres: Pearson Education.

7. Referencias

- Pressman, R. S. (2010). Ingeniería del software: un enfoque práctico (7ª ed.). México: McGraw-Hill Educación.
- Royce, W. W. (1970). Managing the development of large software systems. En Proceedings (pp. 328-338). USA: IEE WESCON.
- Sommerville, I. (2011). Ingeniería de software (9ª ed.). México: Pearson Educación de México.
- Takeuchi, H. & Nonaka, I. (1986). The New New Product Development Game. Harvard Business Review [web]. Recuperado de <https://hbr.org/1986/01/the-new-new-product-development-game>
- Turner, M. S. (2006). Microsoft Solutions Framework Essentials. USA: Microsoft Press.
- Tutorials Point. (2016). Adaptive Software Development Practices [web]. Recuperado de https://www.tutorialspoint.com/adaptive_software_development/adaptive_software_development.html