Tema 2: El proceso de desarrollo del software. Paradigmas o modelos de desarrollo del Software.

BLOQUE I:
INTRODUCCIÓN Y PARADIGMAS DE DESARROLLO EN INGENIERÍA DEL SOFTWARE

Segundo curso de Grado en Ingeniería Informática
Curso 2014-2015

Javier Sánchez Monedero
jsanchezm@uco.es
http://www.uco.es/users/i02samoj

Índice

Índice

1.

2. Introducción

Introducción

Advertencia

Advertencia
Estos apuntes no contienen todo el material necesario respecto al tema. Intentan dar una visión de conjunto y proporcionar los conocimientos básicos para que podáis consultar la bibliografía y manuales de referencia, así como material complementario colgado en Moodle y los ejercicios realizados en clase.

Introducción

- En Ingeniería del Software, la actividad de desarrollo se suele organizar a modo de proyectos, la manera de gestionar estos proyectos estará determinada por el **método de desarrollo** que queramos aplicar.
- El método de desarrollo definirá un **ciclo de vida** (qué **etapas** forman parte del proyecto de desarrollo), qué **procesos**, **actividades** y **tareas** tienen lugar en las diferentes etapas del ciclo de vida, **quién** se encargará de llevar a cabo cada una de las tareas y también la **interacción entre tareas, roles y personas.**

Un poco de humor

(Más en `http://www.dilbert.com/`)

Historia e influencia en métodos

- Disciplina de **gestión de proyectos** des los años cincuenta.
- Project Management Institute (PMI) es el modelo más popular.
- Una de las primeras metáforas que se aplicaron en el desarrollo de software es la gestión científica (*scientific management*, desarrollada por Frederik Taylor a finales del siglo XIX):
    - Descomponer el proceso industrial en un conjunto de pequeñas tareas lo más repetitivas posible que puedan ser ejecutadas por un trabajador altamente especializado.

Figura 1: Los buenos procesos son mejores que los buenos empleados.

Figura 2: Programación ágil.

    - Ejemplo clásico: la cadena de producción industrial.
    - Influencia el *ciclo de vida en cascada*.
- Método de producción Toyota: ***lean-manufacturing***[1], dos grandes principios:
    - Evitar producir productos defectuosos parando, si es necesario, la línea de producción.
    - La producción *just-in-time*: producir sólo aquellos productos que son necesarios en la fase siguiente y no acumular excedentes.
- La aplicación de esta filosofía al desarrollo de software es lo que se conoce como ***lean software development*** Poppendieck and Cusumano [2012].

3. Clasificación métodos

Clasificación métodos

Clasificación de métodos de desarrollo

- Las actividades, a grandes rasgos, son las mismas en los métodos.
- Las diferencias radican en el cuándo y el cómo se realizan las actividades.

Ejemplo
Métodos que realicen un análisis exhaustivo y formal de los requisitos antes de iniciar otras actividades vs construcción de software directamente sin apenas modelos.

Selección del método

Una de las primeras tareas del ingeniero de software será elegir el método de desarrollo más adecuado a la **naturaleza del proyecto** en función de Wysocki [2009]:

- riesgo
- valor de negocio
- duración (menos de tres meses, de tres a seis meses, más de seis meses, etc.)
- complejidad
- tecnología utilizada

[1] http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/

- número de departamentos afectados
- coste

Según si tenemos clara la necesidad que queremos cubrir (**objetivo**) y si conocemos o no los detalles de cómo será la **solución** (requisitos, tecnología, etc.).

	Sol. conocida	Sol. poco conocida
Objetivo claro	1	2
Objectivo poco claro	3	4

Cuadro 1: Tipos de proyectos software.

- **Grupo 1**: está claro qué queremos hacer y cómo lo haremos. En este caso, podremos elegir un método con poca tolerancia al cambio, pero que, en cambio, sea sencillo de aplicar.
- **Grupo 2**: Cambiar las ideas iniciales a medida que el proyecto avance y que facilite el descubrimiento de la solución mediante ciclos de retroalimentación. Actualmente, **la mayoría de los proyectos pertenecen a este grupo**.
- **Grupo 3**: encontramos un tipo de proyecto bastante peculiar, dado que se trata de proyectos en los que tenemos la solución pero todavía debemos buscar el problema.
  - Ejemplo un proyecto en el que queremos evaluar un producto existente para ver si cubre alguna de las necesidades de la organización.
- **Grupo 4**: típicamente, los proyectos de investigación y desarrollo, en los que debemos ser flexibles tanto respecto a la solución final que encontramos como respecto al problema que solucionamos, dado que, muchas veces, se acaba solucionando un problema diferente.
  - Ejemplo el invento del *Post It*.

Tres grandes categorías:

- Grupo 1: Ciclo de vida en cascada
- Grupo 2:
  - Los métodos iterativos e incrementales.
  - Los métodos ágiles.

4. Ciclo de vida en cascada

Ciclo de vida en cascada

Ciclo de vida en cascada o clásico

- Organización del desarrollo similar a una **cadena de producción** con trabajadores altamente especializados (analista funcional, arquitecto software, especialista en pruebas...)
- Los trabajadores producen artefactos que son consumidos por trabajadores de la cadena.
- Ejemplo de ciclo de vida: Métrica 3 Ministerio de Hacienda y Administraciones Públicas [2001].

Ventajas

- Sencillo de aplicar.
- Resulta apropiado para:
  - Desarrollar nuevas versiones de sistemas ya veteranos en los que el desconocimiento de las necesidades de los usuarios, o del entorno de operación no plantea riesgos
  - Sistemas pequeños, sin previsión de evolución a corto plazo

Inconvenientes

- Difícil aplicación por su rigidez:
  - Es necesario terminar por completo cada etapa para pasar a la siguiente.
  - Poco tolerante a cambios.
  - Los errores se propagan a las etapas siguientes.
  - Estimaciones y análisis puramente teóricos.

Lineal o secuencial

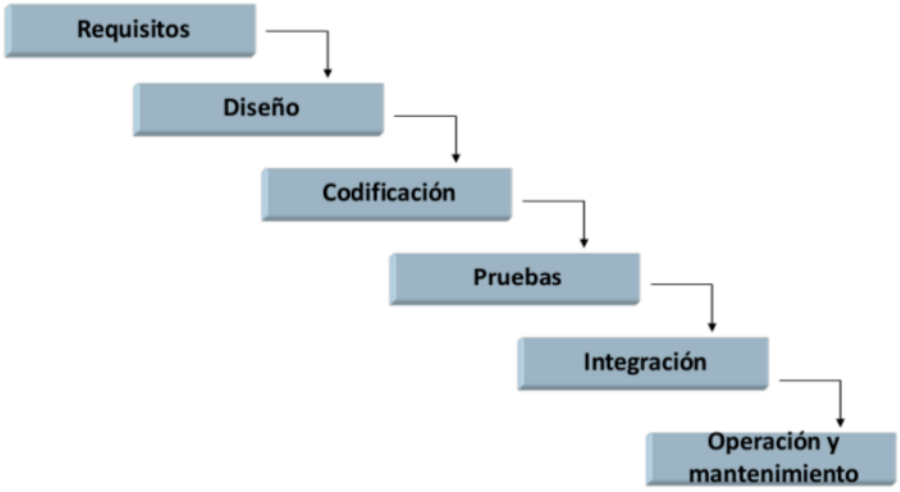

Figura 3: Modelo lineal o secuencial.

Etapas

El producto pasa, progresivamente, por las etapas siguientes:

Requisitos. ¿**Qué** debe ser el producto?:

- *Etapa crítica* para el proyecto (en esta metodología).
- Solucionado parcialmente con la revisión de requisitos (retroalimentación).

Análisis y diseño. ¿Cómo debe ser el producto? Análisis (punto de vista externo) y diseño (punto de vista interno definiendo componentes y relaciones entre ellos).

Implementación. Escribir el código, los manuales y generar el producto ejecutable según la fase anterior.

Pruebas. Se verifica, con usuarios finales, que el producto desarrollado se corresponda con los requisitos.

Mantenimiento. Se pone el sistema a disposición de todos los usuarios y se corrigen los defectos que se vayan encontrando.

Modelo en cascada

- Propuesto por Winston Royce en 1970.
- Inspirado en el modelo secuencial, pero añadiendo **flujos de retorno** o retroalimentación.
- Varias opciones de establecimiento de los flujos de retorno (ver figuras siguientes).
- La primera opción parece indicar que el retorno posible se da solamente entre una fase y la anterior

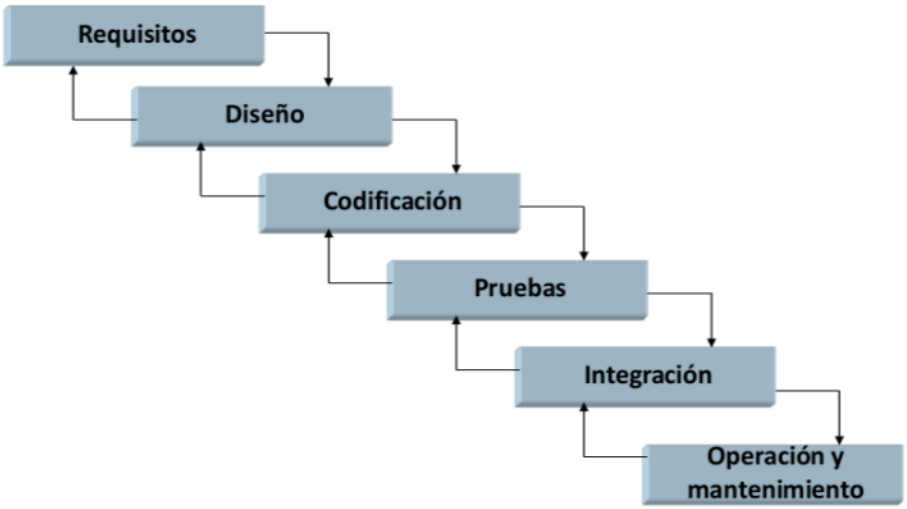

Figura 4: Modelo en cascada con retroalimentación a fase anterior.

- La segunda se refleja mejor el hecho de que en cualquier fase puede surgir un retorno para modificar cualquiera de las anteriores.
- Este modelo, como el anterior, reconoce la importancia de disponer de unos **requisitos** y un **diseño previo** antes de comenzar con la codificación del sistema, pero al mismo tiempo se enfrenta al hecho de que en la realidad la dificultad que supone disponer de documentación elaborada de requisitos y diseño antes de empezar a codificar **puede actuar como una barrera que bloquee el comienzo de la siguiente fase**.
- Algunos textos llaman "cascada" al modelo lineal, y "cascada modificada" al modelo de cascada.

Modelo en cascada con retroalimentación

5. Ciclo de vida iterativo e incremental

5.1. Bases modelo iterativo

Superación de limitaciones del ciclo en cascada

Rigidez y punto de vista sólo teórico

- El ciclo de vida en cascada tiene el inconveniente (especialmente para proyectos grandes) de que no tenemos ningún tipo de **información empírica** sobre el producto final hasta que no estamos a punto de finalizar el proyecto.

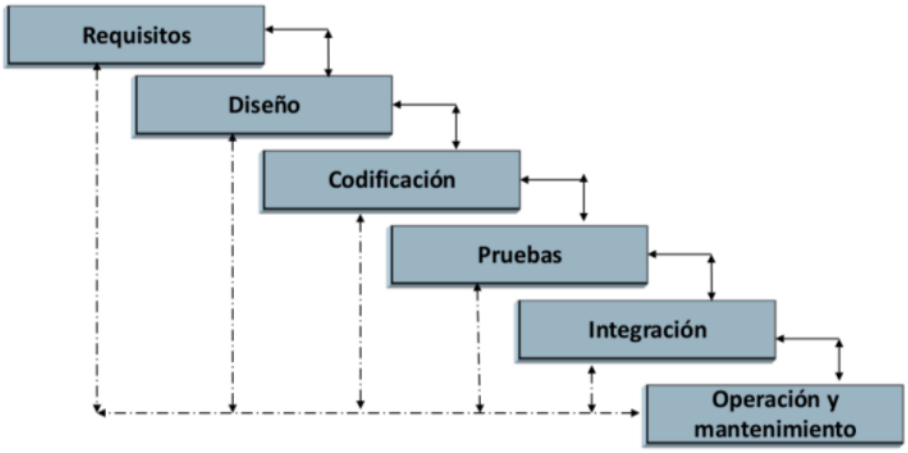

Figura 5: Modelo en cascada con retroalimentación a cualquier fase.

- El **conocimiento íntegro** que tenemos sobre la marcha del proyecto y sobre el producto final es **teórico**.
- El **coste de errores** en las primeras etapas es mucho mayor que en posteriores.
- Se necesitan métodos que permitan cambiar las ideas iniciales según avanza el proyecto.
- Los métodos iterativos facilitan descubrir la solución obteniendo información empírica.

Superación de limitaciones del ciclo en cascada

Resultados parciales funcionales

- Los métodos en cascada no producen un resultado tangible hasta el final.
- En proyectos largos esto supone una inversión sin recuperar ni parcialmente ésta.

Bases de los modelos iterativos

Principios

- El desarrollo se organiza en torno a **iteraciones** cuyo resultado es un **mini-proyecto/subsistema** que amplía el resultado de la iteración anterior. Cada **iteración** cubre todas las actividades del desarrollo (requisitos, análisis, Implementación y pruebas) $\Rightarrow$ **Retroalimentación**

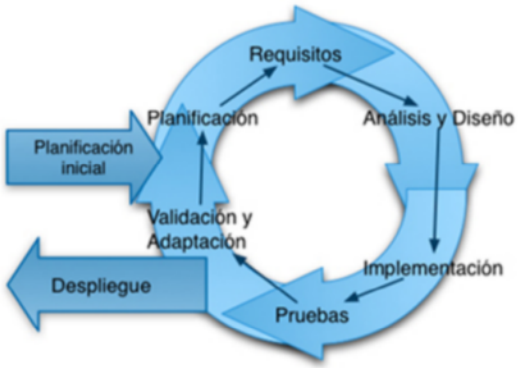

Figura 6: Esquema general de los modelos iterativos

- Al final de la iteración debe haber un **resultado utilizable** ⇒ en cualquier momento se puede decidir usar el sistema.

Esquema ciclo iterativo

Modelo incremental y modelo evolutivo

Algunos autores hablan de **modelo incremental** y **modelo evolutivo**:

- El **modelo incremental** tiene como resultado de cada iteración **subsistemas** que en conjunto formarán el **sistema** final añadiendo valor.
- El **modelo evolutivo** tiene como resultado de cada iteración un **sistema** funcional.

En ambos casos las iteraciones pueden solaparse o no, si hay dependencias entre subsistemas. Las organización de las etapas de cada iteración puede ser de varios tipos (cascada, espiral...).

Modelo incremental

- El usuario dispone de **pequeños subsistemas operativos**.
- Entregas parciales en periodos cortos de tiempo, incorporación de nuevos requisitos.

Resulta apropiado:

- Desarrollo de sistemas en los que el cliente necesita disponer de parte de la funcionalidad antes de lo que costaría desarrollar el sistema completo.

Figura 7: Modelo incremental: subsistemas.

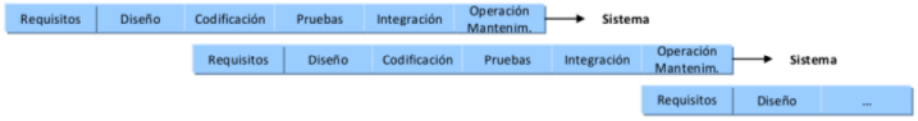

Figura 8: Modelo evolutivo.

- Desarrollo de sistemas en los que por razones del contexto interesa realizar la obtención de los requisitos de forma escalonada a través de subsistemas

Modelo evolutivo

- Compuesto por varios ciclos de desarrollo. Cada uno de ellos produce un **sistema completo** con el que se operará **en el entorno de operación**.
- Información acumulada para la mejora de etapas.
- Ciclo común a sistemas que se mejoran con versiones sucesivas.

Resulta apropiado:

- Desconocimiento inicial de todas las necesidades operativas.
- Necesidad de que el sistema entre en operación en tiempos inferiores a los que serían necesarios para diseñarlo y elaborarlo de forma exhaustiva.
- Necesidad de desarrollar sistemas en entornos cambiantes.

Aclaraciones sobre iteraciones

- Una iteración dura entra 1-6 semanas.
- En la iteración **se trabajan todas las fases** (planificación, requisitos, análisis y diseño, implementación, pruebas y validación).

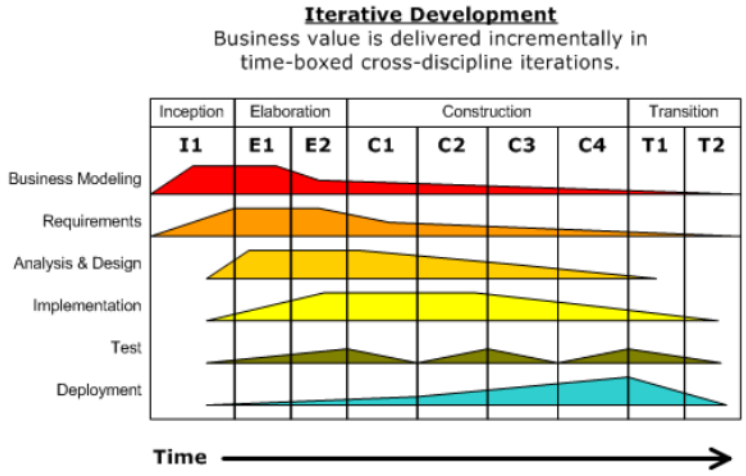

Figura 9: Etapas del proceso Unificado de desarrollo, fuente Wikipedia [2014]

- Según avanza el proyecto el **énfasis en las fases irá variando** de izquierda a derecha (ver siguiente figura).
- *Posible peligro: aplicar modelo en cascada asignando sus etapas a iteraciones.*
- Al final de cada iteración se produce un incremento en el volumen de funcionalidades.
- Facilita la planificación centrada en el cliente y en los riesgos (priorizar funcionalidades).

Etapas del proceso Unificado

Manifiesto Ágil

Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:

- *A los individuos y su interacción, por encima de los procesos y las herramientas.*
- *El software que funciona, por encima de la documentación exhaustiva.*
- *La colaboración con el cliente, por encima de la negociación contractual.*
- *La respuesta al cambio, por encima del seguimiento de un plan.*

Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda.

Fuente: `http://agilemanifesto.org/iso/es/`

Métodos ágiles respecto IS

- Consideran que las **personas** que participan en un proyecto (y la manera como **interactúen**) son más decisivas para el éxito que los procesos que se aplican o las herramientas que usan.
- Suelen ser métodos **poco estrictos en cuanto a los procesos** que se deben seguir y los artefactos que se han de generar, por lo que se denominan también *métodos ligeros*.

Desde el punto de vista de la ingeniería del software, este principio parecería ir en contra del enfoque sistemático, dado que asume que un proceso, por muy definido que esté, siempre dependerá de las personas y, por lo tanto, limita la transferibilidad del conocimiento entre diferentes ejecuciones del mismo proceso; sin embargo, **en la práctica ocurre que el proceso definido en los métodos ágiles se centra más en las personas y en sus interacciones que en los roles y en las tareas que éstas asumen**. **No se trata, pues, de no tener ningún proceso definido, sino de que el proceso se centre en la interacción entre personas.**

Los métodos ágiles son iterativos

La mayoría de los métodos ágiles son iterativos e incrementales
Estos principios encajan perfectamente con la filosofía del desarrollo iterativo e incremental, ya que los métodos iterativos e incrementales dan prioridad al software operativo (el producto final de cada iteración es una versión operativa del software) y facilitan la colaboración con el cliente y la respuesta al cambio, además permiten cambios en la planificación de las diferentes iteraciones

Filosofía *lean*

Lean Software Development: An Agile Toolkit Poppendieck and Poppendieck [2003] recoge una serie de principios muy interesantes y que pueden aclarar cómo aplicar de forma razonable diferentes metodologías (*lean* $\approx$ ligero, delgado...). Podríamos decir que lean engloba a las metodologías ágiles.

Principios *lean*:

1. **Evitar la producción innecesaria**. se trata de producir **sólo aquello que se necesita en la etapa siguiente**. Así, en vez de producir todos los requisitos antes de elaborar el análisis, se producen sólo los necesarios para poder hacer el análisis de esta iteración. En esta misma línea, también debemos concentrarnos en producir sólo los artefactos que realmente aportan valor; a grandes rasgos, lo podríamos resumir con la pregunta: "¿hay alguien esperando este artefacto?" o "¿alguien se quejará si este artefacto no está actualizado?". Sorprendentemente, muchas veces la respuesta a esta pregunta es no.

2. **Amplificar el aprendizaje**. Recopilar toda la información que se va generando sobre el producto y su producción (retroalimentación, iteraciones cortas, etc.) para entrar en un ciclo de mejora continua de la producción.

3. **Decidir lo más tarde posible**. Intentar tomar todas las decisiones con el máximo de información posible. Esto significa atrasar cualquier decisión hasta el punto de que no tomar la decisión resulta más caro que tomarla. Por ejemplo, si un estudiante puede decidir anular la convocatoria de un examen hasta tres días antes del examen, pero no sabe si tendrá tiempo de estudiar, debería tomar la decisión tres días antes del examen, dado que si la toma antes no tendrá tanta información sobre si lo puede aprobar o no como en aquel momento; pero si no la toma en aquel momento, ya será demasiado tarde para decidir.

4. **Entregar el producto cuanto antes mejor**. El proceso de desarrollo debe permitir implementar rápidamente los cambios y las funcionalidades que queramos añadir

5. **Dar poder al equipo**. Es importante que los equipos de personas que forman parte del desarrollo se sientan responsables del producto y, por lo tanto, hay que evitar que haya un gestor que tome todas las decisiones. El equipo de desarrollo es quien decide sobre las cuestiones técnicas y quien se responsabilizan de cumplir los plazos.

6. **Incorporar la integridad**. Desarrollar de forma que se pueda cambiar fácilmente a lo largo del tiempo.

7. **Visión global.** Evitar las optimizaciones parciales del proceso que pueden causar problemas en otras etapas y adoptar una visión global del sistema. Por ejemplo, si decidimos eliminar una funcionalidad por motivos técnicos, hay que tener en cuenta cómo puede afectar esto al valor del producto desde el punto de vista de la organización. Este principio va en contra de los trabajadores hiperespecializados que proponían el ciclo de vida en cascada.

5.2. Espiral

Concepto general

Espiral

- Este modelo, definido por Boehm en 1988, presenta un desarrollo evolutivo.
- También introduce como elemento distintivo la actividad de "**análisis de riesgo**" para guiar la evolución del proceso de desarrollo.

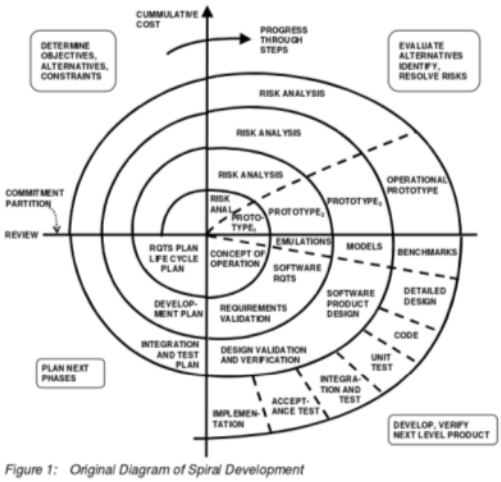

Figura 10: Ciclo espiral original, fuente Boehm [2000]

- El ciclo de **iteración** de este modelo evolutivo se convierte en una espiral, que al representarse sobre ejes cartesianos muestra en cada cuadrante una clase particular de actividad: Planificación, Análisis de riesgo, Ingeniería y Evaluación, que se suceden de forma consecutiva a lo largo del ciclo de vida del desarrollo. La dimensión angular representa el avance relativo en el desarrollo de las actividades de cada cuadrante. En cada ciclo de la espiral se realiza una parte del desarrollo total, a través de los cuatro tipos de actividades.
- En la **planificación** de cada vuelta se establece el contexto del desarrollo y se decide qué parte del mismo se abordará en el ciclo siguiente
- Las **actividades de análisis de riesgo** evalúan las alternativas posibles para la ejecución de la siguiente parte del desarrollo, seleccionando la más ventajosa y previendo los riesgos posibles
- Las **actividades de ingeniería** corresponden a las indicadas en los modelos lineales (secuencial y cascada): análisis, diseño, codificación, etc.
- Las actividades de evaluación analizan los resultados de la fase de ingeniería, tomando el resultado de la **evaluación como punto de partida para el análisis de la siguiente fase**.
- Este modelo permite múltiples combinaciones ya que **en la planificación de cada ciclo se determina el avance que se va a ejecutar durante la vuelta**. Éste puede consistir en la obtención y validación de requisitos, o en el desarrollo del diseño, o el diseño junto con la codificación, o en la obtención de un subsistema completo (cascada de requisitos – diseño – codificación – pruebas – integración).

5.3. Proceso Unificado de Desarrollo

Introducción al PU

- El proceso unificado (UP, *unified process*) fue propuesto por la empresa Rational Software (actualmente parte de IBM) con la intención de ser para los métodos de desarrollo de software lo que el UML fue para los lenguajes de modelización.
- **Uso extensivo de modelos** y la adopción de un **ciclo de vida iterativo e incremental**.
- UP es un **conjunto de procesos configurable** que se puede adaptar a diferentes contextos, de manera que no se aplica igual al desarrollo de una intranet de una empresa que al desarrollo de software militar.
- Varias variantes. Por ejemplo **OpenUP** Foundation [2014] asume los valores del desarrollo ágil y los tiene en cuenta a la hora de personalizar el proceso unificado.

Las mejores prácticas según UP

En términos generales todas las variantes del proceso unificado se basan en potenciar seis prácticas fundamentales (según el ***rational unified process: best practices for software development teams***):

1. Desarrollo iterativo e incremental.
2. Gestión de los requisitos (mediante casos de uso).
3. Arquitecturas basadas en componentes (módulos o subsistemas con una función clara).
4. Utilización de modelos visuales (mediante el lenguaje UML).
5. Verificación de la calidad del software (a lo largo de todas las etapas del proceso, no sólo al final).
6. Control de los cambios al software.

- Además, **UP promueve**:
  - Una **gestión proactiva de los riesgos** y sugiere que se construyan antes las partes que incluyan más riesgos.
  - Construcción de una **arquitectura ejecutable** durante las primeras etapas del proyecto
- El proceso se puede **descomponer de dos maneras**: en función del tiempo (en **fases**) y en función de las actividades o contenido (**actividades**).
- Define los **roles** que tienen las diferentes personas y **qué tareas** ha de llevar a cabo cada rol.

Figura 11: Hitos en el Proceso Unificado.

Fases del proyecto

A lo largo del tiempo (de manera secuencial):

1. **Inicio** (*inception*). Se analiza el proyecto desde el punto de vista de la organización y se determina el **ámbito**, la valoración de los recursos necesarios, la identificación de **riesgos** y de **casos de uso**.
2. **Elaboración** (*elaboration*). Se analiza el dominio del sistema que hay que desarrollar hasta tener un **conjunto de requisitos estable**, se **eliminan los riesgos principales** y se construye la **base de la arquitectura** (ésta será ejecutable).
3. **Construcción** (*construction*). Se desarrolla **el resto de la funcionalidad** (de manera secuencial o en paralelo) y se obtiene, como resultado, el producto y la documentación.
4. **Transición** (*transition*). Se pone el producto a disposición de la comunidad de **usuarios**. Esto incluye las pruebas beta, la migración de entornos, etc.

Hitos en el PU

1. **Objetivos**. Analiza el **coste** y los **beneficios** esperados del proyecto; debemos ser capaces de responder a dos preguntas: ¿estamos de acuerdo sobre el ámbito (qué debe hacer y qué no debe hacer el sistema) y los objetivos del proyecto? ¿Estamos de acuerdo sobre si vale la pena continuar?

   Para poder tomar estas decisiones, necesitaremos:

   - **Haber explorado previamente** cuál será la **funcionalidad** que hay que desarrollar, a quién puede afectar el nuevo sistema y quién tendrá que interactuar con él.
   - Tener una **idea aproximada** de cuál será la **solución final** y su viabilidad.
   - Estimación inicial del **coste** y del **calendario**, así como también de los riesgos que pueden afectar al desarrollo correcto del proyecto.

2. **Arquitectura**. Se alcanza **cuando ya tenemos una versión más o menos estable y final de la arquitectura**, ya hemos implementado las **funcionalidades más importantes** y hemos **eliminado** los principales **riesgos**. Plantearnos si la arquitectura que hemos definido nos servirá para llevar a cabo todo el proyecto, si consideramos que los riesgos que quedan están bajo control y si creemos que estamos añadiendo valor a buen ritmo.

   Para poder tomar estas decisiones, necesitaremos:

- Explorar los **requisitos con más detalle**, de manera que tengamos una estimación clara del coste pendiente del proyecto y de los problemas que nos podemos encontrar.
- Haber implementado un **número significativo de funcionalidades**.

3. **Capacidad operacional inicial**. ¿podemos decir que ya hemos implementado **toda la funcionalidad** que había que implementar?

4. **Entrega del producto**. Debemos decidir si hemos conseguido los **objetivos iniciales** del proyecto. En este caso, es importante que los clientes acepten el proyecto que hemos entregado.

Cada una de las fases incluye actividades que alcanzan todo el ciclo de desarrollo: desde los requisitos hasta la implementación y las pruebas. Por ejemplo, en la fase de inicio, la definición de la arquitectura incluye la creación de una versión ejecutable de ésta.

Ventajas y observaciones del PU

La ventaja principal de este cambio de filosofía respecto al ciclo de vida en cascada es que, con el proceso unificado, podemos **tomar las decisiones basándonos en información empírica** y no sólo en información teórica.

Es muy habitual ver equipos que creen estar siguiendo el proceso unificado pero que en realidad están siguiendo una variante del ciclo de vida en cascada.

Los objetivos del PU y cascada son similares, a grandes rasgos, pero la manera de lograrlos es muy diferente.

Actividades

- **Modelización de negocio** (*business modelling*). Esta actividad intenta solucionar el problema de comunicación entre los expertos en el dominio y los especialistas en tecnología que, habitualmente, usan lenguajes diferentes. Se generan los **casos de uso "de negocio"** (*business use cases*), describen los procesos principales de la organización y que servirán como lenguaje común para todos. Esta actividad es muy importante, ya que es la que nos asegurará que los desarrolladores entiendan cuál es la ensambladura del producto que están desarrollando dentro del contexto de la organización para la que lo están desarrollando.

- **Requisitos** (*requirements*). Describir **qué** debe hacer el sistema (y **que no** debe hacer). El modelo que se usa para describir la funcionalidad del sistema es el denominado **modelo de casos de uso**, que consiste en describir escenarios de uso del sistema mediante una secuencia de acontecimientos (el usuario hace X, el sistema hace Y, etc.).

- **Análisis y diseño** (*analysis & design*). Describir **cómo** se implementará el sistema. Se crean **modelos detallados de los componentes** que formarán el sistema. Estos modelos sirven como guía para los implementadores a la hora de escribir el código fuente de los programas. Estos modelos han de estar relacionados con los casos de uso y deben permitir introducir cambios en el supuesto de que los requisitos funcionales cambien (casos de uso conducen y la arquitectura guía).

- **Implementación** (*implementation*). Definir la **organización del código**, **escribirlo** y **verificar** que cada componente cumple los requisitos de manera **unitaria** (es decir, de manera aislada del resto de los componentes) y generar un sistema ejecutable. El proceso también prevé la **reutilización de componentes** que existan previamente.

- **Pruebas** (*test*). **Verificar la interacción entre los objetos y componentes** que forman el sistema. Dado que las pruebas se ejecutan durante todas las iteraciones, es más fácil detectar errores antes de que se propaguen por el sistema. Las pruebas han de incluir la fiabilidad, la funcionalidad y el rendimiento.

- **Despliegue** (*Deployment*). Producir las entregas del sistema y entregarlas **a los usuarios finales**. Incluye las tareas relativas a la **gestión de la configuración y de las versiones** y, sobre todo, tiene lugar **durante la fase de transición**.

Roles

OpenUP define los roles siguientes:

- **Cliente** o ***Stakeholder***. Cualquier persona que tenga un interés en el resultado final del proyecto.

- **Jefe de proyecto**. Encargado de la planificación y de coordinar la interacción entre los stakehoders. También es responsable de mantener a los miembros del equipo centrados en conseguir cumplir los objetivos del proyecto.

- **Analista**. Recoge la información de los *stakeholders* a modo de requisitos, los clasifica y los prioriza. En otras metodologías se le denomina *analista funcional*.

- **Arquitecto**. Define la arquitectura del software, lo que incluye tomar las decisiones clave que afectan a todo el diseño y a la implementación del sistema.

- **Desarrollador**. **Desarrolla una parte** del sistema, lo que incluye diseñarla de acuerdo con la arquitectura, prototipar (si es necesario) la interfaz gráfica de usuario, implementarla y probarla tanto aislada del resto del sistema como integrada con el resto de los componentes.

- **Experto en pruebas**. Identifica, define, implementa y lleva a cabo las pruebas aportando un punto de vista complementario al del desarrollador. Lo más habitual es que trabaje sobre el sistema construido y no sobre componentes aislados.

5.4. Metodologías ágiles: Scrum

Introducción a Scrum

A pesar de ser contemporáneo de UP, Scrum se popularizó posteriormente como parte del movimiento del desarrollo ágil. Es un método iterativo e incremental para desarrollar software elaborado por Ken Schwaber y Jeff Shuterland.

Se trata de un método muy ligero que, siguiendo el principio *lean* de **generar sólo los artefactos que aportan un valor importante**, minimiza el conjunto de prácticas y artefactos, así como también las tareas y los roles Palacio and Ruata [2011].

Roles

Scrum distingue, de entrada, entre dos grandes grupos de participantes de un proyecto: los que están **comprometidos en el desarrollo** (el equipo) y los que están **involucrados** pero no forman parte del equipo (lo que otros métodos denominan *stakeholders*).

Pollos y cerdos

Scrum denomina pollos a los *stakeholders* y cerdos al equipo. Estos nombres provienen de la siguiente historia: un pollo y un cerdo se encuentran y el pollo le pregunta: ¿montamos un restaurante?. El cerdo se lo piensa y le pregunta a su vez: ¿cómo lo llamaríamos?. El pollo contesta: huevos con tocino. A lo que el cerdo responde: No, gracias, porque yo estaría comprometido y tú sólo estarías involucrado.

Entre los miembros del equipo, Scrum define tres roles muy particulares:

- ***Scrum Master***. Es responsable de asegurar que el equipo sigue los valores, las prácticas y las reglas de Scrum. También se encarga de eliminar los impedimentos que el equipo se pueda encontrar dentro de la organización.
- ***Product owner***. Es la persona responsable de velar por los intereses de todos los *stakeholders* y llevar el proyecto a buen término. Es, por lo tanto, el encargado de decidir qué se implementa y qué es más prioritario.
- *Team*. El resto de los miembros del equipo son los desarrolladores. **No se especializan**, sino que todos deben tener, en mayor o menor medida, habilidades en todas las actividades implicadas. Los miembros del equipo **deciden ellos mismos cómo se organizan el trabajo y nadie (ni siquiera el Scrum Master) les puede decir cómo lo deben hacer**.

Artefactos

Scrum sólo define cuatro artefactos:

- ***Product backlog***. Es la lista de requisitos pendientes de implementar en el producto. Cada entrada (normalmente una "**historia de usuario**") tiene asociada una estimación del valor que aporta a la organización y también del coste de su desarrollo.
- ***Sprint backlog***. El *backlog* para una iteración concreta (Scrum denomina sprint a las iteraciones); más detallada que el *product backlog* y en la que cada historia de usuario se descompone en tareas de entre cuatro y dieciséis horas de duración. Los *sprints* tienen una duración máxima de 6 semanas, aunque se recomienda realizarlos en 2-3 semanas.
- ***Release burndown chart***. Gráfico que muestra el progreso actual del equipo en función del número de historias de usuario que faltan por implementar.
- ***Sprint burndown***. El *burndown* para una iteración concreta, en la que el progreso se puede medir en tareas finalizadas aunque no sean historias de usuario completas.

Prácticas

En cuanto a las prácticas, éstas se basan en dos ciclos de iteraciones: una iteración más larga (como las que podemos encontrar en métodos como UP) y una ïteración diaria".

- ***Sprint planning meeting***. Reunión que se organiza antes de empezar un sprint y en la que se deciden qué historias de usuario se implementarán en este sprint. Por lo tanto, se crea el sprint backlog.
- ***Daily scrum***. Reunión diaria en la que todos los miembros del equipo responden a tres preguntas: qué hicieron ayer, qué piensan hacer hoy y qué impedimentos se han encontrado que les han impedido avanzar. La finalidad de esta reunión es que todos los miembros del equipo estén enterados de lo que está haciendo el resto de los miembros y que así se identifiquen oportunidades de ayudarse unos a otros.
- ***Sprint review meeting***. Al finalizar un sprint se revisa el trabajo realizado y se enseña a quien esté interesado el resultado implementado (la demo).
- ***Sprint retrospective***. Sirve para reflexionar sobre lo que haya pasado durante el sprint y para identificar oportunidades de mejora en el proceso de desarrollo.

6. Conclusiones

Conclusiones

Creación del Modelo del Ciclo de Vida adaptado

Al iniciar el proyecto, el responsable de la arquitectura de procesos debe realizar los siguientes pasos:

- Análisis de las **circunstancias ambientales** del proyecto.
- Diseño del **modelo específico de ciclo de vida** para el proyecto (sobre las bases de los diseños más apropiados, para el desarrollo y la evolución del sistema de software).
- Mapeo de las **actividades** sobre el modelo.
- Desarrollo del **plan para la gestión** del ciclo de vida del proyecto

Debe considerar aspectos como:

- Posibilidad de descomposición del sistema en subsistemas de software, con agendas y entregas diferenciadas
- Estabilidad esperada de los requisitos
- Novedad del proceso o procesos gestionados por el sistema en el entorno del cliente.
- Criticidad de las agendas y presupuestos
- Grado de complejidad del interfaz de operación, criticidad de la usuabilidad
- Grado de conocimiento y familiaridad con el entorno de desarrollo, componentes externos empleados, etc.

Fuentes

El contenido de estos apuntes se basa a su vez en otros apuntes Miquel and Martos [2010] y Ruiz [2013].

Para ampliar información sobre UP se recomienda consultar, además de las referencias, los apuntes del Tema 2 de la Prof. Irene T. Luque Ruiz [2013].

Bibliografía y enlaces

Referencias

B. Boehm. Spiral development: Experience, principles, and refinements. Technical Report CMU/SEI-2000-SR-008, Software Engineering Institute, Pittsburgh, PA, 2000. URL http://www.sei.cmu.edu/reports/00sr008.pdf.

T. E. Foundation. *OpenUP*, 2014. URL http://epf.eclipse.org/wikis/openup/.

Ministerio de Hacienda y Administraciones Públicas. *Métrica v.3. Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información*, 2001. URL http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.VE7FgZvgz0o.

J. Palacio and C. Ruata. Scrum Manager Gestión de Proyectos, 2011. URL http://www.scrummanager.net/bok/index.php?title=Main_Page. version 1.4.

M. Poppendieck and M. A. Cusumano. Lean software development: A tutorial. *IEEE Software*, 29(5):26–32, 2012. ISSN 0740-7459. doi: http://doi.ieeecomputersociety.org/10.1109/MS.2012.107.

M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional, 2003. ISBN 978-0321150783.

J. Pradel i Miquel and J. A. R. Martos. *Introducción a la ingeniería del software. Asignatura Ingeniería de ingeniería del Software*. Universitat Oberta de Catalunya, 2010. FUOC PID_00171152.

I. T. L. Ruiz. Apuntes de la Asignatura Ingeniería del Software. Tema 2: El proceso de desarrollo del software. Paradigmas o modelos de desarrollo del Software. 2013.

Wikipedia. Unified Process — Wikipedia, The Free Encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Unified_Process&oldid=624908063. [Online; accessed 28-October-2014].

R. K. Wysocki. *Effective Project Management: Traditional, Agile, Extreme, 5th Edition*. Wiley, 2009. ISBN 978-0-470-42367-7.