Diseño y metodologías de desarrollo

por Fernando Dodino

Marzo 2013

Índice

1 Objetivo del apunte

Advertimos al lector que la idea es ofrecer una breve introducción a las metodologías de desarrollo de software, y en especial entender su relación con el diseño de sistemas para cubrir los contenidos mínimos de la materia. En otras materias se podrá estudiar los aspectos económicos, sociales, de liderazgo y motivacionales que cada proceso de desarrollo propone, se deja un conjunto de links para que los interesados investiguen.

2 Introducción a las metodologías de desarrollo de software

2.1 ¿Para qué sirve una metodología?

Una metodología nos ordena, nos contiene, nos permite definir límites. Construir software complejo requiere un gran esfuerzo: tecnología, dinero y sobre todo: personas. Personas que interactúan entre sí, con diferentes grados de conocimiento, con diferentes roles, con diferentes intereses. Una metodología propone un esquema de trabajo que nos permite entender cuál es nuestro rol dentro del proyecto, nos acerca una cierta sensación de tranquilidad, de seguridad. Sin un proceso no sabemos cómo comenzar y cuándo terminar.

Una metodología describe una estrategia para encarar un proyecto de sistemas y para ello define[1]:

- El conjunto de **actividades o tareas** que se consideran necesarias para que el proyecto sea exitoso.
- Un conjunto de **roles** que describen cómo se reparten las responsabilidades y las actividades entre las personas que participen del proyecto.
- El conjunto de **entregables o artefactos**, que son el resultado del trabajo en cada actividad, ya sean parte del producto final o elementos intermedios que permiten la comunicación y coordinación entre diferentes actividades o roles.
- Un **ciclo de vida** que describe cómo se organizan las diferentes actividades a lo largo del tiempo.
- Un conjunto de **procesos** que describen la forma en que se toman decisiones, se distribuyen las actividades entre las personas, se manejan los riesgos, etc.

Cada metodología propone diferentes **concepciones** sobre el desarrollo de sistemas, diferentes visiones sobre cómo alcanzar los objetivos del proyecto. No se puede aprovechar

[1] Tener en cuenta que las diferentes metodologías proponen diferentes nombres para estos conceptos. Lo que en algún lado se llama “rol” en otro toma el nombre de “trabajador”, etc.

correctamente las herramientas propuestas por una metodología si uno intenta usarla desde una concepción distinta a la que la metodología propone.

Dado que nuestro foco es el diseño, nos interesa entender cómo las metodologías influyen en el proceso de diseño, las actividades relacionadas con el diseño, así como los roles y entregables relacionados con el proceso de diseño. Y como diseñar tiene que ver con tomar decisiones, es interesante pensar cómo la metodología nos ayuda en este sentido:

- ¿Cómo minimizar la posibilidad de cometer errores, reducir riesgos?
- ¿Cómo validar las decisiones que tomamos, detectar errores, solucionarlos?

2.2 Objetivos de una metodología

El objetivo de un proyecto de desarrollo es producir y poner en funcionamiento un sistema que permita resolver las necesidades del cliente, por ejemplo:

- Que esté enfocado en la problemática real del cliente
- Que sea usable por el cliente
- Que se termine en una cantidad de tiempo adecuada
- Que tenga un costo adecuado
- Que minimice la ocurrencia de errores (funcionales o no funcionales)
- Que se pueda mantener en el tiempo, que sea fácil de modificar, de corregir, etc.
- Que minimice la posibilidad de fracaso del proyecto ante los riesgos.

Cabe destacar que algunos de estos objetivos pueden competir entre sí. Por ejemplo, un sistema que se construya garantizando 100% de ausencia de fallas puede resultar más costoso que uno de menor calidad, y no todos los clientes pueden estar dispuestos a pagar el costo adicional. Si el sistema es de misión crítica, como un sistema médico o uno que controla un avión, se pagará el costo que sea necesario porque no toleramos la presencia de fallas, mientras que en otros sistemas el costo de garantizar la ausencia de fallas puede resultar demasiado alto.

Por otro lado, para un sistema que se piense usar durante mucho tiempo será vital la mantenibilidad, en cambio para un sistema que se use por pocos días (como una promoción puntual en un comercio, o un sistema para una elección) se preferirá minimizar costos y tiempos aún a costa de sacrificar mantenibilidad.

Dado que los objetivos pueden competir entre sí, debemos establecer **prioridades** entre ellos. La mejor metodología será aquella que nos ayude a conseguir los de cada proyecto con sus prioridades particulares. Es decir que no hay una única “metodología de sistemas” ni una única forma adecuada de hacer las cosas, hay diferentes visiones y estrategias, que pueden ser útiles en diferentes situaciones. Incluso muchas de las metodologías modernas no se describen como un algoritmo cerrado que se deba seguir ciegamente, sino como **marcos de trabajo metodológicos**, que deben ser adaptados a cada proyecto puntual.

3 Distintos tipos de metodologías

3.1 Metodologías secuenciales / iterativas

En las metodologías secuenciales, el proceso de desarrollo de software se divide en varios pasos o fases. Cada fase tiene un conjunto de metas a cumplir. El fin de cada fase delimita el comienzo de la fase siguiente. Aunque son normales la superposición de fases, estas metodologías proponen una gran fase de análisis de requerimientos, otra de diseño, otra de construcción y otra de pruebas donde el alcance de cada fase es la totalidad de los requerimientos de un proyecto.

En las metodologías iterativas se divide el proyecto en entregas o iteraciones. Si cada iteración define un conjunto de metas a cumplir, podríamos pensar que no hay una gran diferencia con la metodología secuencial. No obstante, cada iteración define como entregable un software testeable por el usuario. Entonces hay etapas de análisis de requerimientos, diseño, construcción y prueba en cada iteración. Además, cada iteración permite revisar y cambiar los requerimientos a resolverse.

3.2 Metodologías orientadas al proceso / a las personas

Otra taxonomía que divide las metodologías es el grado de importancia que le dan

- al proceso de desarrollo
- y a las personas que ejecutan ese plan

Si bien la mayoría de las metodologías contemplan tanto la serie de pasos que conforman el proceso como qué tipo de tareas deben desarrollar las personas (en base a sus perfiles), hay metodologías en las que el proceso está por encima de las personas. Dicho de otra manera, "respetar el proceso garantiza el éxito del proyecto". El margen de discrecionalidad (cuánto puedo salirme del libreto) es mínimo, sólo en lo operacional (en el día a día). Por eso es importante para estas metodologías poder medir cada tarea del proyecto, sea un ejecutable o documentación.

Por el contrario, las metodologías orientadas a las personas consideran que éstas definen el éxito o fracaso de un proyecto. Les asignan un grado mayor de decisión en cada tarea y confían en su capacidad de resolución de un problema antes que en las métricas que dan los indicadores. Esto no quiere decir que el proceso no importe, sino que ocupa un puesto de menor relevancia en la consecución de un logro.

3.3 Metodologías orientadas a la documentación / al producto

Hay metodologías que sostienen que un producto de software bien elaborado nace de una

documentación extensa y que contemple todas las decisiones que surgieron del análisis y del diseño. De esa manera el desarrollador no tendrá dudas ni excusas a la hora de escribir cada línea de código.

Por el contrario, hay metodologías que privilegian tener un software testeable para el usuario antes que tener el documento que respalde ese software que está corriendo. Esto no significa que haya que programar sin tener una especificación, sino que:

- tomamos una documentación que puede tener definiciones pendientes, estar incompleta en su diseño o que sepamos que esté sujeta a cambios
- construimos el software
- y luego actualizamos las decisiones principales en el documento

con la ventaja de tener la certeza de que lo que hace el sistema es eso.

3.4 Metodologías predictivas / adaptativas

¿Qué ocurre con los cambios que piden los usuarios mientras se va construyendo el software? ¿Cómo se manejan?

- algunas metodologías creen que es posible anticipar los cambios a través de un buen análisis y un buen diseño que contemple diferentes alternativas. Este enfoque no es inocente, sabe perfectamente que el usuario puede cambiar de opinión, que las disposiciones legales e impositivas sufren modificaciones y que los proyectos están siempre sujetos a vaivenes políticos. Pero justamente por eso busca minimizar los cambios para conservar lo más estable posible el entorno: resistirse al cambio es su naturaleza.
- otras metodologías consideran que el cambio es inevitable, que no tiene sentido resistirse a él. De manera que el proceso mismo contempla momentos en los que el usuario puede modificar los requerimientos: esto implica agregar nuevos, descartar otros o modificarlos (no importa si ya fueron construidos o no). "El usuario tiene derecho a cambiar de opinión", sostienen.

4 Algunos ejemplos

4.1 Modelo de Cascada

Originado en el paper de Winston Royce: "Managing the development of large software systems"[2] pareciera ordenar el desarrollo de software en las conocidas fases de Relevamiento (de requerimientos de sistema y de software), Análisis, Diseño, Codificación, Prueba y Mantenimiento:

[2] http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

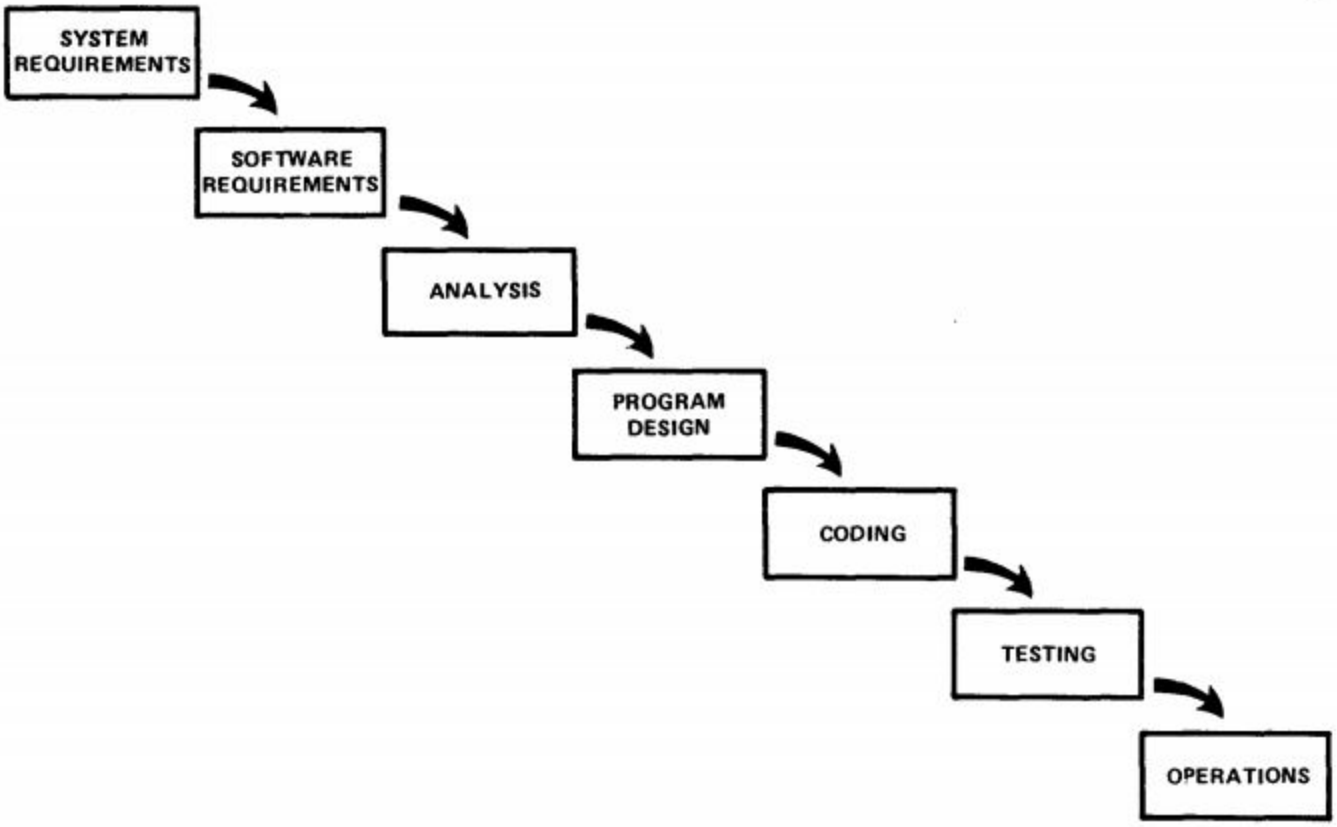


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

I believe in this concept, but the implementation described above is risky and invites failure. The

Este paper, no obstante, planteó en sus orígenes la necesidad de solapar etapas o bien de adelantar tan pronto como sea posible el diseño sin necesidad de esperar a analizar todos los requerimientos.

Algunas cuestiones adicionales del paper que conviene leer:

- la metodología propuesta es fuertemente orientada a la documentación[3] : hay que entender el contexto informal en el que se desarrollaban las aplicaciones de aquel entonces (mediados de los 70).
- Sobre el rol que debe tener el cliente: “Involve the customer - the involvement should be formal, in-depth and continuing”
- Sobre las pruebas: “Test every logic path in the computer program at least once with some kind of numerical check. If I were a customer, I would not accept delivery until this

[3] "how much documentation? My own view is 'quite a lot'; certainly more than most programmers, analysts, or program designers are willing to do if left to their own devices", "If the documentation is in serious default my first recommendation is simple: replace project management. [...] Stop all activities not related to documentation"

> procedure was completed and certified. (...) While this test procedure sounds simple, for a large, complex computer program it is relatively difficult to plow through every logic path with controlled values of input. In fact there are those who will argue that it is very nearly impossible. In spite of this I would persist in my recommendation that every logic path be subjected to at least one authentic check. “

Para desenmascarar que el paper de Royce es en realidad la semilla fundacional de las metodologías iterativas[4], vale transcribir el párrafo siguiente:

“STEP 3: DO IT TWICE
(...) If the computer program in question is being developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned. Figure 7 illustrates how this might be carried out by means of a simulation. Note that it is simply the entire process done in miniature, to a time scale that is relatively small with respect to the overall effort. The nature of this effort can vary widely depending primarily on the overall time scale and the nature of the critical problem areas to be modeled. If the effort runs 30 months then this early development of a pilot model might be scheduled for 10 months. For this schedule, fairly formal controls, documentation procedures, etc., can be utilized. If, however, the overall effort were reduced to 12 months, then the pilot effort could be compressed to three months perhaps, in order to gain sufficient leverage on the mainline development. In this case a very special kind of broad competence is required on the part of the personnel involved. “

[4] Recomendamos ver la conferencia que dictó Glenn Vandenburg en Lone Star Ruby Conference de 2010: http://confreaks.com/videos/282%C2%ADlsrc2010%C2%ADreal%C2%ADsoftware%C2%ADengineering. Allí se explica que la asociación del trabajo de Royce con una metodología en cascada proviene de tomar en cuenta sólo las primeras páginas omitiendo toda la explicación que justifica por qué no debería hacerse una sola pasada por cada fase del proyecto. Esta tergiversación continúa hasta nuestros días...

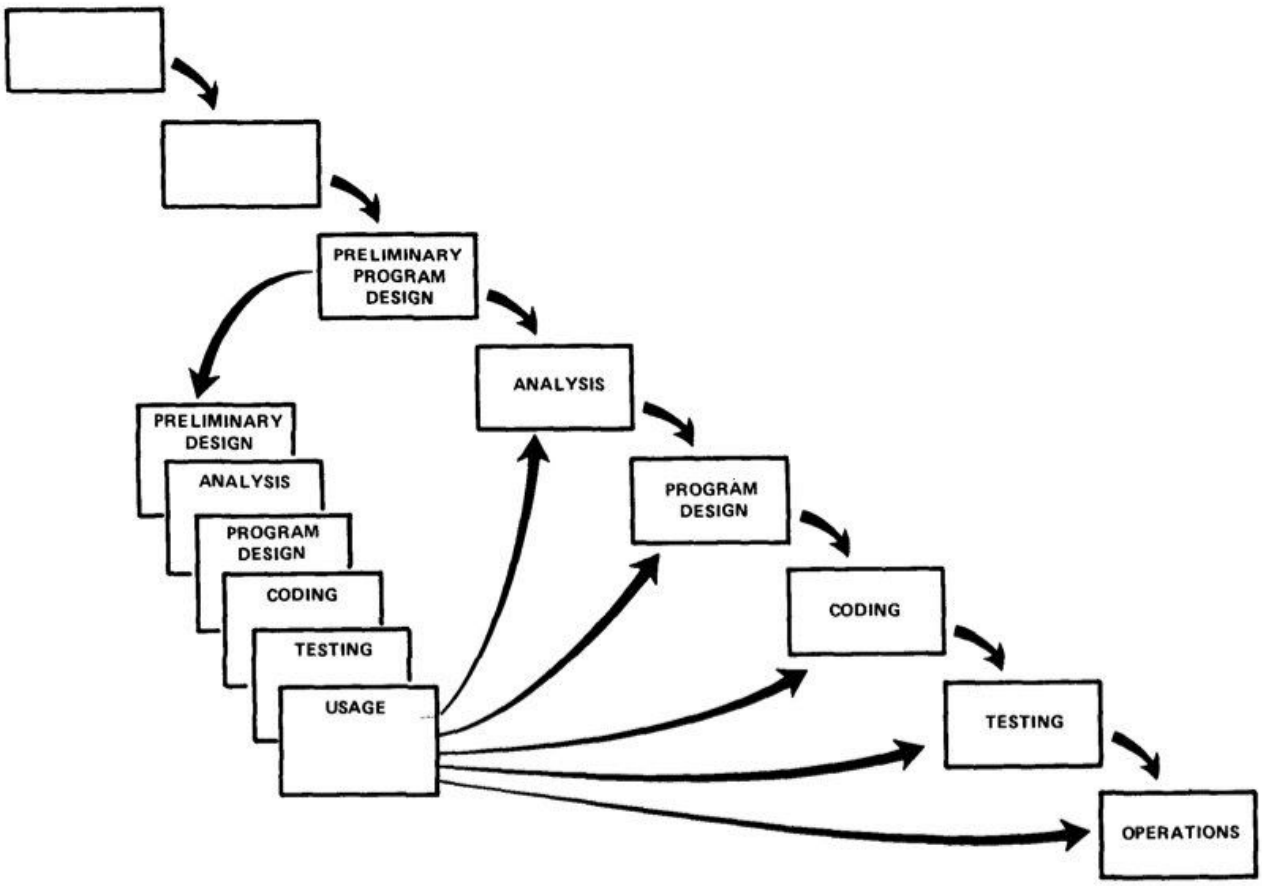


Figura 7: Attempt to do the job twice - the first result provides an early simulation of the final product.

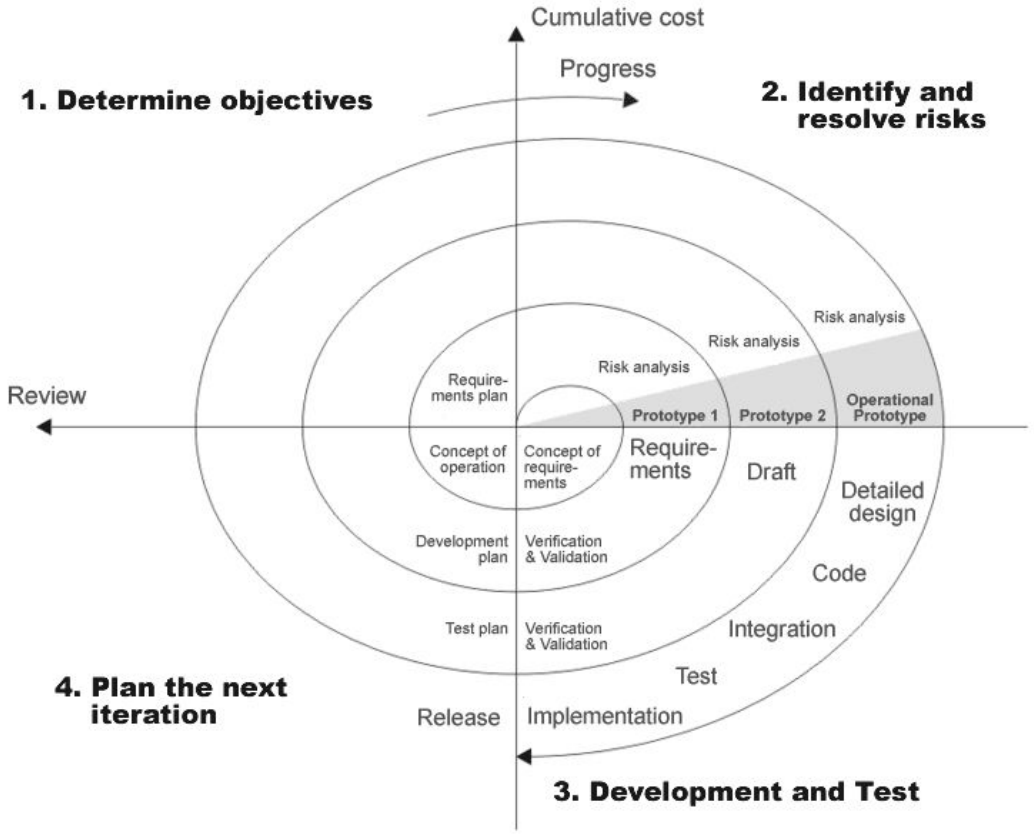
4.2 Espiral de Boehm

Originado en el paper de Barry Boehm: "A Spiral Model of Software Development and Enhancement", surge como primer contracara del modelo de cascada. Sigue un modelo de planificación de objetivos, identificación de riesgos y desarrollo y verificación del producto en *n iteraciones*,

- esto lo convierte en adaptativo (contempla cambios acomodándolos en el plan de las sucesivas iteraciones). Tomamos prestada una frase del artículo donde sostiene la importancia de que el proceso no fuerce al diseñador a tomar decisiones en forma anticipada: "[The spiral approach] fosters the development of specifications that are not necessarily uniform, exhaustive, or formal, in that they defer detailed elaboration of low-risk software elements and avoid unnecesary breakage in their design until the high-risk elements of the design are stabilized"[5].
- El modelo de espiral tiene un contenido altamente orientado al proceso (en el artículo no hay una sola mención sobre el papel que juegan las personas dentro del proyecto)
- No podemos afirmar que sea una metodología orientada al producto ni a la

[5] Aquí se propone la idea más innovadora que introduce esta metodología: priorizo definir primero los componentes de software que más riesgo tienen para el proyecto

documentación, en cada iteración debemos armar el análisis de riesgo, relevar los requerimientos de sistema y de software, hacer el plan de la iteración siguiente, el diseño global y detallado y los casos de prueba, pero por otra parte también la iteración contempla la posibilidad de prototipar y exige la presentación de un software testeable para el usuario.



A continuación trabajaremos sobre las dos metodologías que el mercado adoptó actualmente para soportar el proceso de desarrollo de software.

4.3 Proceso unificado

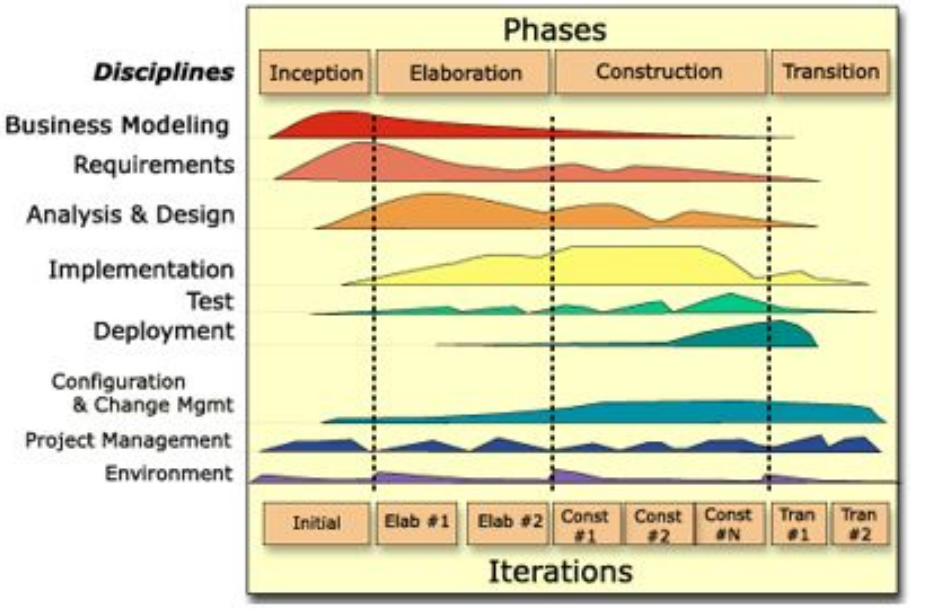
Comercializado por varias empresas, el proceso unificado es el primero que permite adaptarse según el tamaño del proyecto y su complejidad. Se autodefine como un proceso

- dirigido por los casos de uso: los requisitos funcionales se especifican, se diseñan y sobre ellos se construyen los casos de prueba
- centrado en la arquitectura, que es el soporte para poder resolver las necesidades del usuario: plataforma, componentes, requisitos no funcionales y producto
- **iterativo e incremental: "**Se divide el trabajo en partes más pequeñas o miniproyectos, donde cada proyecto es una iteración que resulta en un incremento. [...] En cada iteración, los desarrolladores identifican y especifican los casos de uso

relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes y verifican que los componentes satisfacen los casos de uso."[6]

- que busca anticipar los riesgos del proyecto.

Para eso se arma una matriz con dos ejes: fases del proyecto y actividades/disciplinas/flujos de trabajo. Las fases del proyecto determinan momentos que atraviesa el proyecto, las actividades se van solapando a lo largo del tiempo...



Cada fase es susceptible de dividirse en iteraciones y lo que diferencia una fase de otra es la madurez del producto que se está construyendo:

- en la fase de **inicio** se desarrolla el análisis del negocio que determina el alcance del sistema, tratamos de resolver o mitigar los riesgos que surjan pero también se hacen pequeñas pruebas de concepto (prototipos) que determinan su factibilidad. Dependiendo de la naturaleza de cada proyecto esta fase puede extenderse varias iteraciones, como ocurre en proyectos con una alta dosis de incertidumbre que requiere gran tiempo de investigación.
- en la fase de **elaboración** se trabajan en paralelo los requerimientos del sistema y la arquitectura que dará soporte a esos requerimientos: fíjense la disposición homogénea en la carga de trabajo de las actividades *captura de requisitos*, *diseño* e *implementación*. Por arquitectura se considera no solamente aspectos técnicos, plataforma y requisitos no-funcionales, sino también las necesidades de los usuarios.
- en la fase de **construcción** no sólo se desarrollan los casos de uso, sino que en las iteraciones se analizan (y se implementan) los cambios. Los autores del UP señalan: "el

[6] *El proceso unificado de desarrollo de software*, de Ivar Jacobson, Grady Booch & James Rumbaugh, Rational Software Corporation, Addison Wesley, Capítulo 1

énfasis se traslada de la acumulación del conocimiento básico necesario para construir el proyecto a la construcción propiamente dicha de un sistema o producto dentro de unos parámetros de costo, esfuerzo y agenda"[7]. En esta fase todavía podemos hacer cambios sobre la arquitectura, en base a los cambios que surjan durante el proceso de construcción.

- en la fase de **transición** se centra en implantar el producto en su entorno de operación, atravesando distintas etapas de prueba hasta la aceptación del usuario[8]. Es posible incorporar características al sistema que no hayan sido tenidas en cuenta en iteraciones anteriores, pero los cambios deben ser menores y no reformular el proyecto entero.

Como alerta el paper de Craig Larman[9], no debemos confundir Inception con Análisis, Elaboration con Diseño, Construction con Programación y Transition como Implementación. De la misma manera, al presentar las actividades que guían el flujo de trabajo fundamental, los autores del proceso unificado advierten:

"La descripción separada de los flujos de trabajo fundamentales, como estamos a punto de hacer, podría confundir al lector, y queremos estar seguros de que no suceda. En primer lugar, mediante la descripción separada de los flujos de trabajo uno detrás de otro, damos la impresión de que el proceso de desarrollo de software general, del comienzo al fin del proyecto, pasa por una secuencia de flujos de trabajo sólo una vez. Un lector podría llegar a pensar que los flujos de trabajo fundamentales son un proceso de una pasada, como el antiguo proceso en cascada. En segundo lugar, un lector descuidado podría concluir que cada flujo de trabajo fundamental es un paso monolítico en el proceso.

"Ninguna de estas impresiones es acertada. Describimos los flujos de trabajo en capítulos separados solamente como una forma de explicar por completo, por motivos pedagógicos, la totalidad del flujo de trabajo.

"En referencia al primer tema, la posibilidad de parecerse al ciclo de vida en cascada, recorremos los cinco flujos de trabajo[10] secuencialmente, pero lo hacemos una vez por cada iteración, no una vez para el proyecto completo. Por tanto, si tenemos siete iteraciones sobre cuatro fases, podríamos llevar a cabo los flujos de trabajo siete veces. Para ser más precisos, podríamos no utilizar los cinco flujos de trabajo al principio de la fase de inicio; es decir podríamos no llegar a los últimos flujos de trabajo, como la implementación y la prueba, en la primera iteración. El principio es claro: llevamos a cabo los flujos de trabajo de cada iteración mientras sea necesario para cada iteración en concreto.

"En referencia al segundo tema, el paso monolítico, describimos cada flujo de trabajo fundamental de forma bastante independiente de los otros. Sin embargo, hemos intentado

---

[7] Ivar Jacobson, Grady Booch, James Rumbaugh, *op.cit.*, pág.372

[8] Vale recalcar que la actividad de pruebas en la matriz de UP no comienza con esta fase, a partir de la fase de elaboración cada iteración concluye con una prueba unitaria y su integración con los demás módulos del sistema.

[9] How to Fail with the Rational Unified Process, de Craig Larman, Philippe Kruchten & Kurt Bittner, http://www.cs.unibo.it/~cianca/wwwpages/ids/letture/RUP.pdf

[10] Requisitos, análisis, diseño, implementación y pruebas según el modelo de Proceso Unificado, dado que el gráfico del proceso que se muestra corresponde a la versión comercial RUP de Rational Software Corporation

simplificar un poco cada flujo de trabajo centrándonos en sus actividades básicas, una vez más, por motivos pedagógicos. No hemos entrado en las alternancias con las actividades en otros flujos de trabajo. Por supuesto, estas alternancias son esenciales en un proceso de desarrollo de software iterativo [...]. Por ejemplo, mientras estamos trabajando en un determinado diseño, un desarrollador podría considerar deseable alternar entre los flujos de trabajo de análisis y diseño."

Análisis general del proceso unificado:

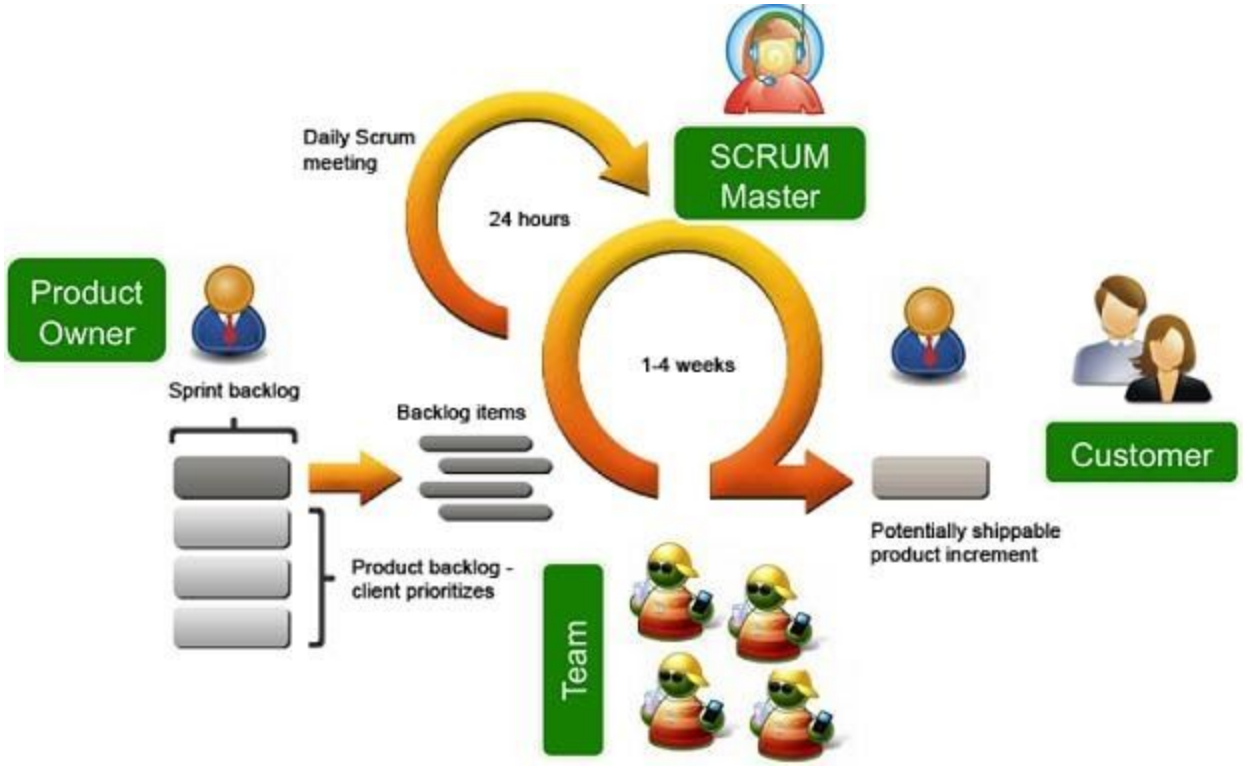
- Desde la fase de Elaboración hay construcción, test y deploy de un ejecutable, por eso su naturaleza es *iterativa*.
- De la misma manera, se evita tomar decisiones en forma prematura y se contempla la posibilidad de modificar los requerimientos hasta la última iteración, por lo que también encaja perfectamente en las metodologías adaptativas.
- UP sugiere una gran cantidad de artefactos de documentación, aunque podemos minimizarlos para concentrarnos en el código funcionando para el usuario. En la discusión filosófica documentación vs. producto podemos decir que UP está a mitad de camino de ambas, como lo confirma esta frase: "El producto no es sólo el ejecutable, también es la documentación que acompaña al sistema: manual de usuario, casos de uso, diagramas de clase, diagrama de arquitectura, etc."
- Respecto a la orientación persona o proceso podemos decir que UP se mantiene en una zona gris. Por un lado define que "las personas son importantes y tienen que saber qué hacer", mientras que al mismo tiempo cada persona cumple un rol que sugiere que esas personas están supeditadas a un proceso en el cual confiar.

4.4 Metodologías ágiles

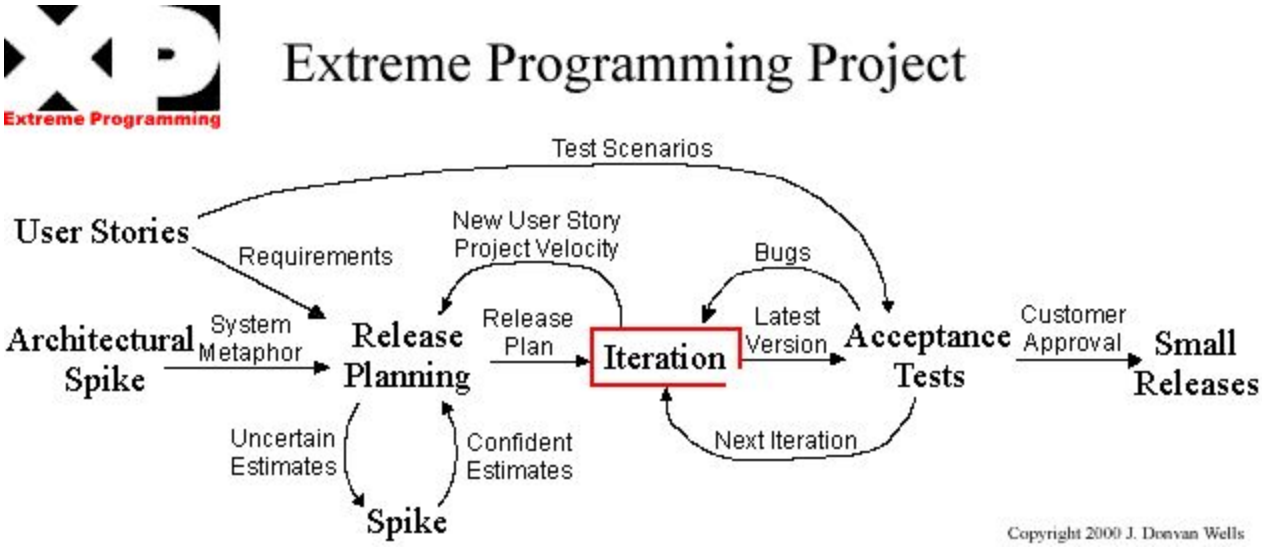
Como extreme Programming (XP), Scrum[11], Kanban, etc. todas se basan en el manifiesto ágil que establece estos principios metodológicos para desarrollar software:

- Las personas y cómo se relacionan entre sí son más importantes que los procesos y las herramientas tecnológicas (es orientado a las personas, por definición)
- Un producto funcionando es más importante que la documentación exhaustiva (orientado al producto)
- La colaboración del cliente es más importante que la negociación del contrato (que segrega al cliente a ser parte externa del producto)
- Responder al cambio es más importante que seguir el plan (adaptativo/iterativo)

[11] El lector puede ver el artículo de Jeff Sutherland y Ken Schwaber en http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum_Guide.pdf#zoom=100 basado en el artículo original de Hirotaka Takeuchi & Ikujiro Nonaka http://hbr.org/product/new-new-product-development-game/an/86116-PDF-ENG



Proceso de desarrollo - SCRUM



Proceso de desarrollo - XP

- Se acepta el cambio y el error como parte natural del proceso de desarrollo
- Se trabaja con dos tipos de planes, uno general que se asume incierto y otro a corto plazo, una **iteración** que define el plazo de un *entregable*
- Se define entre todos los participantes del proyecto cuándo se considera una tarea terminada (si debe incluir la aprobación de los casos de prueba del usuario, o el pasaje al ambiente productivo, o simplemente cuando el desarrollo esté listo).

- Mucho antes de iniciar las iteraciones, tenemos que asegurarnos de tener una arquitectura estable e identificar los riesgos (similar a la fase Inception de UP)
- Aunque las metodologías ágiles se asocian con cierta informalidad, se trabaja bastante con las métricas: cada historia de usuario tiene una prioridad y un estimado, la suma de todas las historias construidas en una iteración define un número que es la "velocidad" del equipo de trabajo. Esa velocidad debe ser sustentable a lo largo de todo el proyecto, no podemos forzar tiempos extra ni podemos construir tiempo. Esto ayuda a establecer estándares de calidad.
- El cliente forma parte del equipo de trabajo, con el cual se negocia más que confrontarlo (este punto constituye muchas veces un gran escollo)
- Se deja de paso el enfoque taylorista[12] de las personas cumpliendo un rol: se trabaja de a pares, intercambiando roles analista y programador. De esta manera no existe un analista funcional que sólo diseña y un programador que sólo codifica. Las metodologías ágiles asumen que cada persona que integra el proyecto es profesional, responsable y que los equipos pueden organizarse por sí mismos, sin necesidad de establecer relaciones jerárquicas de sumisión al poder.
- El proceso está entonces supeditado a lo que las personas profesionales y colaborativas logren en conjunto.

5 Diseño y metodología

¿Qué tiene que ver la metodología con el diseño? ¿En qué me cambia?

5.1 El papel del diseño en la metodología de desarrollo

Si la metodología de trabajo es secuencial, cada fase del proyecto tiene un comienzo y un fin específico. El diseño debe respetar el orden que le corresponde: después del análisis y antes de la programación / construcción. Si la metodología de trabajo es iterativa, esto implica diseñar en diferentes momentos, sin acotarlo a un período determinado.

De la misma manera,

- cuando la metodología es **orientada al proceso**, se genera una gran cantidad de documentación respaldatoria del diseño. El proceso es el eje central del proyecto, que guía al objetivo que se quiere alcanzar (en el diseño esto se concreta con un documento

[12] según Frederick W.Taylor, padre de la *Administración Científica*, "el hombre es, por naturaleza, perezoso e intenta escudarse en ello para realizar lentamente su trabajo haciendo creer al empresario que está dando lo mejor de sí. De ahí que se deben medir los tiempos y los movimientos de estos trabajadores para estudiarlos y encontrar la mejor combinación de movimientos musculares para elevar la producción y, también, dar uniformidad a los procesos [...] **era necesario dividir entre quienes piensan las mejores maneras de hacer el trabajo y quienes tienen las fortalezas físicas para ejecutarlo**, a los primeros se les daba la responsabilidad de adiestrar a los segundos hasta obtener de ellos el mayor rendimiento que su cuerpo pudiera dar. También es importante la especialización de tareas, pues de esta manera, el trabajador gana más tiempo y destreza haciendo lo mismo todos los días."

entregable que es la especificación). Si el proceso está bien definido sólo hay que controlar el avance de las tareas.
- cuando la metodología es **orientada al producto final**, nos interesa más dejar en claro las decisiones importantes que respalden el software que está corriendo. La interacción entre las personas es fundamental y termina definiendo el proceso de diseño, de la misma manera que el éxito o el fracaso del mismo.

5.2 Diseño anticipado y diseño iterativo

La metodología de desarrollo nos lleva a diseñar de dos formas completamente diferentes:
Las metodologías predictivas proponen el **diseño anticipado**, donde se asume que

- el análisis ya ha relevado todos los procesos que el usuario necesita
- tenemos disponible toda la información para poder definir cada proceso
- se minimizarán los cambios en los requerimientos hasta nuestra implementación
- si el diseño es adecuado, la codificación se ajustará perfectamente a lo que el usuario necesita.
    - para ello hay que documentar el sistema en su completitud para que los programadores no tengan que tomar decisiones de diseño en la codificación.
- en la fase de diseño no se debe programar, dado que se estaría solapando la actividad (de la misma manera que en la fase de codificación no se debe diseñar)

Por el contrario, las metodologías adaptativas proponen el **diseño iterativo**, donde se asume que

- sólo tenemos algunos procesos relevados, y aunque los tuviéramos en su totalidad, los requerimientos podrían cambiar.
- es inocente pensar en que se podrá minimizar los cambios en los requerimientos, dado que
    - el usuario no sabe exactamente lo que se va a construir y tiene derecho a pedir modificaciones cuando se da cuenta de que cometió un error al dar información al diseñador.
    - bajo la premisa anterior el diseñador no puede realizar un diseño que no esté sujeto a cambios, por los errores propios que además podría cometer.
- si el diseño no es adecuado, debemos cambiarlo lo más pronto posible. Esto incluye la fase de codificación.
- si queremos reflejar la realidad, tenemos que permitir que haya alternancia entre diseño y programación. No paralelizamos las actividades, sino que una se va solapando a la otra, como en una pila.
- el diseño iterativo considera que los errores son parte del desarrollo mismo y necesitamos poder modificar el diseño en cualquier momento, sin que eso paralice el proyecto (iterativo tiene mucho de "prueba y error").

5.3 Análisis comparativo

Los defensores del diseño anticipado sostienen que en el diseño iterativo se pierde el orden,

que es difícil coordinar un proyecto (no se sabe exactamente en qué porcentaje está cumplida cada actividad), y que el diseño iterativo confunde diseño y programación, al punto en el que en realidad sólo se programa.

Los defensores del diseño iterativo creen que todos los proyectos se construyen de esta manera, lo único que hacemos al utilizar esta metodología es reflejar lo que sucede en la realidad: los requerimientos cambian (aparecen nuevos, se modifican los existentes y algunos incluso desaparecen durante el proyecto), los diseñadores se equivocan, también lo hace el usuario y continuamente nos vemos obligados a adaptar nuestra planificación. Separar un proyecto en varias iteraciones facilita aceptar esos cambios, porque no se mantienen fijos los requerimientos ni los diseños, solamente mantenemos el plazo de entrega (lo que vamos a entregar está sujeto a cambios en cada iteración).

5.4 Pensar y hacer

Al hacer la comparación entre las metodologías orientadas al producto vs. las orientadas al proceso y al ver la enorme diferencia entre el diseño predictivo y el iterativo, se pone sobre la mesa una larga discusión: ¿es correcto asociar los tiempos de diseño y programación a los tiempos donde se piensa y donde finalmente se hace? ¿es tan taxativa la diferencia? ¿puedo abstraer los detalles técnicos para diseñar una solución? ¿debe ocurrir primero la documentación y después la construcción o son tareas que pueden solaparse? ¿es necesario contar con dos perfiles diferenciados, o sea, un analista funcional y otro programador? ¿no sería más sano tener un desarrollador senior con mayores capacidades de abstracción y otro con menos experiencia, que pueda formarse en la materia?

La metodología define la discusión filosófica de si nos interesan más las personas o los procesos. Asumir que el proceso tiene la prioridad presupone que sólo tengo recursos para asignar, y si divido perfiles en personas que piensan (diseñadores) de las que hacen o ejecutan (programadores) eso permite intercambiar gente sin mayores inconvenientes. Si pienso que las personas son importantes, que su grado de experiencia, la relación entre sí, la motivación personal y grupal, la capacidad de aprendizaje y la respuesta ante problemas que surgen son la clave de éxito de un proyecto, no nos interesa hacer la distinción entre pensar y hacer, porque esto ocurre todo el tiempo en forma simultánea, y mientras menos quiera disociar estos eventos (pensar y hacer) menos complicaciones tengo para encontrar un diseñador que sólo diseñe y un programador que sólo programe. Además el programador se transforma en una persona crítica del diseño, diseño que entonces puede mejorarse (es una metodología menos rígida en este sentido).

5.5 Integración de las actividades de diseño en el proceso de desarrollo

¿Qué actividades ocurren al diseñar?

- Interactuamos con el usuario para repreguntar o proponer alternativas a lo que él solicitó

- Interactuamos con el equipo de desarrollo, en donde pueden surgir
    - inconsistencias en las definiciones que no habíamos detectado antes: en el diseño iterativo este "inconveniente" está previsto, no así en el diseño anticipado en donde el equipo del proyecto debe asumir el costo de este imprevisto.
    - dificultades técnicas de implementación: esto suele ser más frecuente de lo que imaginamos, ya sea porque subestimamos la dificultad de un requerimiento, porque no tuvimos en cuenta algún factor tecnológico o porque los imprevistos suceden en todo proyecto.

Mientras que las metodologías secuenciales ven que el análisis condiciona el diseño y éste a su vez define las decisiones de implementación, es interesante notar que en los casos que mencionamos arriba es al revés: las cuestiones técnicas impactan sobre el diseño y el diseño puede hacer variar lo relevado en el análisis. De hecho, éste es el valor agregado de un buen diseñador: hacer las preguntas que disparen mejoras en lo que el usuario pide.

6 Resumen

Las características que tiene el diseño como actividad están fuertemente relacionadas con las definiciones que adopte la metodología de desarrollo del proyecto, en particular, si prevalecen las decisiones de las personas o si deben seguir lo que determine el proceso, si se tratan de atacar todas las decisiones de antemano o hay un espacio para tener en cuenta márgenes de error, qué peso tienen el producto y la documentación respaldatoria, etc.

7 preguntas para el lector curioso

1. Lea el paper Managing the development of large software systems de Winston Royce y conteste

1. ¿Cuál es el objetivo central del paper?
2. ¿Por qué decimos que en el paper puede hablarse de la naturaleza iterativa del desarrollo de software? Justifique.
3. Explique alguno de los cinco postulados del paper:
    a. Program design comes first.
    b. Document the design.
    c. Do it twice.
    d. Plan, control and monitor testing.
    e. Involve customer.

2. Lea el artículo A spiral model of Software Development and Enhacement de Barry Boehm y conteste

1. Explique cómo se desarrolla una aplicación según el modelo en espiral.
2. ¿Qué papel juega el análisis del riesgo en la metodología? Justifique.
3. ¿Cuáles son las fortalezas y debilidades del modelo en cascada según el paper?
4. ¿Cuáles son las fortalezas y debilidades del modelo en espiral según el paper?

3. Lea el artículo How to Fail with the Rational Unified Process, de Craig Larman, Philippe Kruchten & Kurt Bittner. Tenga en cuenta la naturaleza irónica del paper y conteste

1. Explique el concepto “creeping requirement” y qué quiere expresar la figura 1 de la página 3.
2. Explique qué quiere decir el autor con la frase “Superimpose Waterfall Thinking”
3. En la página 5 se explica que asumir esto es incorrecto:

“1. Inception—do most of the requirements
2. Elaboration—do the detailed design and models
3. Construction—implement
4. Transition—integration, system test, deployment”

Justifique por qué en base a lo que afirma el paper.

4. ¿Qué advertencia hace el autor sobre las iteraciones cortas o largas?
5. Indique cuál sería la estrategia apropiada de estimación para planificar un proyecto según el paper.
6. Justifique el grado de documentación que el autor propone para el proceso unificado.
7. Explique por qué es importante para el autor que personas con capacidades técnicas tomen decisiones de arquitectura.

8. ¿Cuáles son los consejos que da el autor para implementar el proceso unificado como una metodología realmente iterativa?

4. Lea el paper UP: Best Practices for Software Development Teams y conteste
   1. Explique brevemente qué se considera un artefacto UP.
   2. Explique en qué consiste alguna de estas actividades:
      a. Business modeling
      b. Requirements
      c. Analysis & Design
      d. Implementation
      e. Test
      f. Deployment
      g. Project Management
      h. Configuration and Change Management
      i. Environment
   3. Qué es un rol/worker en el proceso unificado y cómo es la relación persona física/worker.
   4. ¿Por qué decimos que el proceso unificado es iterativo? Demuéstrelo a partir del proceso que explica el paper.
   5. Detalle en qué consiste una iteración del proceso unificado.
   6. ¿Cuál es la estrategia que adopta UP ante el cambio en los requerimientos?
   7. Explique qué significa que UP es un proceso configurable (o framework metodológico) según el paper.
   8. Indique la relación que tienen la especificación UML y el proceso unificado.

5. Investigue el sitio http://www.extremeprogramming.org/ y explique las reglas que aplican sobre
   1. el diseño (designing)
   2. el gerenciamiento (management)
   3. la planificación del proyecto (planning)
   4. la implementación (coding)
   5. las pruebas (testing)

(elija un área de incumbencia)

6. Lea el paper Scrum Guides, de Jeff Sutherland y Ken Schwaber y conteste
   1. Explique el concepto de desarrollo a través de sprints.
   2. Describa cómo se conforma un equipo de trabajo en Scrum.
   3. Explique qué significan estos conceptos dentro de la metodología Scrum (proporcione un ejemplo de cada uno)
      a. Transparencia
      b. Adaptación
   4. Explique (y dé un ejemplo concreto) de los siguientes artefactos de Scrum:

a. Product Backlog
b. Sprint Backlog
c. Increment

7. Busque información sobre el Modelo en V y conteste
    1. Explique el proceso de desarrollo según esta metodología.
    2. Analice cómo es la actividad de diseño según esta metodología.