XUNTA DE GALICIA

ESCOLA GALEGA DE ADMINISTRACIÓN PÚBLICA

31. INGENIERÍA DEL SOFTWARE. PROCESO SOFTWARE, MODELOS DE PROCESO SOFTWARE. PROCESO UNIFICADO. CICLOS DE VIDA. MODELOS DE CICLO DE VIDA. FASES DEL CICLO DE VIDA. MODELOS DE DESARROLLO. MODELOS ÁGILES. METODOLOGÍAS DE DESARROLLO DE SOFTWARE. MÉTRICA VERSIÓN 3.

Fecha de actualización 26 de julio de 2011

Tema 31. Ingeniería del software. Proceso software, modelos de proceso software. Ciclo de vida. Modelos de ciclo de vida. Fases del ciclo de vida. Modelos de desarrollo. Modelos ágiles. Metodologías de desarrollo de software. Métrica versión 3

INDICE

31.6 Metodologías de desarrollo de sistemas de información.
31.7 Métrica Versión 3

31.1 Ingeniería del software

31.1.1 La crisis del software.

La rápida expansión de la Informática, sobre todo a partir de la segunda generación de ordenadores en los años 60, llevó a la escritura de millones de líneas de código antes de que se empezaran a plantear de manera seria metodologías para el diseño y la construcción de sistemas software y métodos para resolver los problemas de mantenimiento, fiabilidad, etc. Esta expansión sin control tuvo como consecuencia la denominada **crisis del software**, que es el nombre genérico que se ha acuñado para referirse a un conjunto de problemas que se han ido encontrando en el desarrollo del software. Esta problemática no sólo se limita al software que no funciona adecuadamente, sino que abarca otros aspectos como la forma de desarrollar el software, el mantenimiento de un volumen creciente de software existente y la forma de satisfacer la demanda creciente de software.

Los síntomas que hacen palpable la aparición de la crisis del software son, entre otros, los siguientes:

- **Expectativas**: los sistemas no responden a las expectativas que de ellos tienen los usuarios.
- **Fiabilidad:** los programas fallan demasiado a menudo**.**
- **Costo:** los costos del software son muy difíciles de prever y, frecuentemente, son muy superiores a lo esperado.
- **Plazos:** el software se suele entregar tarde y con menos prestaciones de las ofertadas.
- **Portabilidad:** es difícil cambiar un programa de su entorno hardware, aun cuando las tareas a realizar son las mismas**.**

- **Mantenimiento:** la modificación del software es una tarea costosa, compleja y propensa a errores.
- **Eficiencia**: los esfuerzos que se hacen para el desarrollo del software no hacen un aprovechamiento óptimo de los recursos disponibles (personas, tiempo, dinero, herramientas, etc.).

La solución a la crisis del software se centra, pues, en abordar y resolver los siguientes problemas principales:

- La planificación del proyecto software y la estimación de los costes de desarrollo, que son muy imprecisos.
- La productividad de las personas, que no se corresponde con la demanda de sus servicios.
- La calidad del producto software, que es, en muchos casos, inadecuada.

31.1.2 La Ingeniería del Software

La Ingeniería del Software se puede definir como *el establecimiento y uso de principios de ingeniería orientados a obtener, de manera económica, software que sea fiable y funcione eficientemente sobre máquinas reales*.

La Ingeniería del Software abarca tres elementos clave: métodos, herramientas y procedimientos. Los **métodos** proporcionan la manera de construir técnicamente el software. Abarcan las tareas de planificación y estimación de proyectos, análisis de los requerimientos del sistema y del software, diseño de las estructuras de datos, de la arquitectura de programas y de los procedimientos algorítmicos, y la codificación, pruebas y mantenimiento. Las **herramientas** suministran el soporte automático o semiautomático para los métodos; esto es, dan soporte al desarrollo del software. Los **procedimientos** definen la secuencia en la que se aplican los métodos, los controles que ayudan a asegurar la calidad y a coordinar los cambios y las guías que facilitan a los gestores del software.

El trabajo que se asocia a la ingeniería del software se puede dividir según Pressman en **tres fases genéricas**, con independencia del área de aplicación, tamaño o complejidad del proyecto:

- **La fase de definición** se centra sobre el *qué.* Es decir, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué comportamiento del sistema, qué interfaces van a ser establecidas, qué restricciones de diseño existen, y qué criterios de validación se necesitan para definir un sistema correcto. Por tanto, han de identificarse los requisitos clave del sistema y del software. Aunque los métodos aplicados durante la fase de definición variarán dependiendo del paradigma de ingeniería del software (o combinación de paradigmas) que se aplique, de alguna manera tendrán lugar tres tareas principales: ingeniería de sistemas o de información, planificación del proyecto del software y análisis de los requisitos.
- **La *fase de desarrollo*** se centra en el *cómo.* Es decir, durante el desarrollo un ingeniero del software intenta definir cómo han de diseñarse las estructuras de datos, cómo ha de implementarse la función dentro de una arquitectura de software, cómo han de implementarse los detalles procedimentales, cómo han de caracterizarse interfaces, cómo ha de traducirse el diseño en un lenguaje de programación (o lenguaje no procedimental) y cómo ha de realizarse la prueba. Los métodos aplicados durante la fase de desarrollo variarán, aunque siempre tendremos: diseño del software, generación de código y prueba del software.
- **La fase de mantenimiento** se centra en el *cambio* que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente. Durante la fase de mantenimiento se encuentran cuatro tipos de cambios:

- **Corrección.** Incluso llevando a cabo las mejores actividades de garantía de calidad, es muy probable que el cliente descubra los defectos en el software. El *mantenimiento correctivo* cambia el software para corregir los defectos.
- **Adaptación.** Con el paso del tiempo, es probable que cambie el entorno original para el que se desarrolló el software. El *mantenimiento adaptativo* produce modificaciones en el software para acomodarlo a los cambios de su entorno externo (hardware, sistema operativo, reglas de negocio,...).
- **Mejora.** Conforme se utilice el software, el cliente/usuario puede descubrir funciones adicionales que van a producir beneficios. El *mantenimiento perfectivo* lleva al software más allá de sus requisitos funcionales originales.
- **Prevención.** El software de computadora se deteriora debido al cambio. En esencia, el *mantenimiento preventivo* hace cambios en programas de computadora a fin de que se puedan corregir, adaptar y mejorar más fácilmente.

Estas fases se complementan con un número de ***actividades protectoras*** se aplican a lo largo de todo el proceso del software. Entre las actividades típicas de esta categoría se incluyen: seguimiento y control del proyecto de software, revisiones técnicas formales, garantía de calidad del software, gestión de configuración del software, preparación y producción de documentos, gestión de reutilización, mediciones y gestión de riesgos.

31.2 Proceso software. Modelos de proceso software

Según Sommerville, *un proceso del software es un conjunto de actividades que conducen a la creación de un producto software*. Estas actividades pueden consistir en el desarrollo de software desde cero en un lenguaje de programación estándar como Java o C. Sin embargo, cada vez más, se desarrolla nuevo software ampliando y modificando los sistemas existentes

y configurando e integrando software comercial o componentes del sistema.

Para Fugetta, un proceso software es *un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software*.

Los procesos del software son complejos y, como todos los procesos intelectuales y creativos, dependen de las personas que toman decisiones y juicios. Debido a la necesidad de juzgar y crear, los intentos para automatizar estos procesos han tenido un éxito limitado. Las herramientas de ingeniería del software asistida por computadora (CASE) pueden ayudar a algunas actividades del proceso, pero tienen limitaciones. Una razón por la cual la eficacia de las herramientas CASE está limitada se halla en la inmensa diversidad de procesos del software. No existe un proceso ideal, y muchas organizaciones han desarrollado su propio enfoque para el desarrollo del software. Los procesos han evolucionado para explotar las capacidades de las personas de una organización, así como las características específicas de los sistemas que se están desarrollando. Para algunos sistemas, como los sistemas críticos, se requiere un proceso de desarrollo muy estructurado. Para sistemas de negocio, con requerimientos rápidamente cambiantes, un proceso flexible y ágil probablemente sea más efectivo.

Aunque existen muchos procesos diferentes de software, algunas actividades fundamentales son comunes para todos ellos:

1. ***Especificación del software*** donde los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.
2. ***Desarrollo del software*** donde el software se diseña y programa.
3. ***Validación del software*** donde el software se valida para asegurar que es lo que el cliente requiere.

4. ***Evolución del software*** *donde* el software debe evolucionar para cubrir las necesidades cambiantes del cliente.

Diferentes tipos de sistemas necesitan diferentes procesos de desarrollo. Por lo tanto, estas actividades genéricas pueden organizarse de diferentes formas y describirse en diferentes niveles de detalle para diferentes tipos de software. El uso de un proceso inadecuado del software puede reducir la calidad o la utilidad del producto de software que se va a desarrollar y/o incrementar los costes de desarrollo.

Los procesos del software se pueden mejorar con la estandarización. Esto conduce a mejorar la comunicación y a una reducción del tiempo de formación, y hace la ayuda al proceso automatizado más económica. La estandarización también es un primer paso importante para introducir nuevos métodos, técnicas y buenas prácticas de ingeniería del software.

De todos modos, la existencia de un proceso de software no es garantía de que éste será entregado a tiempo, de que satisfará las necesidades del cliente, o de que mostrará las características técnicas que conducirán a características de calidad a largo plazo. El proceso de software debe **evaluarse** para asegurarse de que cumpla una serie de criterios básicos que han demostrado ser esenciales para una ingeniería de software exitosa.

CMMI (Modelo de Capacidad de Madurez Integrado) es un modelo total del proceso, que describe las metas, prácticas y capacidades específicas con que debe contar un proceso de software maduro. El estándar **SPICE (ISO/IEC15504)** define un conjunto de requisitos para la evaluación del proceso de software. Lo que pretende es ayudar a las organizaciones en el desarrollo de una evaluación objetiva de la eficacia de cualquier proceso de software definido.

El **ISO 9001:2000 para software** es un estándar genérico que se aplica a cualquier organización que desee mejorar la calidad general de los productos, sistemas o servicios que provee. El ISO 9001:2000 subraya la

importancia que tiene para una organización identificar, implementar, gestionar y mejorar de manera continua la efectividad de los procesos necesarios para el sistema de administración de la calidad, y gestionar las interacciones de estos procesos para conseguir los objetivos de la organización.

El ISO 9001:2000 ha adoptado un ciclo de "planear-hacer-revisar-actuar" (PDCA Plan-Do-Check-Act en inglés, también conocido como Círculo de Deming) que se aplica a los elementos de gestión de calidad de un proyecto de software. Dentro de un contexto de software, "planear" establece los objetivos, las actividades y tareas del proceso necesarios para conseguir un software de alta calidad y una satisfacción del cliente; "hacer" implementa el proceso de software (incluidas las actividades del marco de trabajo y las actividades sombrilla); "revisar" monitorea y mide el proceso para asegurarse de que todos los requisitos establecidos para la gestión de calidad hayan sido cumplidos; "actuar" inicia las actividades de mejoramiento del proceso de software, el cual tiene una continuidad de trabajo para mejorar el proceso.

Un **modelo de procesos del software** es una descripción abstracta y simplificada de un proceso del software que presenta una visión de ese proceso. Cada modelo de proceso representa un proceso desde una perspectiva particular y así proporciona sólo información parcial sobre ese proceso. Son abstracciones de los procesos que se pueden utilizar para explicar diferentes enfoques para el desarrollo de software. Puede pensarse en ellos como marcos de trabajo del proceso que pueden ser extendidos y adaptados para crear procesos más específicos de ingeniería del software. Cada modelo describe una sucesión de fases y un encadenamiento entre ellas. Según las fases y el modo en que se produzca este encadenamiento, tenemos diferentes modelos de proceso. Un modelo es más adecuado que otro para desarrollar un proyecto dependiendo de un conjunto de características del proyecto. Los modelos pueden incluir actividades que

son parte de los procesos y productos de software y el papel de las personas involucradas en la ingeniería del software. Alternativamente, a veces se usan los términos **ciclo de vida**, **modelo de ciclo de vida y modelo de desarrollo**.

La mayor parte de los modelos de procesos del software se basan en uno de los tres modelos generales o paradigmas de desarrollo de software:

1. ***El enfoque en cascada***. Considera las actividades anteriores y las representa como fases de procesos separados, tales como la especificación de requerimientos, el diseño del software, la implementación, las pruebas, etcétera. Después de que cada etapa queda definida «se firma» y el desarrollo continúa con la siguiente etapa.
2. ***Desarrollo iterativo***. Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones muy abstractas. Éste se refina basándose en las peticiones del cliente para producir un sistema que satisfaga las necesidades de dicho cliente. El sistema puede entonces ser entregado. De forma alternativa, se puede reimplementar utilizando un enfoque más estructurado para producir un sistema más sólido y mantenible.
3. ***Ingeniería del software basada en componentes*** *(CBSE)*. Esta técnica supone que las partes del sistema ya existen previamente. El proceso de desarrollo del sistema se enfoca en la integración de estas partes más que desarrollarlas desde el principio.

Estos tres modelos de procesos genéricos se utilizan ampliamente en la práctica actual de la ingeniería del software. No se excluyen mutuamente y a menudo se utilizan juntos, especialmente para el desarrollo de sistemas grandes. Los subsistemas dentro de un sistema más grande pueden ser desarrollados utilizando enfoques diferentes. Por lo tanto, aunque es

conveniente estudiar estos modelos separadamente, debe entenderse que, en la práctica, a menudo se combinan.

Pressman separa entre el **modelo personal** (modelo utilizado por cada desarrollador) y **modelo en equipo** (cuando el proyecto es dirigido por varios profesionales) para el proceso de software. Ambos destacan la medición, la planeación y la autodirección como ingredientes clave para un proceso de software exitoso.

31.3 Ciclo de vida del software

Según el estándar ISO-12207 **el ciclo de vida** de un sistema de información es *el marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso*. También se podría definir el ciclo de vida de un sistema de información como *el conjunto de etapas por las que atraviesa el sistema desde su concepción, hasta su retirada de servicio pasando por su desarrollo y explotación*.

No existe un único **modelo de ciclo de vida** que defina los estados por los que pasa cualquier producto software. Dado que existe una gran variedad de aplicaciones y que dicha variedad supone situaciones totalmente distintas, es natural que existan diferentes modelos de ciclo de vida. No obstante, todo modelo de ciclo de vida debe cubrir los siguientes objetivos básicos:

- Definir las actividades a realizar y en qué orden, es decir, determinar el orden de las fases del proceso software.
- Establecer los criterios de transición para pasar de una fase a la siguiente.

- Proporcionar puntos de control para la gestión del proyecto, es decir, calendario y organización.
- Asegurar la consistencia con el resto de los sistemas de información de la organización.

Cada proyecto debe seleccionar el modelo de ciclo de vida que sea más apropiado para su caso, el cual se elige en base a considerar una serie de factores como: la cultura de la organización, el deseo de asumir riesgos, el área de aplicación, la volatilidad de los requisitos, la comprensión de dichos requisitos, etc. En cualquier proyecto software, el modelo de ciclo de vida permite responder a las cuestiones de ¿qué se hará a continuación? y ¿por cuánto tiempo se hará? Dado que cada modelo de ciclo de vida tiene sus ventajas y sus inconvenientes, no se suelen seguir en la práctica los modelos en su forma pura, sino que de acuerdo con las peculiaridades del sistema y la experiencia del personal, se pueden adoptar aspectos de otros modelos que sean más adecuados al caso concreto.

Es importante no confundir el concepto de ciclo de vida con el de metodología. Mientras que el ciclo de vida indica qué actividades hay que realizar y en qué orden, la **metodología** indica cómo avanzar en la construcción del sistema, esto es, con qué técnicas, y entre sus características está la de determinar los recursos a utilizar o las personas implicadas en cada actividad.

31.4 Modelos de ciclo de vida del software

Una posible clasificación sería la que divide los modelos de ciclo de vida en:

- **Modelos tradicionales**: Son los de más amplia utilización. Dentro de este grupo estarían:
  - Modelo en cascada.
  - Modelos basados en prototipos:
    - Modelo de construcción de prototipos.

        - Modelo de desarrollo incremental.
        - Modelo de prototipado evolutivo.
- **Modelos alternativos**
    - Modelo en espiral
    - Modelos basados en transformaciones: La filosofía general es llegar a generar código a partir de unas especificaciones transformándolas por medio de herramientas. Según usemos unas u otras herramientas tendremos:
        - Las que usan técnicas de cuarta generación (Roger Pressman): lenguajes no procedimentales para consultas a BD; generadores de código, de pantallas, de informes; herramientas de manipulación de datos; facilidades gráficas de alto nivel.
        - Basados en modelos de transformación (Carma McClure) => Basados en herramientas CASE que permiten, siguiendo el MCV clásico, pasar de una etapa a otra aplicando las transformaciones que dan las herramientas.
    - Desarrollo de Software Basado en Componentes (DSBC o CBSB).

Aparte de estos modelos de ciclo de vida en la actualidad existen nuevas alternativas:

- Proceso unificado de desarrollo del software. (PUDS)
- Programación Extrema.

31.4.1 Modelo Codificar y Corregir (Code and Fix)

Este es el modelo básico utilizado en los inicios del desarrollo de software. Contiene dos pasos:

- Escribir código.

- Corregir problemas en el código.

Se trata de primero implementar algo de código y luego pensar acerca de requisitos, diseño, validación, y mantenimiento. Este modelo tiene tres problemas principales:

- Después de un número de correcciones, el código puede tener una muy mala estructura, hace que los arreglos sean muy costosos. Esto hizo ver la necesidad de una fase previa de diseño antes de la de codificación.
- Frecuentemente, aún el software bien diseñado, no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara. Esto condujo a la necesidad de introducir una fase de análisis de requerimientos antes del diseño.
- El código es difícil de reparar por su pobre preparación para probar y modificar. Este problema hizo resaltar la necesidad de la planificación y preparación de las distintas tareas en cada fase.

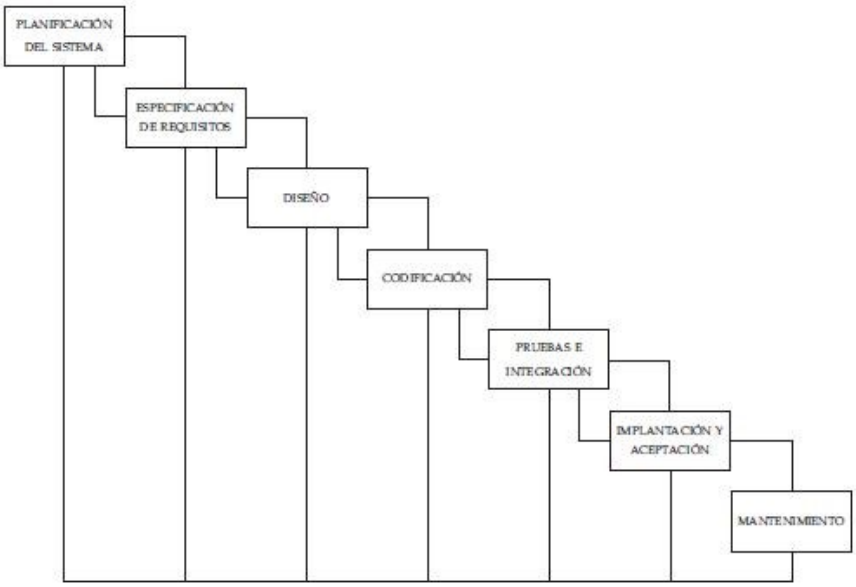
31.4.2 Modelo por Etapas y Modelo en Cascada

Los problemas apuntados del modelo Code and Fix llevaron a la necesidad de realizar el desarrollo del software siguiendo un modelo de etapas sucesivas, el **modelo por etapas** (Stage Wise) que considera las siguientes: Planificación, Especificaciones de operación, Especificaciones de codificación, Codificación, Prueba de cada unidad, Prueba de integración, Eliminación de problemas, y Evaluación del sistema.

El **modelo en cascada** introduce una serie de mejoras respecto al modelo por etapas tales como considerar la realización de bucles de realimentación entre etapas, permitiendo que se puedan resolver los problemas detectados en una etapa, en la etapa anterior y permitir la incorporación inicial del prototipado a fin de captar las especificaciones durante el análisis, o para probar distintas soluciones durante el diseño.

El modelo en cascada se compone de una serie de fases que se suceden secuencialmente, generándose en cada una de ellas unos resultados que serán necesarios para iniciar la fase siguiente. Es decir, la evolución del producto software se produce a través de una secuencia ordenada de transiciones de una fase a la siguiente, según un orden lineal. El número de fases en este modelo es irrelevante, ya que lo que le caracteriza es la *secuencialidad* de las mismas y la *necesidad de completar* cada una de ellas para pasar a la siguiente. El modelo del ciclo de vida en cascada está regido por la documentación, es decir, la decisión del paso de una fase a la siguiente se toma en función de si la documentación asociada a dicha fase está completa o no. Sin embargo, esta forma de proceder no es la más adecuada para algunos tipos de software como el que se usa en las aplicaciones interactivas y de usuario final.

Desde su presentación, el modelo en cascada ha tenido un papel fundamental en el desarrollo de proyectos software. Ha sido, y todavía sigue siendo, el más utilizado, tanto que este modelo se conoce con el nombre de ciclo de vida clásico, si bien incorporando infinidad de variaciones que eliminan el carácter simplista del mismo. Aun así, existen una serie de limitaciones que justifican la necesidad de definir otros modelos.



Como se ha indicado anteriormente, las fases que comprende el ciclo de vida clásico son irrelevantes, tanto en número, como en cuáles sean esas fases siempre que se produzcan secuencialmente. Posiblemente, el modelo clásico más utilizado sea el modelo de siete fases que son:

- **Planificación del sistema:** En esta fase es necesario fijar el ámbito del trabajo a realizar, los recursos necesarios, las tareas a realizar, las referencias a tener en cuenta, el coste estimado del proyecto, la composición del equipo de desarrollo y el orden de las actividades.
- **Especificación de requisitos:** En esta fase es preciso analizar, entender y documentar el problema que el usuario trata de resolver con el sistema y se han de especificar con detalle las funciones, objetivos y restricciones del mismo, a fin de que usuarios y desarrolladores puedan tomar éstas como punto de partida para acometer el resto del sistema. Es decir, en la fase de especificación de requisitos se trata de definir ***qué*** debe hacer el sistema, e identificar la información a procesar, las funciones a realizar, el rendimiento del sistema, las interfaces con otros sistemas y las ligaduras de diseño.
- **Diseño:** Arranca de las especificaciones de la fase anterior. En la fase de diseño, una vez elegida la mejor alternativa, se debe crear la

solución al problema descrito atendiendo a aspectos de interfaces de usuario, estructura del sistema y decisiones sobre la implantación posterior. La fase de diseño trata de definir el ***cómo***.

- **Codificación:** Esta fase consiste en traducir las especificaciones y representaciones del Diseño a un lenguaje de programación capaz de ser interpretado y ejecutado por el ordenador.
- **Pruebas e integración:** Una vez que se tienen los programas en el formato adecuado al ordenador, hay que llevar a cabo las pruebas necesarias que aseguren la corrección de la lógica interna del programa y que éste cubre las funcionalidades previstas. La integración de las distintas partes que componen la aplicación o el sistema debe garantizar el buen funcionamiento del conjunto.
- **Implantación y aceptación del sistema:** El objetivo de esta fase es conseguir la aceptación del sistema por parte de los usuarios del mismo, y llevar a cabo las actividades necesarias para su puesta en producción.
- **Mantenimiento del sistema:** La fase de mantenimiento comienza una vez que el sistema ha sido entregado al usuario y continúa mientras permanece activa su vida útil. Puede deberse a errores no detectados previamente (correctivo), a modificaciones, mejoras o ampliaciones solicitadas por los usuarios (perfectivo, o aumentativo) o a adaptaciones requeridas por la evolución del entorno tecnológico o cambios normativos (mantenimiento adaptativo).

Las principales **críticas** al modelo se centran en sus características básicas, es decir secuencialidad y utilización de los resultados de una fase para acometer la siguiente de manera que el sistema sólo se puede validar cuando está terminado. En cuanto al flujo secuencial, los proyectos reales raramente siguen el flujo secuencias que propone el modelo. Siempre ocurren interacciones y en las últimas fases sobre todo se pueden realizar en paralelo algunas áreas como por ejemplo codificación y pruebas. Una

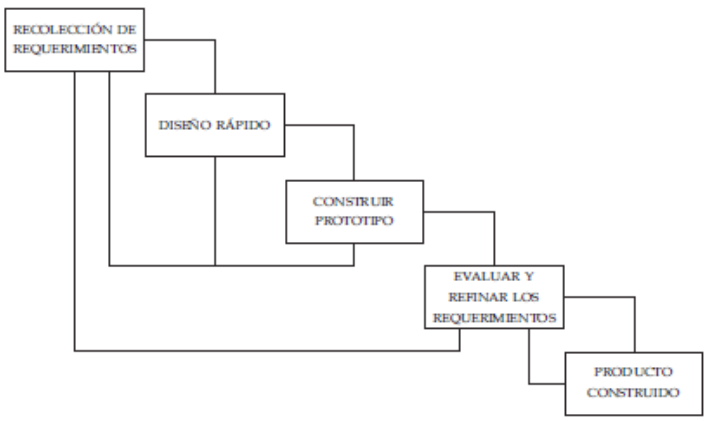
aplicación del modelo en sentido estricto obligaría a la “congelación” de los requisitos de los usuarios, supuesto este completamente alejado de la realidad. El modelo no contempla la posibilidad de realimentación entre fases. Por otro lado, el modelo no prevé revisiones o validaciones intermedias por parte del usuario, así los resultados de los trabajos sólo se ven al final de una serie de tareas y fases de tal forma que si se ha producido un error en las primeras fases este sólo se detectará al final y su corrección tendrá un costo muy elevado, puesto que será preciso rehacer todo el trabajo desde el principio.

31.4.3 Modelos basados en prototipos

Permiten a los desarrolladores construir rápidamente versiones tempranas de los sistemas software que pueden evaluar los usuarios. Existen varios modelos derivados del uso de prototipos

31.4.3.1 Prototipado rápido

Los prototipos deben poder construirse con facilidad para evaluarlos en una temprana fase del desarrollo y además han de ser baratos y desarrollados en poco tiempo. También se denominan de usar y tirar.



El prototipo sirve para crear y validar la especificación, y para que el usuario tenga una idea de cómo será el software antes de que comience el desarrollo. Es importante precisar que el prototipo se construye sólo para servir como mecanismo de definición de los requerimientos funcionales. Posteriormente ha de desecharse y debe construirse el sistema con los criterios normales de calidad y mantenimiento, siguiendo, por ejemplo, el ciclo de vida clásico, ya que generalmente el prototipo se ha construido tomando decisiones de implementación contrarias al buen criterio de desarrollo de software. Los objetivos del prototipo son:

- Reducir el riesgo de construir un producto que se aleje de las necesidades del usuario
- Reducir el coste de desarrollo al disminuir las correcciones en etapas avanzadas del mismo.
- Aumentar las posibilidades de éxito del producto.

El principal problema de este modelo es que el usuario ve en el prototipo lo que parece ser una versión de trabajo del software, sin saber que con la prisa de hacer que funcione no se ha tenido en cuenta la calidad del software global o la facilidad de mantenimiento a largo plazo. Cuando se informa de que el producto se debe construir otra vez para que se puedan mantener los niveles altos de calidad, el cliente no lo entiende y pide que se apliquen unos pequeños ajustes que puedan hacer del prototipo un producto final.

31.4.3.2 Prototipado evolutivo

En este tipo de ciclo de vida se construye una implementación parcial del sistema que satisface los requisitos conocidos, la cual es utilizada por el usuario para llegar a comprender mejor la totalidad de requisitos que desea.

Desde un punto de vista genérico, se puede decir que los modelos evolutivos se encaminan a conseguir un sistema flexible que se pueda expandir, de forma que se pueda realizar rápidamente un nuevo sistema cuando cambian los requisitos. Estos modelos consisten en implementar un producto software operativo y hacerle evolucionar de acuerdo con la propia experiencia operacional. Están especialmente indicados en situaciones en que se utilizan lenguajes de cuarta generación (L4G) y para aquellas otras en que el usuario no puede decir lo que requiere, pero lo reconocerá cuando lo vea. Los modelos evolutivos dan al usuario una rápida capacidad de operación inicial y una buena base para determinar mejoras del sistema. Está relacionado con el concepto de RAD (Rapid Application Development – Desarrollo Rápido de Aplicaciones), que identifica los asistentes, plantillas y entornos de fácil y rápida creación de software.

La *diferencia* fundamental entre el prototipado rápido y el evolutivo estriba en que mientras que en el primer caso se asume que existen una serie de requisitos reales, aunque para establecer lo que el usuario quiere realmente es necesario establecer una serie de iteraciones antes de que los requisitos se estabilicen al final, en el caso evolutivo se asume desde el principio que los requisitos cambian continuamente.

En el prototipo rápido lo lógico es implementar sólo aquellos aspectos del sistema que se entienden mal, mientras que en el prototipo evolutivo lo lógico es comenzar por los aspectos que mejor se comprenden y seguir construyendo apoyados en los puntos fuertes y no en los débiles. Como resultado de este modo de desarrollo, la solución software evoluciona acercándose cada vez más a las necesidades del usuario; ahora bien, pasado un tiempo el sistema software así construido deberá ser rehecho o sufrir una profunda reestructuración con el fin de seguir evolucionando.

El modelo de prototipado evolutivo (Evolutionary Development model) también tiene sus **dificultades**. Se le puede considerar como una nueva versión, utilizando lenguajes de programación de más alto nivel, del viejo modelo CODE-AND-FIX. Otro inconveniente que presenta es partir de la suposición, muchas veces no realista, de que el sistema operacional del usuario final será lo suficientemente flexible como para poder incorporar caminos de evolución futuros no planificados con anterioridad.

31.4.3.3 Modelo de Desarrollo Incremental

El modelo de desarrollo incremental consiste en desarrollar un sistema que satisfaga una parte de los requisitos especificados y posteriormente ir creando versiones que incorporen los requisitos que faltan, hasta llegar al sistema final. Actuando así, se pretende disponer pronto de un sistema que aunque sea incompleto, sea utilizable y satisfaga parte de los requisitos, evitando de paso el efecto big-bang, es decir, que durante un período largo de tiempo no se tenga nada y de repente haya una situación completamente nueva. Por otra parte, también se logra que el usuario se implique estrechamente en la planificación de los pasos siguientes.

El modelo de desarrollo incremental también se utiliza para evitar la demanda de funcionalidades excesivas al sistema por parte de los usuarios, ya que como a éstos les resulta difícil definir sus necesidades reales

tienden a pedir demasiado. Actuando con este modelo se atiende primero a las funcionalidades esenciales y las funcionalidades accesorias sólo se incluyen en las versiones sucesivas cuando realmente son necesarias.

La diferencia entre el modelo de desarrollo evolutivo y el modelo de desarrollo incremental radica en dos aspectos:

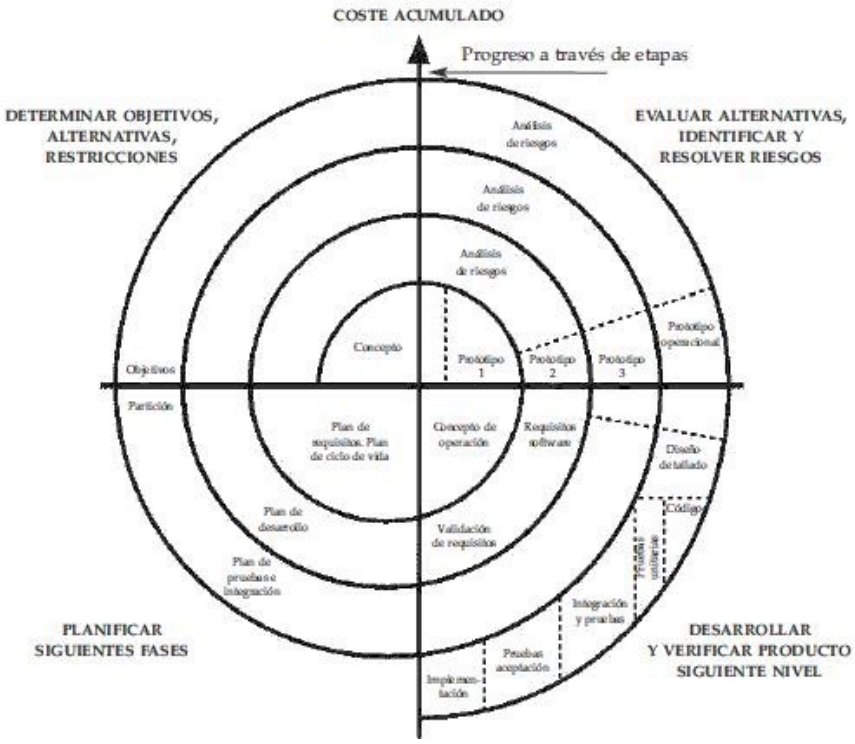
- El modelo incremental parte de la hipótesis de que se conocen todos los requisitos y éstos se van incorporando al sistema en versiones sucesivas. En el modelo evolutivo sólo se conocen unos pocos requisitos y los restantes se van descubriendo en sucesivas evoluciones del prototipo.
- Cada vez que se desarrolla una nueva versión, en el modelo evolutivo es una versión de todo el sistema, mientras que en el incremental es una versión anterior sin cambios, más un número de nuevas funciones.

31.4.4 Modelo en Espiral

El *modelo en espiral,* propuesto originalmente por Boehm, es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo en cascada. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado. Las principales diferencias entre el modelo en espiral y los modelos de ciclo de vida más tradicionales son:

- En el modelo en espiral hay un reconocimiento explícito de las diferentes **alternativas** para alcanzar los objetivos del proyecto.

- El modelo en espiral se centra en la identificación de los **riesgos** asociados a cada alternativa y en la manera de resolver dichos riesgos.
- En el modelo en espiral los proyectos se dividen en ciclos (ciclos de espiral), avanzándose en el desarrollo mediante consensos al final de cada ciclo.
- El modelo en espiral se adapta a cualquier tipo de actividad.



El modelo en espiral refleja la idea de que cada ciclo implica una progresión en el desarrollo del producto software que aplica la misma secuencia de pasos para cada parte del producto y para cada uno de sus niveles de elaboración, desde la concepción global hasta la codificación individual de cada programa. Esta secuencia de pasos, iterativa en cada fase del desarrollo, se compone de las cuatro actividades siguientes:

- **Planificación**: Este primer paso con el que comienza cada ciclo de espiral consiste en la identificación de los objetivos de la parte del producto que está siendo elaborada (funcionalidad, rendimiento,

adaptación a los cambios, etc.), identificación de las alternativas principales para realizar o implementar esta parte del producto, y la identificación de las restricciones impuestas (coste, plazo de realización, interfaces, etc.).

- **Análisis de riesgos**: Comienza con la evaluación de cada alternativa respecto a los objetivos y a las restricciones. Este proceso de evaluación identificará áreas de incertidumbre que son fuentes significativas de riesgo en el proyecto. Se decidirá como resolver los riesgos asociados a la alternativa elegida.
- **Ingeniería**. Este paso consiste en el desarrollo y verificación del producto objeto de la fase (ciclo de espiral) en que nos encontremos. Como esta implementación está dirigida por el riesgo, el desarrollo podrá seguir las pautas de un prototipado evolutivo, las del ciclo de vida clásico, las orientadas a transformaciones automáticas, o cualquier otro enfoque del desarrollo. En definitiva, esto permite al modelo en espiral acomodarse a cualquier mezcla de estrategias de desarrollo.
- **Evaluación del cliente**. Una característica importante del modelo en espiral es que cada ciclo de espiral se completa con una revisión en la que participan aquellos que tienen relación con el producto (desarrolladores, usuarios, etc.). Esta revisión incluye todos los productos desarrollados durante el ciclo, los planes para el siguiente ciclo y los recursos necesarios para llevarlos a cabo.

Según esto, el modelo se puede representar mediante unos ciclos externos de espiral, que representan las fases en que se ha dividido el desarrollo del proyecto software, normalmente las del modelo clásico, y unos ciclos internos, iterativos para cada fase, en los que se llevan a cabo las cuatro actividades antes citadas. La dimensión radial indica los costes de desarrollo acumulativos, mientras que la dimensión angular indica el progreso hecho en cumplimentar cada fase.

La principal ventaja del modelo en espiral es el amplio rango de opciones a que puede ajustarse y que éstas permiten utilizar los modelos de proceso de construcción de software tradicionales; por otra parte, su orientación al riesgo evita, si no elimina, muchas de las posibles dificultades. Otras **ventajas** son:

- Concentra su atención en opciones que permiten la reutilización de software ya existente.
- Se centra en la eliminación de errores y alternativas poco atractivas.
- No establece procedimientos diferentes para el desarrollo del software y el mantenimiento del mismo.
- Proporciona un marco estable para desarrollos integrados hardware-software.
- Permite preparar la evolución del ciclo de vida del producto software, así como el crecimiento y cambios de éste.
- Permite incorporar objetivos de calidad en el desarrollo de productos software.
- Se adapta muy bien al diseño y programación orientada a objetos. Posiblemente con este método es cuando obtiene mejores resultados.

En cuanto a los **inconvenientes** que plantea la utilización del modelo en espiral, cabe citar:

- Dificultad para adaptar su aplicabilidad al software contratado, debido a la poca flexibilidad y libertad de éste.
- Dependencia excesiva de la experiencia que se tenga en la identificación y evaluación de riesgos.
- Necesidad de una elaboración adicional de los pasos del modelo, lo que depende también, en gran medida, de la experiencia del personal.

31.4.5 Modelos basados en Transformaciones

Estos modelos, también llamados *ciclo de vida con producción automática de diseño y código*, se han propuesto como solución al problema que plantean los modelos de desarrollo evolutivo de producir software mal estructurado. Se basan en la posibilidad de convertir automáticamente una especificación formal de un producto software en un programa que satisfaga las especificaciones, utilizando herramientas de cuarta generación. Para ello, los pasos típicos que siguen estos modelos son:

1. Especificación formal del producto tal como lo permita la comprensión inicial del problema.
2. Transformación automática de la especificación en código.
3. Realizar bucles iterativos para mejorar el rendimiento del código resultante.
4. Probar el producto resultante.
5. Reajustar las especificaciones para dejarlas en concordancia con el resultado de la experiencia operativa y volver a generar el código a partir de las especificaciones, volviendo a optimizar y probar el producto.

El modelo de transformación, por tanto, evita la dificultad de tener que modificar el código poco estructurado (por haber pasado por sucesivas reoptimizaciones), puesto que las modificaciones las aplica sobre la especificación de partida. Esto, también evita el tiempo adicional que se emplearía en los pasos intermedios de diseño, codificación y pruebas.

La dificultad que presentan estos modelos es que las posibilidades de transformación automática generalmente sólo están disponibles para productos relativamente pequeños y aplicados a unas áreas muy limitadas. También comparte algunas de las dificultades del modelo de desarrollo evolutivo tales como, por ejemplo, la suposición de que el sistema operacional del usuario final se prestará a evoluciones no planificadas con anterioridad.

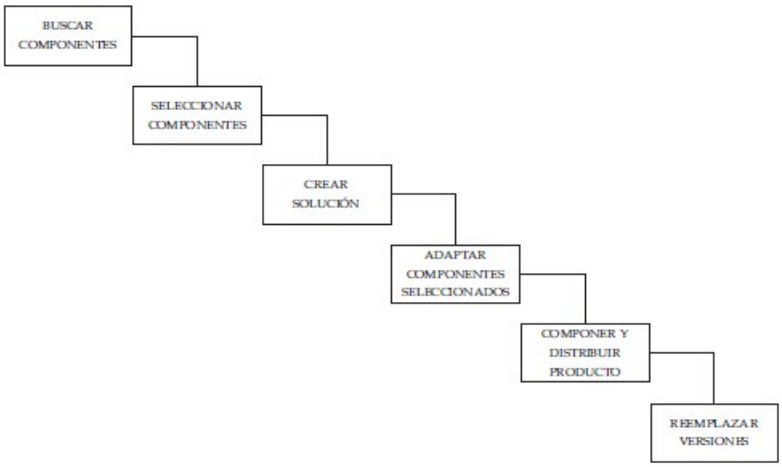
Dentro de este tipo de modelos se encuentran:

- Los que usan técnicas de cuarta generación (Roger Pressman): Suelen estar basados en herramientas de cuarta generación. Estos permiten la generación de código rápido. En ellos se indica qué se quiere obtener, no cómo.
- Basados en modelos de transformación (Carma McClure) => Basados en herramientas CASE que permiten, siguiendo el MCV clásico, pasar de una etapa a otra aplicando las transformaciones que dan las herramientas.

En ambos casos, la filosofía general es llegar a generar código a partir de unas especificaciones transformándolas por medio de herramientas.

31.4.6 Desarrollo basado en componentes

La complejidad de los sistemas computacionales actuales nos ha llevado a buscar la reutilización del software existente. El desarrollo de software basado en componentes permite reutilizar piezas de código preelaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión. En esencia, un componente es una pieza de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar. Los componentes son los "ingredientes de las aplicaciones", que se juntan y combinan para llevar a cabo una tarea. El paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen se conoce como Desarrollo de Software Basado en Componentes.



Los pasos de que consta el ciclo de desarrollo para un sistema basado en componentes son:

1. Buscar componentes, tanto COTS (Comercial Off-The-Shelf) como no COTS.
2. Seleccionar los componentes más adecuados para el sistema.
3. Crear una solución compuesta que integre la solución previa.
4. Adaptar los componentes seleccionados de forma que se ajusten al modelo de componentes o a los requisitos de la aplicación.
5. Componer y distribuir el producto.
6. Reemplazar versiones anteriores o mantener las partes COTS y no COTS del sistema.

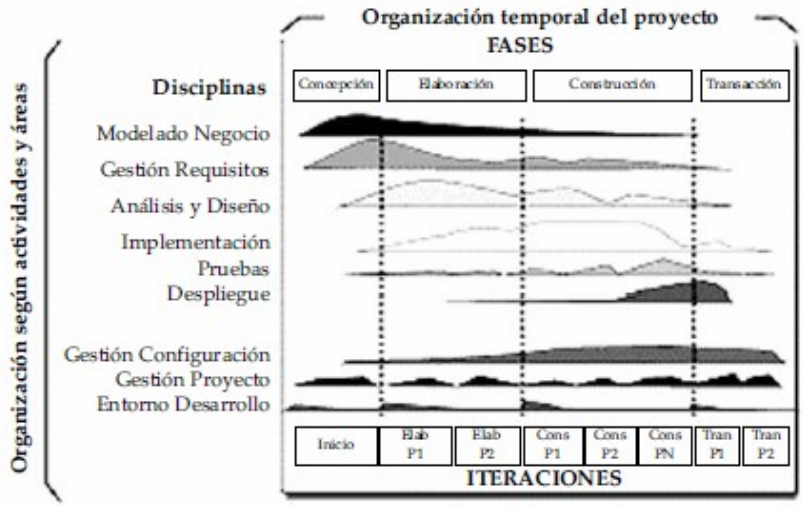
Además de los problemas inherentes a la reutilización del software, los productos COTS presentan problemas específicos como incompatibilidad, inflexibilidad (no existe código fuente), complejidad (esfuerzo de aprendizaje) o cambio de versiones, por lo que el establecimiento de métodos sistemáticos y repetibles para evaluar y seleccionar dichos componentes es un aspecto importante para el desarrollo del software basado en componentes y, en general, para la Ingeniería del Software Basada en Componentes (ISBC).

Entre las ventajas del Desarrollo basado en componentes tenemos que se reducen tiempos y costes de desarrollo y se aumenta la fiabilidad. Entre

los inconvenientes, tendremos la dificultad para reconocer los componentes potencialmente reutilizables, dificultad de catalogación y recuperación y los problemas de gestión de configuración.

31.4.7 Proceso Unificado de Desarrollo de software (PUDS)

En realidad es una metodología que propone un modelo de ciclo de vida. Está desarrollada por tres padres de la IS moderna: Yourdon, Booch y Rumbaugh. Plantea un modelo de ciclo de vida iterativo e incremental, centrado en una arquitectura que guía el desarrollo del sistema, cuyas actividades están dirigidas por casos de uso y soporta las técnicas orientadas a objetos. PUDS impulsa un control de calidad y una gestión de riesgos objetivos y continuos.



El PUDS se compone de fases, iteraciones y ciclos. Una fase es el intervalo de tiempo entre dos hitos importantes del proceso durante la cual se cumple un conjunto bien definido de objetivos, se completan entregables y se toman las decisiones sobre si pasar o no a la siguiente fase. Las **fases** son:

1. **Iniciación**. En esta fase se establece la visión del negocio, que incluye el contexto del negocio, los factores de éxito, y la previsión económica. Para completar la visión del negocio se genera un plan del proyecto, una descripción de los posibles riesgos y del propio proyecto (requisitos principales del proyecto, restricciones y características claves)
2. **Elaboración**. Es donde el proyecto comienza a tomar forma. En esta fase se hace el análisis del dominio del problema y se obtiene una idea básica de la arquitectura del sistema, además de revisarse los riesgos. En esta fase el proyecto todavía puede cancelarse o rediseñarse.
3. **Construcción**. En esta fase el enfoque se traslada al desarrollo de componentes y otras características del sistema que está siendo diseñado. Aquí se realiza el grueso de las tareas de codificación. En proyectos grandes, se puede dividir la fase en varias iteraciones para dividir los casos de uso en segmentos manejables que produzcan prototipos funcionales.
4. **Transición**. El producto se implanta en la organización del usuario final. Aquí se lleva a cabo la formación de los usuarios finales y las pruebas de aceptación del sistema para validarlo contra las expectativas del usuario.

En cada fase hay una o varias iteraciones. Una **iteración** ofrece como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo. Cada fase e iteración se centra en disminuir algún riesgo y concluye con un hito bien definido. El paso a través de las 4 fases constituye un **ciclo** de desarrollo y produce una generación de software. El primer ciclo es el inicial y después serán ciclos de evolución del sistema.

Los flujos de trabajo del proceso son los siguientes:

- Modelado del negocio. El objetivo es establecer una mejor comprensión y un mejor canal de comunicación entre los clientes y los expertos en sistemas.
- Requisitos. El objetivo es describir lo que el sistema debe hacer.
- Análisis y diseño. Aquí se muestra la forma que tendrá el sistema en la fase de implementación.
- Implementación. Codificar y realizar pruebas unitarias.
- Pruebas. Se realizan pruebas de integración.
- Despliegue. Incluye una amplia variedad de actividades como la generación de versiones estables o la distribución e instalación del software.
- Configuración y gestión de cambios.
- Gestión del proyecto. Se realiza a 2 niveles, un nivel de grano grueso que trata la planificación de las fases y otro nivel de grano fino que trata la planificación de las iteraciones.
- Entorno.

31.4.8 Modelo de métodos formales

El modelo de métodos formales comprende un conjunto de actividades que conducen a la especificación matemática del software de computadora. Los métodos formales permiten que un ingeniero de software especifique, desarrolle y verifique un sistema basado en computadora aplicando una notación rigurosa y matemática. Algunas organizaciones de desarrollo del software actualmente aplican una variación de este enfoque, llamado **ingeniería del software de sala limpia**.

Cuando se utilizan métodos formales la ambigüedad, lo incompleto y la inconsistencia se descubren y se corrigen más fácilmente, no mediante una revisión a propósito para el caso, sino mediante la aplicación del análisis

matemático. Cuando se utilizan métodos formales durante el diseño, sirven como base para la verificación de programas y por consiguiente permiten que el ingeniero del software descubra y corrija errores que no se pudieron detectar de otra manera. Los modelos de métodos formales ofrecen la promesa de un software libre de defectos. Sin embargo, se ha hablado de una gran preocupación sobre su aplicabilidad en un entorno de gestión:

1. El desarrollo de modelos formales actualmente es bastante caro y lleva mucho tiempo.
2. Se requiere un estudio detallado porque pocos responsables del desarrollo de software tienen los antecedentes necesarios para aplicar métodos formales.
3. Es difícil utilizar los modelos como un mecanismo de comunicación con clientes que no tienen muchos conocimientos técnicos.

No obstante es posible que el enfoque a través de métodos formales tenga más partidarios entre los desarrolladores que deben construir software de mucha seguridad y robustez.

31.4.9 Programación Extrema (eXtreme Programming)

En la programación extrema, todos los requerimientos se expresan como escenarios (llamados historias de usuario), los cuales se implementan directamente como una serie de tareas. Los programadores trabajan en parejas y desarrollan pruebas para cada tarea antes de escribir el código. Todas las pruebas se deben ejecutar satisfactoriamente cuando el código nuevo se integre al sistema. Existe un pequeño espacio de tiempo entre las entregas del sistema. La programación extrema implica varias prácticas, que se ajustan a los principios de los métodos ágiles:

1. El desarrollo incremental se lleva a cabo través de entregas del sistema pequeñas y frecuentes y por medio de un enfoque para la

descripción de requerimientos basado en las historias de cliente o escenarios que pueden ser la base para el proceso de planificación.

2. La participación del cliente se lleva a cabo a través del compromiso a tiempo completo del cliente en el equipo de desarrollo. Los representantes de los clientes participan en el desarrollo y son los responsables de definir las pruebas de aceptación del sistema.
3. El interés en las personas, en vez de en los procesos, se lleva a cabo a través de la programación en parejas, la propiedad colectiva del código del sistema, y un proceso de desarrollo sostenible que no implique excesivas jornadas de trabajo.
4. El cambio se lleva a cabo a través de las entregas regulares del sistema, un desarrollo previamente probado y la integración continua.
5. El mantenimiento de la simplicidad se lleva a cabo a través de la refactorización constante para mejorar la calidad del código y la utilización de diseños sencillos que no prevén cambios futuros en el sistema.

Los clientes del sistema son parte del equipo de desarrollo y discuten escenarios con otros miembros del equipo. Desarrollan conjuntamente una «tarjeta de historias» (story card) que recoge las necesidades del cliente. El equipo de desarrollo intentará entonces implementar ese escenario en una entrega futura del software. Una vez que se han desarrollado las tarjetas de historias, el equipo de desarrollo las divide en tareas y estima el esfuerzo y recursos requeridos para su implementación. El cliente establece entonces la prioridad de las historias a implementar.

El problema con la implementación de cambios imprevistos es que tienden a degradar la estructura del software, por lo que los cambios se hacen cada vez más difíciles de implementar. La programación extrema aborda este problema sugiriendo que se debe refactorizar constantemente el software. Esto significa que el equipo de programación busca posibles mejoras del

software y las implementa inmediatamente. Por lo tanto, el software siempre debe ser fácil de entender y cambiar cuando se implementen nuevas historias.

Otra práctica innovadora que se ha introducido es que los programadores trabajan en parejas para desarrollar el software. Las ventajas de esto son que apoya la idea de la propiedad y responsabilidad comunes del sistema, actúa como un proceso de revisión informal del código y ayuda en la refactorización

31.6 Metodologías de desarrollo de sistemas de información.

Un proceso de software detallado y completo suele denominarse “Metodología”. Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, etc.).

Adicionalmente una metodología debería definir con precisión los productos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc. Habitualmente se utiliza el término “método” para referirse a técnicas, notaciones y guías asociadas, que son aplicables a una (o algunas) actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis y/o diseño.

La comparación y/o clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, información disponible y alcance de cada una de ellas. A grandes rasgos, si tomamos como criterio las notaciones utilizadas para especificar productos producidos en actividades de análisis y diseño, podemos clasificar las metodologías en dos grupos: Metodologías Estructuradas y Metodologías Orientadas a Objetos. Por otra parte, considerando su filosofía de desarrollo, aquellas metodologías con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales (o peyorativamente

denominada Metodologías Pesadas, o Peso Pesado). Otras metodologías, denominadas Metodologías Ágiles, están más orientadas a la generación de código con ciclos muy cortos de desarrollo, se dirigen a equipos de desarrollo pequeños, hacen especial hincapié en aspectos humanos asociados al trabajo en equipo e involucran activamente al cliente en el proceso. A continuación se revisan brevemente algunas de estas categorías de metodologías.

- **Metodologías estructuradas**: Los métodos estructurados comenzaron a desarrollarse a fines de los 70's con la Programación Estructurada. A mediados de los 70's aparecieron técnicas para el Diseño (por ejemplo: el diagrama de Estructura) primero y posteriormente para el Análisis (por ejemplo: Diagramas de Flujo de Datos). Estas metodologías son particularmente apropiadas en proyectos que utilizan para la implementación lenguajes de 3ra y 4ta generación. Ejemplos de metodologías estructuradas de ámbito gubernamental: MERISE (Francia), MÉTRICA (España), SSADM (Reino Unido). Ejemplos de propuestas de métodos estructurados en el ámbito académico: Gane & Sarson, Ward & Mellor, Yourdon & DeMarco e Information Engineering.
- **Metodologías orientadas a objetos**: Su historia va unida a la evolución de los lenguajes de programación orientada a objetos. A fines de los 80's comenzaron a consolidarse algunos métodos Orientados a Objetos. En 1995 Booch y Rumbaugh proponen el Método Unificado con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta a un objetivo más modesto, para dar lugar al Unified Modeling Language (UML), la notación OO más popular en la actualidad. Algunos métodos OO con notaciones predecesoras de UML son: OOAD (Booch), OOSE (Jacobson), Coad & Yourdon, Shaler & Mellor y OMT (Rumbaugh). Algunas metodologías orientadas a objetos que utilizan la notación

UML son: Rational Unified Process (RUP), OPEN, MÉTRICA (que también soporta la notación estructurada).

- **Metodologías tradicionales (no ágiles):** Las metodologías no ágiles son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema. Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales..
- **Metodologías ágiles**: Un proceso es ágil cuando el desarrollo de software es **incremental** (entregas pequeñas de software, con ciclos rápidos), **cooperativo** (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), **sencillo** (el método en sí mismo es fácil de aprender y modificar, bien documentado), y **adaptable** (permite realizar cambios de último momento). Algunas de las metodologías ágiles identificadas son Extreme Programming, Scrum, Familia de Metodologías Crystal, Feature Driven Development, Proceso Unificado Rational, Dynamic Systems Development Method, Adaptive Software Development. Se verán con más detalle en un apartado posterior.
- **Proceso Unificado de Rational**: Es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Se verá en un apartado posterior.
- **Métrica V3**: La metodología MÉTRICA Versión 3 ofrece a las Organizaciones un instrumento útil para la sistematización de las actividades que dan soporte al ciclo de vida del software. Se verá en un apartado posterior.

- **Open Source Development Software**: Open Source es software desarrollado con la falta de coordinación, donde los programadores colaboran libremente, utilizando el código fuente distribuido y la infraestructura de comunicaciones de Internet. El código abierto se basa en la filosofía del software libre, sin embargo, extiende esta ideología ligeramente para presentar un enfoque más comercial que incluye tanto un modelo de negocio como una metodología de desarrollo.

31.7 Métrica Versión 3.

La metodología MÉTRICA Versión 3 ofrece a las Organizaciones un instrumento útil para la sistematización de las actividades que dan soporte al ciclo de vida del software dentro del marco que permite alcanzar los siguientes objetivos:

- Proporcionar o definir Sistemas de Información que ayuden a conseguir los fines de la Organización mediante la definición de un marco estratégico para el desarrollo de los mismos.
- Dotar a la organización de productos software que satisfagan las necesidades de los usuarios dando una mayor importancia al análisis de requisitos.
- Mejorar la productividad de los departamentos de Sistemas y Tecnologías de la Información y las Comunicaciones, permitiendo una mayor capacidad de adaptación a los cambios y teniendo en cuenta la reutilización en la medida de lo posible.
- Facilitar la comunicación y entendimiento entre los distintos participantes en la producción de software a lo largo del ciclo de vida del proyecto, teniendo en cuenta su papel y responsabilidad, así como las necesidades de todos y cada uno de ellos.
- Facilitar la operación, mantenimiento y uso de los productos software obtenidos.

En la elaboración de MÉTRICA Versión 3 se han tenido en cuenta los métodos de desarrollo más extendidos, así como los últimos estándares de ingeniería del software y calidad, además de referencias específicas en cuanto a seguridad y gestión de proyectos. En una única estructura, la metodología MÉTRICA Versión 3 cubre distintos tipos de desarrollo: estructurado y orientado a objetos, facilitando a través de interfaces la realización de los procesos de apoyo u organizativos: Gestión de Proyectos, Gestión de Configuración, Aseguramiento de Calidad y Seguridad.

En lo que se refiere a estándares se ha tenido en cuenta como referencia el Modelo de Ciclo de Vida de Desarrollo propuesto en la norma ISO 12.207 "*Information technology - Software life cycle processes*". Siguiendo este modelo se ha elaborado la estructura de MÉTRICA Versión 3 en la que se distinguen procesos principales (Planificación, Desarrollo y Mantenimiento) e interfaces (Gestión de Proyectos, Aseguramiento de la Calidad, Seguridad y Gestión de Proyectos) cuyo objetivo es dar soporte al proyecto en los aspectos organizativos. Además de la norma ISO 12.207, entre los estándares de referencia hay que destacar las normas ISO/IEC TR 15.504/SPICE "*Software Process Improvement and Assurance Standards Capability Determination*", *UNE-EN-ISO 9001:2000 Sistemas de Gestión de la Calidad. Requisitos, UNE-EN-ISO 9000:2000 Sistemas de Gestión de la Calidad. Fundamentos y Vocabulario* y el estándar *IEEE 610.12-1.990* "*Standard Glossary of Software Engineering Terminology*". Igualmente se han tenido en cuenta otras metodologías como SSADM, Merise, Information Engineering, *MAGERIT. Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información* promovida por el Consejo Superior de Informática y *EUROMÉTODO.*

Se ha diferenciado entre la aplicación de Técnicas, como conjunto de heurísticas y procedimientos apoyados en estándares que utilizan notaciones específicas en términos de sintaxis y semántica, y de Prácticas cuya utilización no conlleva reglas preestablecidas con la misma rigidez.

Las nuevas técnicas están ampliamente soportadas por herramientas comerciales.

31.7.1 Procesos Principales de Métrica Versión 3

MÉTRICA Versión 3 tiene un enfoque orientado al proceso y por ello, como ya se ha dicho, se ha enmarcado dentro de la norma ISO 12.207, que se centra en la clasificación y definición de los procesos del ciclo de vida del software. MÉTRICA Versión 3 cubre el Proceso de Desarrollo y el Proceso de Mantenimiento de Sistemas de Información.

MÉTRICA Versión 3 ha sido concebida para abarcar el desarrollo completo de Sistemas de Información sea cual sea su complejidad y magnitud, por lo cual su estructura responde a desarrollos máximos y deberá adaptarse y dimensionarse en cada momento de acuerdo a las características particulares de cada proyecto. La metodología descompone cada uno de los procesos en actividades, y éstas a su vez en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes. El orden asignado a las actividades no debe interpretarse como secuencia en su realización, ya que éstas pueden realizare en orden diferente a su numeración o bien en paralelo. Sin embargo, no se dará por acabado un proceso hasta no haber finalizado todas las actividades del mismo determinadas al inicio del proyecto.

Así los procesos de la estructura principal de MÉTRICA Versión 3 son los siguientes:

- PLANIFICACIÓN DE SISTEMAS DE INFORMACIÓN.
- DESARROLLO DE SISTEMAS DE INFORMACIÓN.
- MANTENIMIENTO DE SISTEMAS DE INFORMACIÓN.

31.7.2 Planificación de sistemas de información

El objetivo de un Plan de Sistemas de Información es proporcionar un marco estratégico de referencia para los Sistemas de Información de un determinado ámbito de la Organización. El resultado del Plan de Sistemas debe, por tanto, orientar las actuaciones en materia de desarrollo de Sistemas de Información con el objetivo básico de apoyar la estrategia corporativa, elaborando una arquitectura de información y un plan de proyectos informáticos para dar apoyo a los objetivos estratégicos. Por este motivo es necesario un proceso como el de Planificación de Sistemas de Información, en el que participen, por un lado los responsables de los procesos de la organización con una visión estratégica y por otro, los profesionales de SI capaces de enriquecer dicha visión con la aportación de ventajas competitivas por medio de los sistemas y tecnologías de la información y comunicaciones.

Como productos finales de este proceso se obtienen los siguientes:

- Catálogo de requisitos de PSI que surge del estudio de la situación actual en el caso de que sea significativo dicho estudio, del diagnóstico que se haya llevado a cabo y de las necesidades de información de los procesos de la organización afectados por el plan de sistemas.
- Arquitectura de información que se compone de los siguientes productos: modelo de información, modelo de sistemas de información, arquitectura tecnológica, plan de proyectos y plan de mantenimiento del PSI.

Este nuevo enfoque de alineamiento de los sistemas de información con la estrategia de la organización, la implicación directa de la alta dirección y la propuesta de solución presenta como ventajas:

- La implicación de la alta dirección facilita que se pueda desarrollar con los recursos necesarios y el calendario establecido.

- La perspectiva horizontal de los procesos dentro de la Organización facilita que se atienda a intereses globales y no particulares de unidades organizativas que puedan desvirtuar los objetivos del Plan.
- La prioridad del desarrollo de los sistemas de información de la organización por objetivos estratégicos.
- La propuesta de Arquitectura de Información que se hace en el plan es más estratégica que tecnológica.

31.7.3 Desarrollo de Sistemas de Información

El proceso de Desarrollo de MÉTRICA Versión 3 contiene todas las actividades y tareas que se deben llevar a cabo para desarrollar un sistema, cubriendo desde el análisis de requisitos hasta la instalación del software. Además de las tareas relativas al análisis, incluye dos partes en el diseño de sistemas: arquitectónico y detallado. También cubre las pruebas unitarias y de integración del sistema. Este proceso es, sin duda, el más importante de los identificados en el ciclo de vida de un sistema y se relaciona con todos los demás.

En MÉTRICA Versión 3 se han abordado los dos tipos de desarrollo: estructurado y orientado a objeto, por lo que ha sido necesario establecer actividades a realizar en función del tipo de desarrollo elegido. Se han definido 5 subprocesos para este apartado:

- Estudio de Viabilidad del Sistema (EVS)
- Análisis Del Sistema De Información (ASI).
- Diseño Del Sistema De Información (DSI).
- Construcción Del Sistema De Información (CSI).
- Implantación Y Aceptación Del Sistema (IAS).

31.7.3.1 Estudio de Viabilidad del Sistema (EVS)

El propósito de este proceso es analizar un conjunto concreto de necesidades, con la idea de proponer una solución a corto plazo. Los criterios con los que se hace esta propuesta no serán estratégicos sino tácticos y relacionados con aspectos económicos, técnicos, legales y operativos. Los resultados del Estudio de Viabilidad del Sistema constituirán la base para tomar la decisión de seguir adelante o abandonar. Si se decide seguir adelante pueden surgir uno o varios proyectos que afecten a uno o varios sistemas de información. Se considerarán alternativas de solución basadas en soluciones "a medida", soluciones basadas en la adquisición de productos software del mercado o soluciones mixtas. Para valorar las alternativas planteadas y determinar una única solución, se estudiará el impacto en la organización de cada una de ellas, la inversión y los riesgos asociados. El resultado final de este proceso son los productos relacionados con la solución que se propone para cubrir la necesidad concreta que se planteó en el proceso, y que depende de si la solución conlleva desarrollo a medida o no.

31.7.3.2 Análisis del Sistema de Información (ASI)

El propósito de este proceso es conseguir la especificación detallada del sistema de información, a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso de Diseño del Sistema de Información.

En primer lugar se describe inicialmente el sistema de información, a partir de los productos generados en el proceso Estudio de Viabilidad del Sistema (EVS). Se delimita su alcance, se genera un catálogo de requisitos

generales y se describe el sistema mediante unos modelos iniciales de alto nivel. Se recogen de forma detallada los requisitos funcionales que el sistema de información debe cubrir, catalogándolos, lo que permite hacer la traza a lo largo de los procesos de desarrollo. Además, se identifican los requisitos no funcionales del sistema, es decir, las facilidades que ha de proporcionar el sistema, y las restricciones a que estará sometido, en cuanto a rendimiento, frecuencia de tratamiento, seguridad, etc. Para facilitar el análisis del sistema se identifican los subsistemas de análisis, y se elaboran los modelos de Casos de Uso y de Clases, en desarrollos orientados a objetos, y de Datos y Procesos en desarrollos estructurados. Se especificarán todas las interfaces entre el sistema y el usuario, como formatos de pantallas, diálogos, formatos de informes y formularios de entrada. Finalizados los modelos, se realiza un análisis de consistencia. Una vez realizado dicho análisis de consistencia se elabora el producto ***Especificación de Requisitos Software***, que constituye un punto de referencia en el desarrollo del software y la línea base de referencia para las peticiones de cambio sobre los requisitos inicialmente especificados. En este proceso se inicia también la especificación del Plan de Pruebas, que se completará en el proceso Diseño del Sistema de Información (DSI).

En este proceso es muy importante la participación de los usuarios, a través de técnicas interactivas, como diseño de diálogos y prototipos, que permiten al usuario familiarizarse con el nuevo sistema y colaborar en la construcción y perfeccionamiento del mismo.

31.7.3.3 Diseño del Sistema de Información (DSI)

El propósito del Diseño del Sistema de Información (DSI) es obtener la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información. A partir de dicha información, se generan todas las

especificaciones de construcción relativas al propio sistema, así como la especificación técnica del plan de pruebas, la definición de los requisitos de implantación y el diseño de los procedimientos de migración y carga inicial, éstos últimos cuando proceda.

Este proceso consta de un primer bloque de actividades, que se realizan en paralelo, y cuyo objetivo es obtener el diseño de detalle del sistema de información que comprende la partición física del sistema de información, independiente de un entorno tecnológico concreto, la organización en subsistemas de diseño, la especificación del entorno tecnológico sobre el que se despliegan dichos subsistemas y la definición de los requisitos de operación, administración del sistema, seguridad y control de acceso. En el caso de diseño orientado a objetos, conviene señalar que se ha contemplado que el diseño de la persistencia se lleva a cabo sobre bases de datos relacionales.

Un segundo bloque de actividades complementa el diseño del sistema de información, en el que se generan todas las especificaciones necesarias para la construcción del sistema de información.

31.7.3.4 Construcción del Sistema de Información (CSI)

La construcción del Sistema de Información (CSI) tiene como objetivo final la construcción y prueba de los distintos componentes del sistema de información, a partir del conjunto de especificaciones lógicas y físicas del mismo, obtenido en el Proceso de Diseño del Sistema de Información (DSI). Se desarrollan los procedimientos de operación y seguridad y se elaboran los manuales de usuario final y de explotación, estos últimos cuando proceda.

Para conseguir dicho objetivo, se recoge la información relativa al producto del diseño Especificaciones de construcción del sistema de

información, se prepara el entorno de construcción, se genera el código de cada uno de los componentes del sistema de información y se van realizando, a medida que se vaya finalizando la construcción, las pruebas unitarias de cada uno de ellos y las de integración entre subsistemas.

Si fuera necesario realizar una migración de datos, es en este proceso donde se lleva a cabo la construcción de los componentes de migración y procedimientos de migración y carga inicial de datos.

31.7.3.5 Implantación y Aceptación del Sistema (IAS)

Este proceso tiene como objetivo principal, la entrega y aceptación del sistema en su totalidad, que puede comprender varios sistemas de información desarrollados de manera independiente, según se haya establecido en el proceso de Estudio de Viabilidad del Sistema (EVS), y un segundo objetivo que es llevar a cabo las actividades oportunas para el paso a producción del sistema.

Se establece el plan de implantación, una vez revisada la estrategia de implantación y se detalla el equipo que lo realizará. Para el inicio de este proceso se toman como punto de partida los componentes del sistema probados de forma unitaria e integrados en el proceso Construcción del Sistema de Información (CSI), así como la documentación asociada. El Sistema se someterá a las Pruebas de Implantación con la participación del usuario de operación cuya responsabilidad, entre otros aspectos, es comprobar el comportamiento del sistema bajo las condiciones más extremas. También se someterá a las Pruebas de Aceptación cuya ejecución es responsabilidad del usuario final. En este proceso se elabora el plan de mantenimiento del sistema de forma que el responsable del mantenimiento conozca el sistema antes de que éste pase a producción. También se establece el acuerdo de nivel de servicio requerido una vez que se inicie la producción.

31.7.4 Mantenimiento de Sistemas de Información (MSI)

El objetivo de este proceso es la obtención de una nueva versión de un sistema de información desarrollado con MÉTRICA, a partir de las peticiones de mantenimiento que los usuarios realizan con motivo de un problema detectado en el sistema o por la necesidad de una mejora del mismo. Sólo se considerarán en MÉTRICA Versión 3 los tipos de Mantenimiento **Correctivo** y **Evolutivo**. Ante una petición de cambio de un sistema de información ya en producción, se realiza un registro de las peticiones, se diagnostica el tipo de mantenimiento y se decide si se le da respuesta o no, en función del plan de mantenimiento asociado al sistema afectado por la petición, y se establece con qué prioridad. La definición de la solución al problema o necesidad planteada por el usuario que realiza el responsable de mantenimiento, incluye un estudio del impacto, la valoración del esfuerzo y coste, las actividades y tareas del proceso de desarrollo a realizar y el plan de pruebas de regresión.

31.7.5 Interfaces De Métrica Versión 3

La estructura de MÉTRICA Versión 3 incluye también un conjunto de interfaces que definen una serie de actividades de tipo organizativo o de soporte al proceso de desarrollo y a los productos, que en el caso de existir en la organización se deberán aplicar para enriquecer o influir en la ejecución de las actividades de los procesos principales de la metodología y que si no existen habrá que realizar para complementar y garantizar el éxito del proyecto desarrollado con MÉTRICA Versión 3. Son cuatro:

- **Gestión de Proyectos**: Tiene como finalidad principal la planificación, el seguimiento y control de las actividades y de los recursos humanos y materiales que intervienen en el desarrollo de un Sistema de Información. Como consecuencia de este control es posible conocer en

todo momento qué problemas se producen y resolverlos o paliarlos lo más pronto posible, lo cual evitará desviaciones temporales y económicas. Las actividades de la Interfaz de Gestión de Proyectos son de tres tipos:

- *Actividades de Inicio del Proyecto*, que permiten estimar el esfuerzo y establecer la planificación del proyecto.
- *Actividades de Seguimiento y Control*, supervisando la realización de las tareas por parte del equipo de proyecto y gestionando las incidencias y cambios en los requisitos que puedan presentarse y afectar a la planificación del proyecto.
- *Actividades de Finalización del Proyecto*, cierre y registro de la documentación de gestión.

- **Seguridad**: La interfaz de Seguridad hace posible incorporar durante la fase de desarrollo las funciones y mecanismos que refuerzan la seguridad del nuevo sistema y del propio proceso de desarrollo, asegurando su consistencia y seguridad, completando el plan de seguridad vigente en la organización o desarrollándolo desde el principio, utilizando MAGERIT como metodología de análisis y gestión de riesgos en el caso de que la organización no disponga de su propia metodología. Contempla dos tipos de actividades diferenciadas: las relacionadas con la seguridad intrínseca del sistema de información, y las que velan por la seguridad del propio proceso de desarrollo del sistema de información. Además se hace especial hincapié en la formación en materia de seguridad. Al ser finitos los recursos, no pueden asegurarse todos los aspectos del desarrollo de los sistemas de información, por lo que habrá que aceptar un determinado nivel de riesgo concentrándose en los aspectos más comprometidos o amenazados.
- **Gestión de la Configuración**: La interfaz de gestión de la configuración consiste en la aplicación de procedimientos administrativos y técnicos durante el desarrollo del sistema de

información y su posterior mantenimiento. Su finalidad es identificar, definir, proporcionar información y controlar los cambios en la configuración del sistema, así como las modificaciones y versiones de los mismos. Este proceso permitirá conocer el estado de cada uno de los productos que se hayan definido como elementos de configuración, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan. La gestión de configuración facilita además el mantenimiento del sistema, aportando información precisa para valorar el impacto de los cambios solicitados y reduciendo el tiempo de implementación de un cambio, tanto evolutivo como correctivo.

- **Aseguramiento de la Calidad**: El objetivo de la interfaz de Aseguramiento de la Calidad es proporcionar un marco común de referencia para la definición y puesta en marcha de planes específicos de aseguramiento de calidad aplicables a proyectos concretos. Las actividades están orientadas a verificar la calidad de los productos. Son actividades que evalúan la calidad y que son realizadas por un grupo de Asesoramiento de la Calidad independiente de los responsables de la obtención de los productos. Las actividades contempladas permitirán reducir, eliminar y prevenir las deficiencias de calidad de los productos a obtener, así como alcanzar una razonable confianza en que las prestaciones y servicios esperados por el cliente o el usuario queden satisfechas.

31.5 Metodologías Ágiles.

En las **metodologías ágiles**, la creación de valor mediante la adaptación a las necesidades cambiantes aparece en un primer plano frente a la tradicional idea de diseñar un plan y cumplir unos calendarios/requerimientos estáticos. Los proyectos gestionados con metodologías ágiles se inician sin un detalle cerrado de lo que va a ser

construido. A nivel comercial, los proyectos pueden ser vendidos como servicios y no como productos. Las características básicas de los proyectos gestionados con metodologías ágiles son las siguientes:

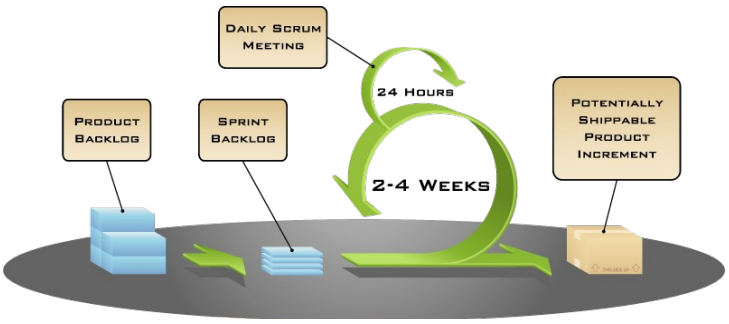
- **Incertidumbre**: la dirección indica la necesidad estratégica que se desea cubrir (sin entrar en detalles), ofreciendo máxima libertad al equipo de trabajo.
- **Equipos auto-organizados**: no existen roles especializados
    - Autonomía: libertad para la toma de decisiones.
    - Auto-superación: de forma periódica se evalúa el producto que se esta desarrollando.
    - Auto-enriquecimiento: transferencia del conocimiento.
- **Fases de desarrollo solapadas**: Las fases no existen como tal sino que se desarrollan tareas/actividades en función de las necesidades cambiantes durante todo el proyecto. De hecho, en muchas ocasiones no es posible realizar un diseño técnico detallado antes de empezar a desarrollar y ver algunos resultados. Por otra parte, las fases tradicionales efectuadas por personas diferentes no favorece el trabajo en equipo y pueden llegar a generar más inconvenientes que ventajas (por ej. un retraso en una fase, afecta a todo el proyecto).
- **Control sutil**: establecimientos de puntos de control para realizar un seguimiento adecuado sin limitar la libertad y creatividad del equipo. Así mismo, se recomienda:
    - Evaluar el ambiente laboral, siendo fundamental la elección de personas que no generen conflictos.
    - Reconocer los méritos mediante un sistema de evaluación justo y entender los errores como puntos de mejora y aprendizaje.
    - Potenciar la interacción entre el equipo y el negocio, para que puedan conocer las necesidades de primera mano.

- **Difusión y transferencia del conocimiento**: alta rotación de los miembros de los equipos entre diferentes proyectos. Por otra parte, potenciar el acceso libre a la información y documentación.

Algunas de las metodologías ágiles más conocidas las veremos en los apartados siguientes.

31.5.1 Scrum

Es un proceso de desarrollo de software iterativo e incremental utilizado comúnmente en entornos basados en la metodología Agile de desarrollo de software. Es un proceso marco que incluye un conjunto de prácticas y roles predefinidos. Los roles principales en Scrum son el ScrumMaster, que mantiene los procesos y trabaja de forma similar al director de proyecto, el ProductOwner, que representa a los stakeholders (clientes externos o internos), y el Team que incluye a los desarrolladores. Durante cada sprint, un periodo entre 15 y 30 días (la longitud es definida por el equipo), el equipo crea un incremento de software potencialmente entregable (utilizable). El conjunto de características que forma parte de cada sprint viene del **product backlog**, que es un conjunto de requisitos de alto nivel priorizados que dan forma al trabajo a realizar. Los elementos del backlog que forman parte del sprint se determinan durante la reunión de **sprint planning**. Durante esta reunión, el Product Owner informa al equipo de los elementos en el product backlog que quiere ver completados. El equipo entonces determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el sprint backlog, lo que significa que los requisitos están congelados durante el sprint. Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas amarillas "post-it" y pizarras hasta paquetes de software. Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar.



31.5.2 Dynamic Systems Development Method (DSDM)

Provee un framework para el desarrollo ágil de software, apoyado por la continua implicación del usuario en un desarrollo iterativo y creciente. DSDM fue desarrollado en el Reino Unido en los años 90. Como extensión del Desarrollo rápido de aplicaciones (RAD), DSDM se centra en los proyectos de sistemas de información que son caracterizados por presupuestos y agendas apretadas. DSDM trata los problemas que ocurren con frecuencia en el desarrollo de los sistemas de información en lo que respecta a pasar sobre tiempo y presupuesto y otras razones comunes para la falta en el proyecto tal como falta de implicación del usuario y de la comisión superior de la gerencia. DSDM consiste en 3 fases: fase del pre-proyecto, fase del ciclo de vida del proyecto, y fase del post-proyecto. La fase del ciclo de vida del proyecto se subdivide en 5 etapas: estudio de viabilidad, estudio de la empresa, iteración del modelo funcional, diseño e iteración de la estructura, e implementación. Tiene 9 principios fundamentales:

- **Involucrar al cliente** es la clave para llevar un proyecto eficiente y efectivo, donde ambos, cliente y desarrolladores, comparten un entorno de trabajo para que las decisiones puedan ser tomadas con precisión.
- **El equipo del proyecto debe tener el poder** para tomar decisiones que son importantes para el progreso del proyecto, sin esperar aprobación de niveles superiores.

- DSDM se centra en la **entrega frecuente de productos**, asumiendo que entregar algo temprano es siempre mejor que entregar todo al final. Al entregar el producto frecuentemente desde una etapa temprana del proyecto, el producto puede ser verificado y revisado allí donde la documentación de registro y revisión puede ser tenida en cuenta en la siguiente fase o iteración.
- El principal **criterio de aceptación** de entregables reside en entregar un sistema que satisface las actuales necesidades de negocio.
- El **desarrollo** es **iterativo e incremental**, guiado por la realimentación de los usuarios para converger en una solución de negocio precisa.
- Todos los **cambios** durante el desarrollo son reversibles.
- Requerimientos globales antes de comenzar el proyecto.
- Las pruebas son realizadas durante todo el ciclo vital del proyecto.
- La comunicación y cooperación entre todas las partes interesadas.

31.5.3 Extreme Programming (XP)

La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. XP construye un proceso de diseño evolutivo que se basa en refactorizar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras. Los Valores originales de la programación extrema

son: **simplicidad** (de diseño, código y documentación), **comunicación** (la comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide que características tienen prioridad y siempre debe estar disponible para solucionar dudas), **retroalimentación** (*feedback.* Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.), y **coraje** (Se requiere coraje para implementar las características que el cliente quiere ahora sin caer en la tentación de optar por un enfoque más flexible que permita futuras modificaciones). Un quinto valor, **respeto** (los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, sino orientándolos a realizarlo mejor, obteniendo como resultado una mejor autoestima en el equipo y elevando el ritmo de producción en el equipo**)**, fue añadido posteriormente.

Las características fundamentales de esta metodología son:

- Desarrollo iterativo e incremental
- Pruebas unitarias continuas
- Programación en parejas
- Integración del equipo de programación con el cliente
- Corrección de todos los errores
- Refactorización del código
- Propiedad del código compartida
- Simplicidad en el código

31.5.4 Agile Modeling (AM)

Se puede describir como una metodología basada en la práctica para el modelado efectivo de sistemas de software. No define procedimientos detallados de cómo crear un tipo de modelo dado. En lugar de eso, sugiere prácticas para que los modelos y documentación sean efectivos. Su secreto no está en las técnicas de modelado a usar, sino en como se aplican. No es

un desarrollo de software completo ya que no cubre actividades de programación, prueba, gestión de proyectos, implementación, soporte u otros elementos de la realización de proyectos que no sean la documentación y el modelado. Es necesario, por lo tanto, combinarlo con otras metodologías como pueden ser XP, DSDM, SCRUM o RUP. Los valores de esta metodología son la **comunicación** (entre participantes del equipo de trabajo, desarrolladores y analistas, etc.), **simplicidad**, **coraje** (para tomar decisiones importantes y ser capaces de cambiar de dirección cuando el camino tomado no es el correcto) y **humildad** (todos los interesados en el proyecto pueden contribuir en algo para la mejor realización).

31.5.5 *Feature Driven Development (FCC):*

Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar. Las iteraciones se deciden en base a funcionalidades, que son pequeñas partes del software con significado para el cliente**. N**o cubre todo el ciclo de vida sino sólo las fases de diseño y construcción. No requiere un modelo específico de proceso y se complementa con otras metodologías. FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema:

- **Desarrollo de un modelo general**: Cuando comienza esta fase, los expertos del dominio ya tienen una idea del contexto y los requerimientos del sistema. El dominio global es dividido en diferentes áreas y se realiza informe detallado para cada una de ellas por parte de los expertos del dominio.
- **Construcción de la lista de funcionalidades** Los ensayos, modelos de objetos y documentación de requerimientos proporcionan la base para construir una amplia lista de funcionalidades. Estas

funcionalidades son pequeños ítems útiles a los ojos del cliente. La lista de funcionalidades es revisada por los usuarios y patrocinadores para asegurar su validez. Las funcionalidades que requieran de más de diez días se descomponen en otras más pequeñas.

- **Planeamiento por funcionalidades**: En esta etapa se incluye la creación de un plan de alto nivel, en el cual la lista de funcionalidades es ordenada en base a la prioridad y a la dependencia entre cada funcionalidad. Además, las clases identificadas en la primera etapa son asignadas a cada programador.
- **Diseño y construcción por funcionalidades**: El diseño y construcción de la funcionalidad es un proceso iterativo durante el cual las funcionalidades seleccionadas son producidas. Una iteración puede llevar desde unos pocos días a un máximo de dos semanas. Este proceso iterativo incluye tareas como inspección del diseño, codificación, pruebas unitarias, integración e inspección del código.

31.5.6 Familia Cristal

Alistair Cockburn es el propulsor detrás de la serie de metodologías Crystal. Es una familia porque él cree que los tipos diferentes de proyectos requieren tipos diferentes de metodologías. Él mira esta variación a lo largo de dos ejes: el número de personas en el proyecto, y las consecuencias de los errores. Dispone un código de **color** para marcar la complejidad de cada metodología. Comparte con la XP una orientación humana, pero esta centralización en la gente se hace de una manera diferente. Alistair considera que las personas encuentran difícil seguir un proceso disciplinado, así que más que seguir la alta disciplina de la XP, Alistair explora la metodología menos disciplinada que aun podría tener éxito, intercambiando conscientemente productividad por facilidad de ejecución. Él considera que aunque Cristal es menos productivo que la XP, más personas serán capaces de seguirlo. Alistair también pone mucho peso en

las revisiones al final de la iteración, animando al proceso a ser "automejorable". Defiende que el desarrollo iterativo está para encontrar los problemas temprano, y entonces permitir corregirlos. Esto pone más énfasis en la gente supervisando su proceso y afinándolo conforme desarrollan.

Bibliografía

- Ingeniería del Software. Un enfoque práctico. ROGER S. PRESSMAN. Ed. McGraw Hill
- Ingeniería de Software 7 Edición - Ian Sommerville
- Metodologías para la gestión y desarrollo de software - ¿?
- http://msdn.microsoft.com/es-es/library/bb972268.aspx Desarrollo basado en Componentes.
- http://alarcos.inf-cr.uclm.es/per/fruiz/cur/pso/trans/res.pdf Curso de Proceso Software: Conceptos, Estándares, Modelos, Arquitecturas y Herramientas. Francisco Ruiz
- http://administracionelectronica.gob.es/archivos/pae_000001027.pdf Introducción a Métrica Versión 3. Ministerio de Administraciones Públicas
- Análisis, Diseño y Mantenimiento del Software. José Ramón Álvarez Sánchez y Manuel Arias Calleja. Dpto. de Inteligencia Artificial - ETSI Informática - UNED
- http://www.marblestation.com/?p=661. Metodologías ágiles de gestión de proyectos. Sergi Blanco Cuaresma.
- http://es.wikipedia.org/wiki/DSDM
- Agile Modeling (AM) Felipe Ferrada.
- http://www.ort.edu.uy/fi/publicaciones/ingsoft/investigacion/ayudantias/metodologia_FDD.pdf. Metodología FDD. Cátedra de Ingeniería de Software. Luis Calabria. Universidad ORT Uruguay
- http://www.programacionextrema.org/articulos/newMethodology.es.html La Nueva Metodología. Martin Fowler

Autor: Hernán Vila Pérez
Jefe del Servicio de Informática. Instituto Galego de Vivenda e Solo
Vicepresidente del CPEIG