XUNTA DE GALICIA

ESCOLA GALEGA DE ADMINISTRACIÓN PÚBLICA

33. INGENIERÍA DEL SOFTWARE. PROCESO SOFTWARE, MODELOS DE PROCESO SOFTWARE. CICLOS DE VIDA. MODELOS DE CICLO DE VIDA. FASES DEL CICLO DE VIDA. MODELOS DE DESARROLLO.

Fecha de actualización 26 de julio de 2011

XUNTA DE GALICIA

ESCOLA GALEGA DE ADMINISTRACIÓN PÚBLICA

Tema 33. 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.

INDICE

33.1 Ingeniería del software

33.1.1 Inicios de la ingeniería del Software.

El auge que se produjo en el ámbito de la informática en los años sesenta, debido en su mayor parte a la aparición de la segunda generación de ordenadores, tuvo como consecuencia que se realizara una masiva escritura incontrolada de líneas de código. Este proceso de creación de software se llevó a cabo sin plantearse ningún tipo de metodología para el diseño y construcción de software, ni ningún método para solventar los problemas relacionados con el 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.

33.1.2 ¿Qué es la Ingeniería del Software?

Según Pressman, la Ingeniería del software se puede definir como *el establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico que sea fiable y funcione de manera eficiente sobre máquinas reales*.

La Ingeniería del Software abarca tres elementos clave:

- **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.

- **Herramientas:** suministran el soporte automático o semiautomático para los métodos; esto es, dan soporte al desarrollo del software.

- **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.

Pressman divide en tres fases el trabajo asociado a la ingeniería del software:

- **La fase de definición** (el *qué).* 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** *(*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** *(*el *cambio).* El cambio 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.

Todas estas fases van acompañadas de un conjunto de actividades que se realizan a lo largo de todo el proceso de creación de software. Estas actividades se denominan actividades protectoras, siendo las más importantes:

- Seguimiento y control del proyecto
- Revisiones técnicas formales
- Garantía de calidad del software
- Gestión de configuración del software

- Preparación y producción de documentos
- Gestión de reutilización
- Mediciones
- Gestión de riesgos

33.2 Proceso software. Modelos de proceso software

Existen diversas definiciones formales para determinar el concepto de *Proceso de Software:*

- *Un proceso del software es un conjunto de actividades que conducen a la creación de un producto software*. Esta es una definición propuesta por Sommerville, en donde estas actividades pueden consistir en el desarrollo de software desde cero, desarrollo de nuevo software ampliando y modificando sistemas existentes o configurando e integrando software comercial o componentes del sistema.
- Desde otro punto de vista, Fugetta determina un proceso software como *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.

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.

33.3 Ciclo de vida del software

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*.

Existen diversos modelos de ciclo de vida, los cuales determinan una serie de etapa, fases o estados por los que ha de pasar un producto software. Esta diversidad es debida en gran medida al hecho de que existen multitud de aplicaciones diferentes y de distinto índole, lo que provoca que existan modelos de ciclo de vida que se ajusten mejor a unos desarrollos que a otros.

Sin embargo, a pesar de esta variedad, 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.

33.4 Modelos de ciclo de vida del software

Los distintos modelos de clico de vida del software pueden ser clasificados siguiendo diversos criterios. Siguiendo un criterio basándose en la utilización de los mismos existen:

- **Modelos tradicionales**: Son los de más amplia utilización.
  - Modelo Codificar y Corregir
  - Modelo por Etapas
  - Modelo en Cascada.
  - Modelos Evolutivos
    - Modelo Iterativo incremental
    - Modelo en Espiral
    - Modelos basados en prototipos:
      - Modelo de construcción de prototipos.
      - Modelo de prototipado evolutivo.
- **Modelos alternativos**
  - 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).

33.4.1 Modelo Codificar y Corregir (Code and Fix)

Este es un modelo simple, utilizado en los primeros desarrollos de software, el cual se fundamenta en la creación del código y en la corrección y adaptación del mismo. 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.

33.4.2 Modelo por Etapas

El modelo por etapas nace como respuesta a los problemas que surgen al utilizar el modelo anterior. Como solución se plantea un modelo para la creación de software que se basa en un conjunto de etapas o fases sucesivas que van construyendo el sistema planteado. Las etapas determinadas por este modelo (Stage Wise) son:

- Planificación
- Especificaciones de operación

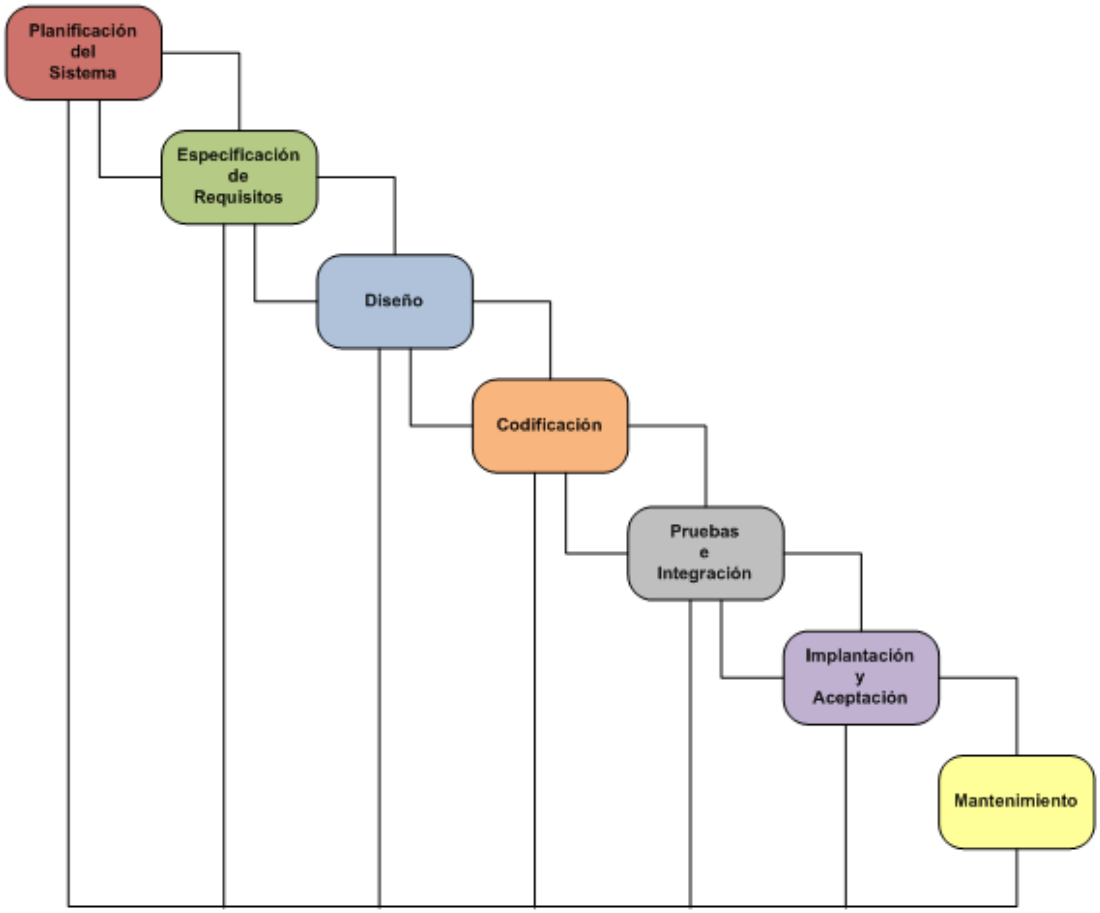
- Especificaciones de codificación
- Codificación
- Prueba de cada unidad
- Prueba de integración
- Eliminación de problemas
- Evaluación del sistema.

33.4.3 Modelo en Cascada

Este modelo se fundamente en el modelo por etapas, al cual incorpora un conjunto de mejoras 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.

33.4.4 Modelos Evolutivos

El software evoluciona con el tiempo. Los requisitos del usuario y del producto suelen cambiar conforme se desarrolla el mismo. Las fechas de mercado y la competencia hacen que no sea posible esperar a poner en el mercado un producto absolutamente completo, por lo que se debe

introducir una versión funcional limitada de alguna forma para aliviar las presiones competitivas.

En esas u otras situaciones similares los desarrolladores necesitan modelos de progreso que estén diseñados para acomodarse a una evolución temporal o progresiva, donde los requisitos centrales son conocidos de antemano, aunque no estén bien definidos a nivel detalle.

Los evolutivos son modelos iterativos, permiten desarrollar versiones cada vez más completas y complejas, hasta llegar al objetivo final deseado; incluso evolucionar más allá, durante la fase de operación.

33.4.4.1 Modelo Iterativo Incremental

El incremental *es un modelo de tipo evolutivo que está basado en varios ciclos cascada realimentados aplicados repetidamente, con una filosofía iterativa*, es decir, consiste en desarrollar un sistema que logré cubrir una parte de los requisitos especificados y luego ir generando nuevas versiones del sistema que incorporen el resto de funcionalidades y requisitos especificados, hasta llegar a un producto final, que se asemeje al sistema planteado.

Con este modelo, 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.

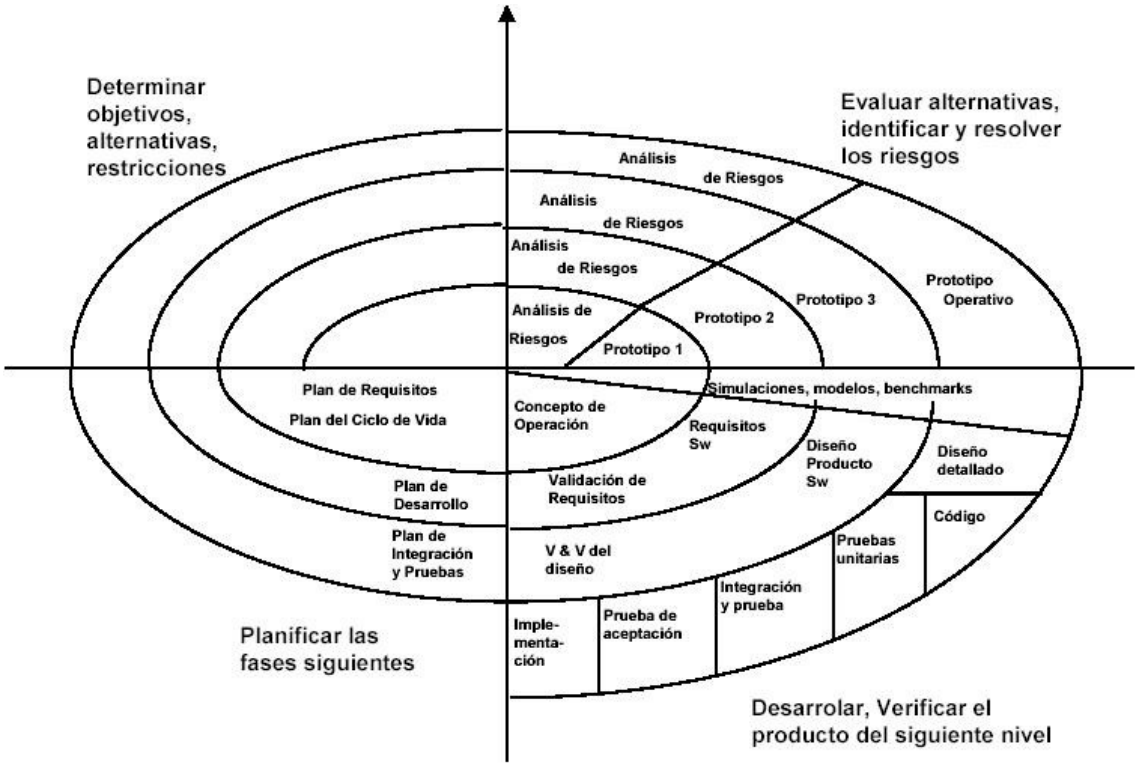
33.4.4.2 Modelo en Espiral

Es un modelo de proceso de software evolutivo con las características propias de un desarrollo iterativo mediante el cual se generan distintos prototipos, a las cuales se le suman los aspectos controlados y sistemáticos del modelo en cascada.

Ofrece 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.

Sus diferencias más importantes con los modelos más clásicos 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.

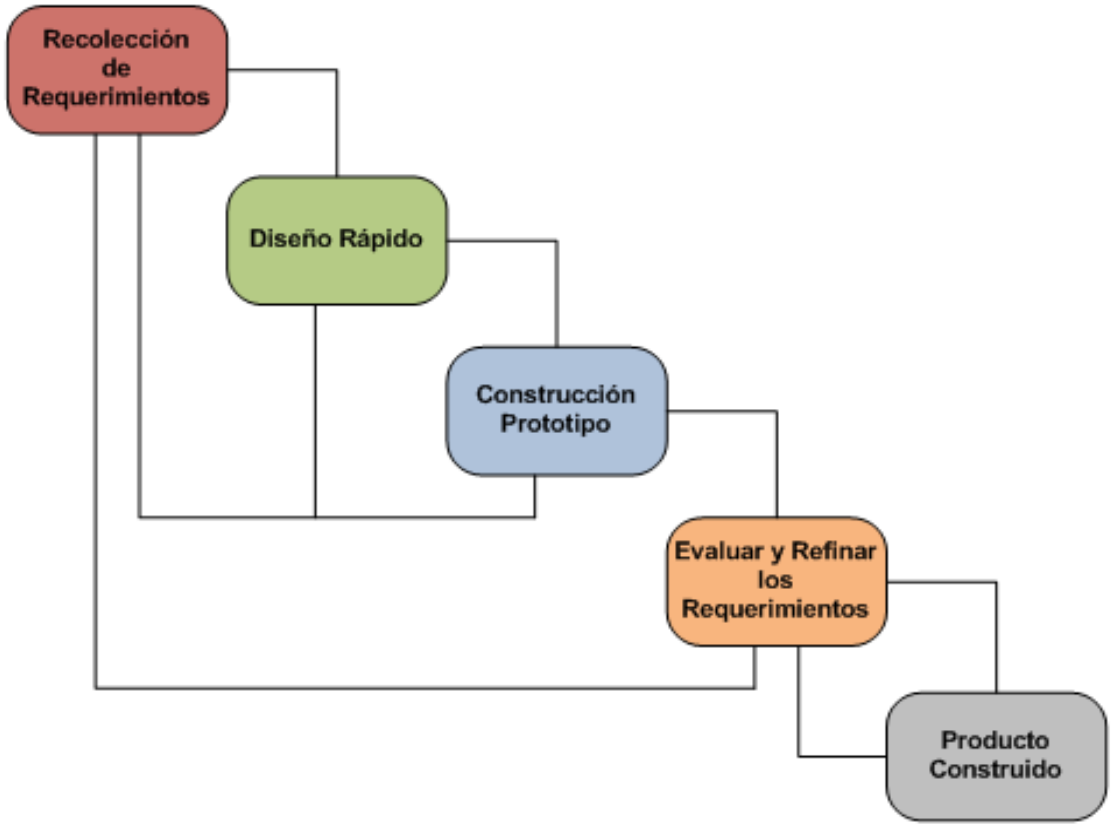
33.4.4.3 Modelos basados en prototipos

Los modelos basados en prototipos se centran en la idea de ofrecer una mayor comprensión de los requisitos que el usuario plantea, sobre todo si este no posee una idea clara y concreta de sus pretensiones.

Además, este tipo de modelos puede utilizarse para intentar valorar de una manera temprana la viabilidad de la solución propuesta, cuando no se confía plenamente en ella.

33.4.4.3.1 Prototipado rápido

Este modelo se fundamenta en la construcción de prototipos de una manera fácil, barata y en un reducido período de tiempo, permitiendo así su temprana evaluación. También se denominan de usar y tirar.



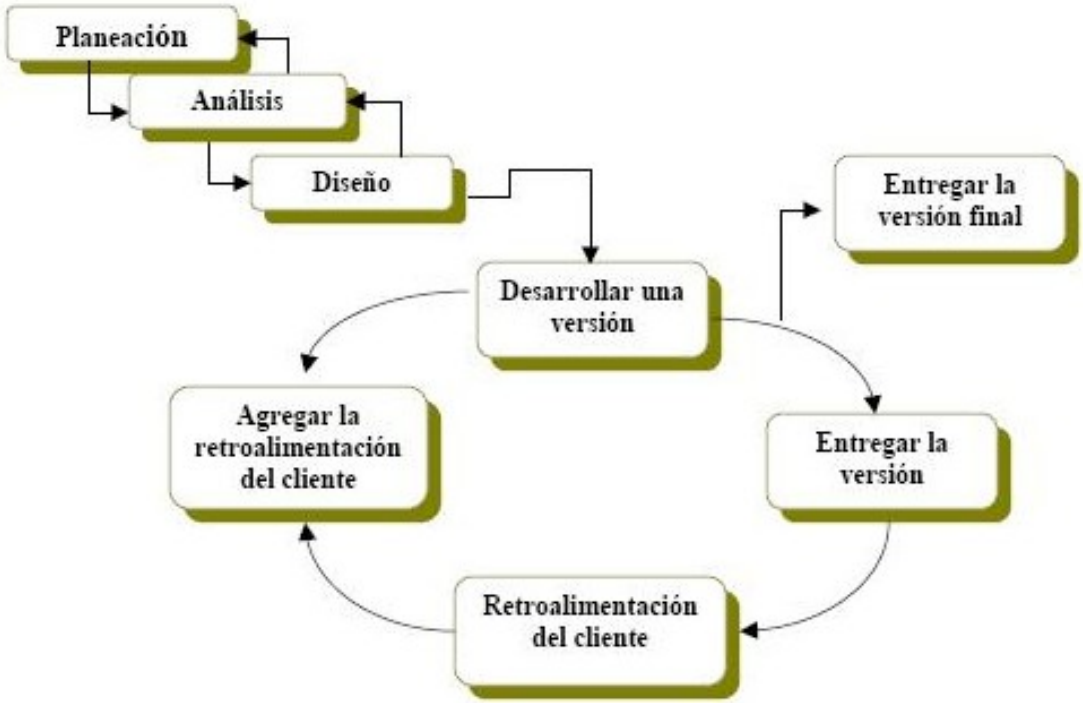
El prototipo sirve para crear y validar la especificación, y para que el usuario tenga una idea de cómo será el software antes de que comience el desarrollo. Es importante precisar que el prototipo se construye sólo para servir como mecanismo de definición de los requerimientos funcionales. Posteriormente 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.

33.4.4.3.2 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.

33.4.5 Modelos basados en Transformaciones

Surgen como solución al problema que plantean los modelos de desarrollo que producen software con problemas estructurales. Su principal virtud es que ofrecen la posibilidad de convertir de una manera automáticamente una especificación formal de un sistema en un software que cumpla lo establecido en los requisitos.

Los pasos más importantes que siguen este tipo de 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:

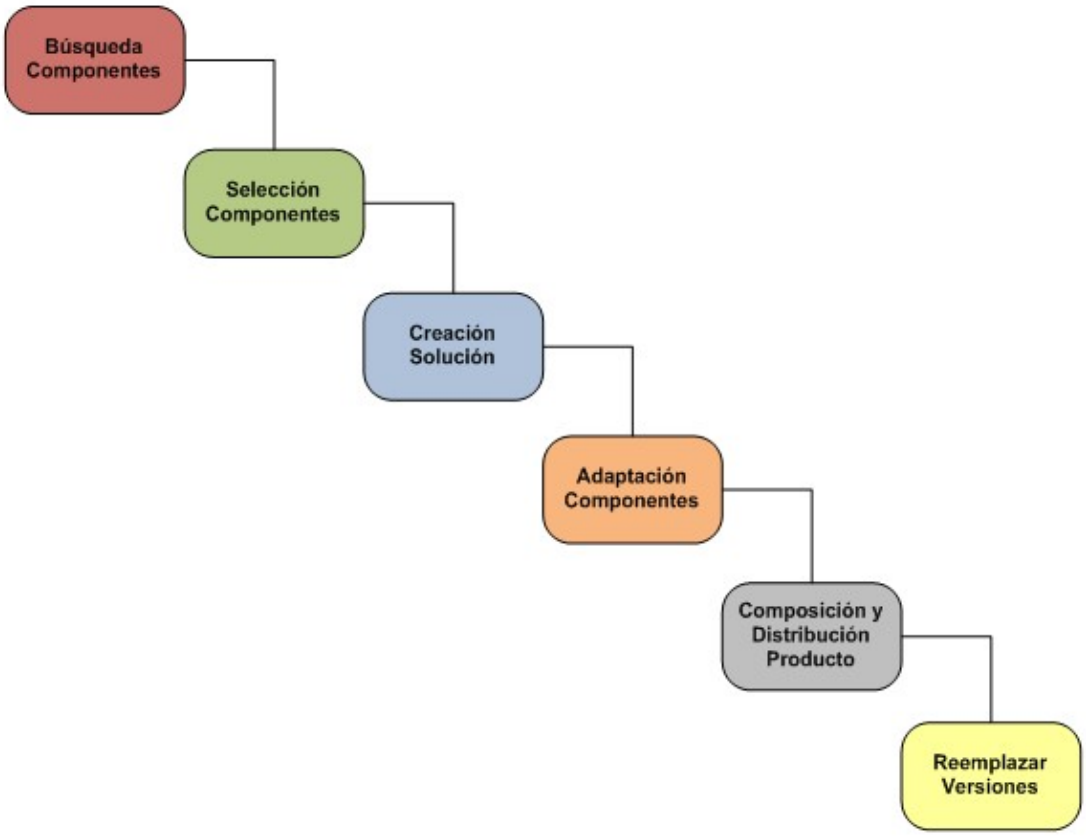
- Los que usan técnicas de cuarta generación (Roger Pressman): Suelen estar basados en herramientas de cuarta generación. Estos permiten la generación de código rápido. En ellos se indica qué se quiere obtener, no cómo.
- Basados en modelos de transformación (Carma McClure) => Basados en herramientas 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.

33.4.6 Modelo basado en componentes

El modelo basado en componentes surge de la necesidad de reutilización que los complejos sistema actuales precisan para acelerar su desarrollo. Este modelo posibilita que ciertas piezas de código preelaboradas puedan ser reutilizadas en otras partes del sistema o incluso en otros sistemas para llevar a cabo 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.

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

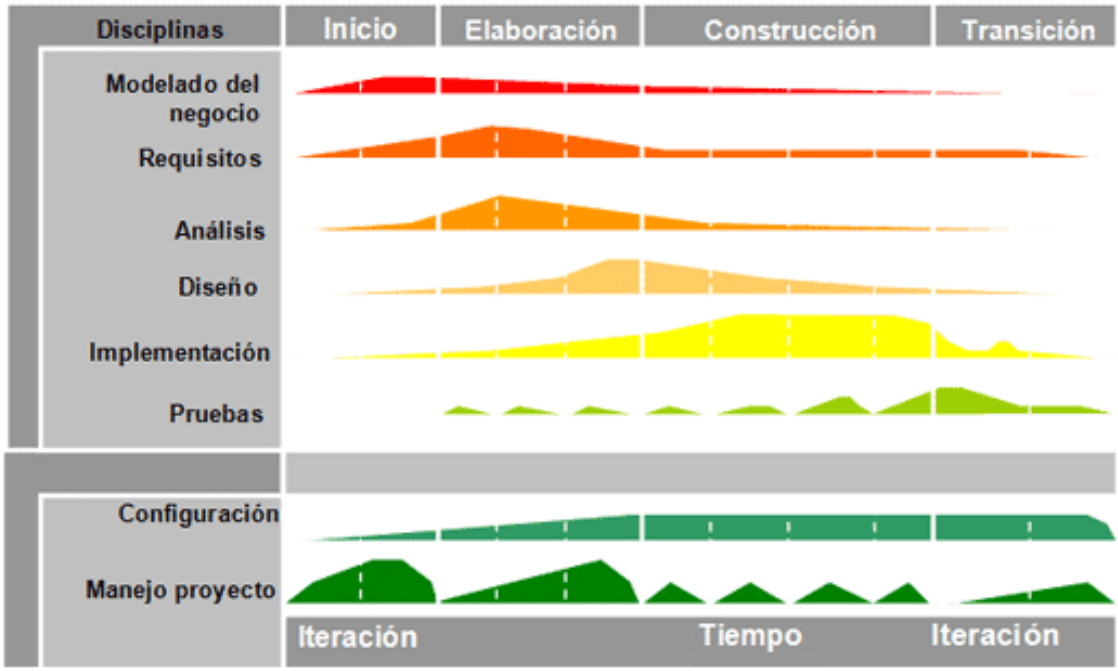
métodos sistemáticos y repetibles para evaluar y seleccionar dichos componentes es un aspecto importante para el desarrollo del software basado en componentes y, en general, para la Ingeniería del Software Basada en Componentes (ISBC).

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.

33.5 Modelos de Desarrollo

33.5.1 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.

33.5.2 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.

Bibliografía

- Ingeniería del Software. Un enfoque práctico. ROGER S. PRESSMAN. Ed. McGraw Hill

- Ingeniería de Software 7 Edición - Ian Sommerville

- Ingeniería de Sistemas de Software – Gonzalo León Serrano. Ed. Isdefe.

- Metodologías para la gestión y desarrollo de software

Autor: Francisco Javier Rodríguez Martínez.

Subdirector de la Escuela Superior de Ingeniería Informática.

Universidad de Vigo.