Proceso de desarrollo de software

Introducción

Un sistema informático está compuesto por hardware y software. En cuanto al hardware, su producción se realiza sistemáticamente y la base de conocimiento para el desarrollo de dicha actividad está claramente definida. La fiabilidad del hardware es, en principio, equiparable a la de cualquier otra máquina construida por el hombre. Sin embargo, respecto del software, su construcción y resultados han sido históricamente cuestionados debido a los problemas asociados, entre ellos podemos destacar los siguientes [1]:

- Los sistemas no responden a las expectativas de los usuarios.
- Los programas “fallan” con cierta frecuencia.
- Los costes del software son difíciles de prever y normalmente superan las estimaciones.
- La modificación del software es una tarea difícil y costosa.
- El software se suele presentar fuera del plazo establecido y con menos prestaciones de las consideradas inicialmente.
- Normalmente, es difícil cambiar de entorno hardware usando el mismo software.
- El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, etc.) no suele cumplirse.

Según el Centro Experimental de Ingeniería de Software (CEIS)[1], el estudio de mercado *The Chaos Report* realizado por Standish Group Internactional[2] en 1996, concluyó que sólo un 16% de los proyectos de software son exitosos (terminan dentro de plazos y costos y cumplen los requerimientos acordados). Otro 53% sobrepasa costos y plazos y cumple parcialmente los requerimientos. El resto ni siquiera llega al término. Algunas deficiencias comunes en el desarrollo de software son:

- Escasa o tardía validación con el cliente.
- Inadecuada gestión de los requisitos.
- No existe medición del proceso ni registro de datos históricos.
- Estimaciones imprevistas de plazos y costos.
- Excesiva e irracional presión en los plazos.
- Escaso o deficiente control en el progreso del proceso de desarrollo.
- No se hace gestión de riesgos formalmente.
- No se realiza un proceso formal de pruebas.
- No se realizan revisiones técnicas formales e inspecciones de código.

El primer reconocimiento público de la existencia de problemas en la producción de software tuvo lugar en la conferencia organizada en 1968 por la Comisión de Ciencias de la OTAN en Garmisch (Alemania), dicha situación problemática se denominó **crisis del software**. En esta conferencia, así como en la siguiente realizada en Roma en 1969, se estipuló el interés hacia los aspectos técnicos y administrativos en el desarrollo y mantenimiento de productos software. Se pretendía acordar las bases para una ingeniería de construcción de software. Según Fritz Bauer [2] lo que se necesitaba era “*establecer y usar principios de ingeniería orientados a obtener software de manera económica, que sea fiable y funcione eficientemente sobre máquinas reales*”. Esta definición marcaba posibles cuestiones tales como: ¿Cuáles son los principios robustos de la ingeniería aplicables al desarrollo de software de computadora? ¿Cómo construimos el software económicamente para que sea fiable? ¿Qué se necesita para crear programas de computadora que funcionen eficientemente no en una máquina sino en diferentes máquinas reales?. Sin embargo, dicho planteamiento además debía incluir otros aspectos, tales como: mejora de la calidad del software, satisfacción del cliente, mediciones y métricas, etc.

[1] http://www.ceis.cl/Gestacion/Gestacion.htm (5.3.2003)
[2] http://standishgroup.com/ (5.3.2003)

El "*IEEE Standard Glossary of Software Engineering Terminology*" (Stad. 610.12-1990) ha desarrollado una definición más completa para ingeniería del software [1]: "(1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software. (2) El estudio de enfoques en (1)".

Pressman [1] caracteriza la Ingeniería de Software como "una tecnología multicapa", ilustrada en la Figura 1.

Figura 1: Capas de la Ingeniería de Software.

Dichas capas se describen a continuación:

- Cualquier disciplina de ingeniería (incluida la ingeniería del software) debe descansar sobre un esfuerzo de organización de **calidad**. La gestión total de la calidad y las filosofías similares fomentan una cultura continua de mejoras de procesos que conduce al desarrollo de enfoques cada vez más robustos para la ingeniería del software.
- El fundamento de la ingeniería de software es la **capa proceso**. El proceso define un marco de trabajo para un conjunto de áreas clave, las cuales forman la base del control de gestión de proyectos de software y establecen el contexto en el cual: se aplican los métodos técnicos, se producen resultados de trabajo, se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.
- Los **métodos** de la ingeniería de software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.
- Las **herramientas** de la ingeniería del software proporcionan un soporte automático o semi-automático para el proceso y los métodos, a estas herramientas se les llama herramientas CASE (C*omputer-Aided Software Engineering*).

Dado lo anterior, el objetivo de la ingeniería de software es lograr productos de software de calidad (tanto en su forma final como durante su elaboración), mediante un proceso apoyado por métodos y herramientas.

A continuación nos enfocaremos en el proceso necesario para elaborar un producto de software.

El proceso de desarrollo del software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente. Dicho proceso, en términos globales se muestra en la Figura 2 [3]. Este proceso es intensamente intelectual, afectado por la creatividad y juicio de las personas involucradas [4]. Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales, relativos esencialmente a la naturaleza del producto obtenido. A continuación se explican algunas particularidades asociadas al desarrollo de software y que influyen en su proceso de construcción.

Un producto software en sí es complejo, es prácticamente inviable conseguir un 100% de confiabilidad de un programa por pequeño que sea. Existe una inmensa combinación de factores que impiden una verificación exhaustiva de las todas posibles situaciones de ejecución que se puedan presentar (entradas, valores de variables, datos almacenados, software del sistema, otras aplicaciones que intervienen, el hardware sobre el cual se ejecuta, etc.).

Un producto software es intangible y por lo general muy abstracto, esto dificulta la definición del producto y sus requisitos, sobre todo cuando no se tiene precedentes en productos software similares. Esto hace que los requisitos sean difíciles de consolidar tempranamente. Así, los cambios en los requisitos son inevitables, no sólo después de entregado en producto sino también durante el proceso de desarrollo.

Además, de las dos anteriores, siempre puede señalarse la inmadurez de la ingeniería del software como disciplina, justificada por su corta vida comparada con otras disciplinas de la ingeniería. Sin embargo, esto

no es más que un inútil consuelo.

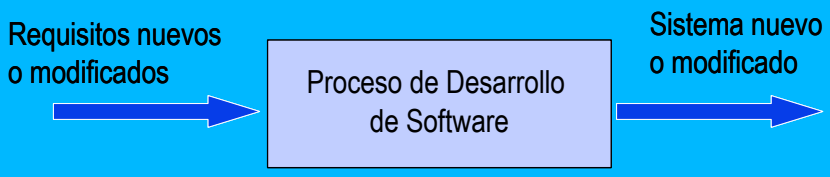

Figura 2: proceso de desarrollo de software.

El proceso de desarrollo de software no es único. No existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo. Debido a esta diversidad, es difícil automatizar todo un proceso de desarrollo de software.

A pesar de la variedad de propuestas de proceso de software, existe un conjunto de actividades fundamentales que se encuentran presentes en todos ellos [4]:

1. **Especificación de software**: Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software.
2. **Diseño e Implementación**: Se diseña y construye el software de acuerdo a la especificación.
3. **Validación**: El software debe validarse, para asegurar que cumpla con lo que quiere el cliente.
4. **Evolución**: El software debe evolucionar, para adaptarse a las necesidades del cliente.

Además de estas actividades fundamentales, Pressman [1] menciona un conjunto de "actividades protectoras", que se aplican a lo largo de todo el proceso del software. Ellas se señalan a continuación:

- Seguimiento y control de 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.
- Gestión de riesgos.

Pressman [1] caracteriza un proceso de desarrollo de software como se muestra en la Figura 3. Los elementos involucrados se describen a continuación:

- **Un marco común del proceso**, definiendo un pequeño número de actividades del marco de trabajo que son aplicables a todos los proyectos de software, con independencia del tamaño o complejidad.
- **Un conjunto de tareas**, cada uno es una colección de tareas de ingeniería del software, hitos de proyectos, entregas y productos de trabajo del software, y puntos de garantía de calidad, que permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software y los requisitos del equipo del proyecto.
- **Las actividades de protección**, tales como garantía de calidad del software, gestión de configuración del software y medición, abarcan el modelo del proceso. Las actividades de protección son independientes de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.

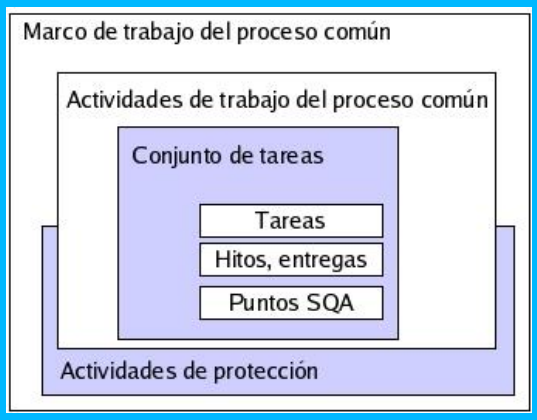

Figura 3: Elementos del proceso del software

Otra perspectiva utilizada para determinar los elementos del proceso de desarrollo de software es establecer las relaciones entre elementos que permitan responder **Quién** debe hacer **Qué**, **Cuándo** y **Cómo** debe hacerlo [5].

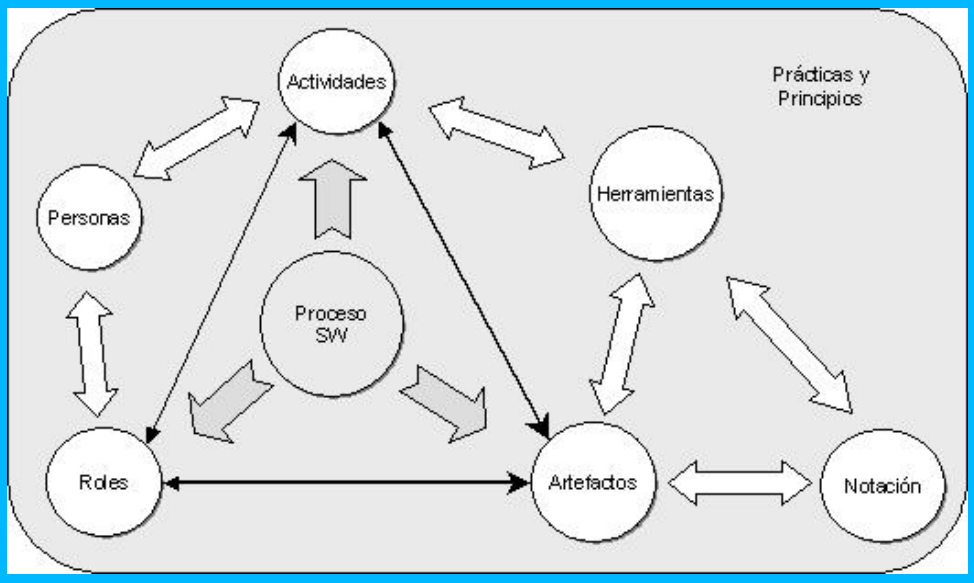

Figura 4: Relación entre elementos del proceso del software

En la Figura 4 se muestran los elementos de un proceso de desarrollo de software y sus relaciones. Así las interrogantes se responden de la siguiente forma:

- **Quién:** Las Personas participantes en el proyecto de desarrollo desempeñando uno o más Roles específicos.
- **Qué:** Un Artefacto[3] es producido por un Rol en una de sus Actividades. Los Artefactos se especifican utilizando Notaciones específicas. Las Herramientas apoyan la elaboración de Artefactos soportando ciertas Notaciones.
- **Cómo y Cuándo:** Las Actividades son una serie de pasos que lleva a cabo un Rol durante el proceso de desarrollo. El avance del proyecto está controlado mediante hitos que establecen un determinado estado de terminación de ciertos Artefactos.

[3] Un artefacto es una pieza de información que (1) es producida, modificada o usada por el proceso, (2) define un área de responsabilidad para un rol y (3) está sujeta a control de versiones. Un artefacto puede ser un modelo, un elemento de modelo o un documento.

La composición y sincronía de las actividades está basada en un conjunto de Principios y Prácticas. Las Prácticas y Principios enfatizan ciertas actividades y/o la forma como deben realizarse, por ejemplo: desarrollar iterativamente, gestionar requisitos, desarrollo basado en componentes, modelar visualmente, verificar continuamente la calidad, gestionar los cambios, etc.

Modelos de proceso software

Sommerville [4] define modelo de proceso de software como “Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real.”

Los modelos genéricos no son descripciones definitivas de procesos de software; sin embargo, son abstracciones útiles que pueden ser utilizadas para explicar diferentes enfoques del desarrollo de software.

Modelos que se van a discutir a continuación:

- Codificar y corregir
- Modelo en cascada
- Desarrollo evolutivo
- Desarrollo formal de sistemas
- Desarrollo basado en reutilización
- Desarrollo incremental
- Desarrollo en espiral

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 [7]:

- Después de un número de correcciones, el código puede tener una muy mala estructura, hace que los arreglos sean muy costosos.
- 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.
- El código es difícil de reparar por su pobre preparación para probar y modificar.

Modelo en cascada

El primer modelo de desarrollo de software que se publicó se derivó de otros procesos de ingeniería [8]. Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.

El modelo en cascada consta de las siguientes fases:

1. Definición de los requisitos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle.
2. Diseño de software: Se particiona el sistema en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema.
3. Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
4. Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se

entrega el conjunto probado al cliente.

5. Operación y mantenimiento: Generalmente es la fase más larga. El sistema es puesto en marcha y se realiza la corrección de errores descubiertos. Se realizan mejoras de implementación. Se identifican nuevos requisitos.

La interacción entre fases puede observarse en la Figura 5. Cada fase tiene como resultado documentos que deben ser aprobados por el usuario.

Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.

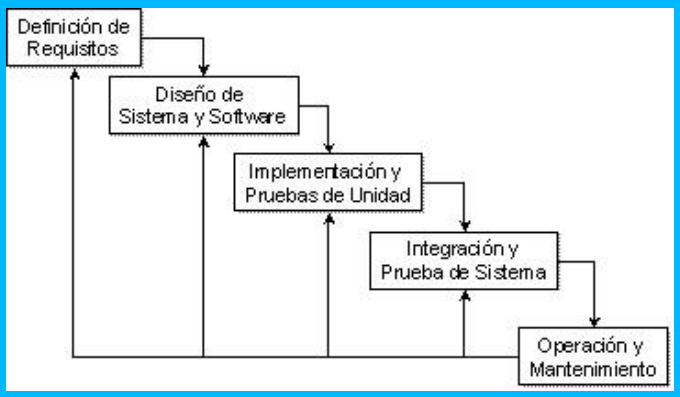

Figura 5: Modelo de desarrollo en cascada.

En la práctica, este modelo no es lineal, e involucra varias iteraciones e interacción entre las distintas fases de desarrollo. Algunos problemas que se observan en el modelo de cascada son:

- Las iteraciones son costosas e implican rehacer trabajo debido a la producción y aprobación de documentos.
- Aunque son pocas iteraciones, es normal congelar parte del desarrollo y continuar con las siguientes fases.
- Los problemas se dejan para su posterior resolución, lo que lleva a que estos sean ignorados o corregidos de una forma poco elegante.
- Existe una alta probabilidad de que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto.
- Es inflexible a la hora de evolucionar para incorporar nuevos requisitos. Es difícil responder a cambios en los requisitos.

Este modelo sólo debe usarse si se entienden a plenitud los requisitos. Aún se utiliza como parte de proyectos grandes.

Desarrollo evolutivo

La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado. En la Figura 6 se observa cómo las actividades concurrentes: especificación, desarrollo y validación, se realizan durante el desarrollo de las versiones hasta llegar al producto final.

Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

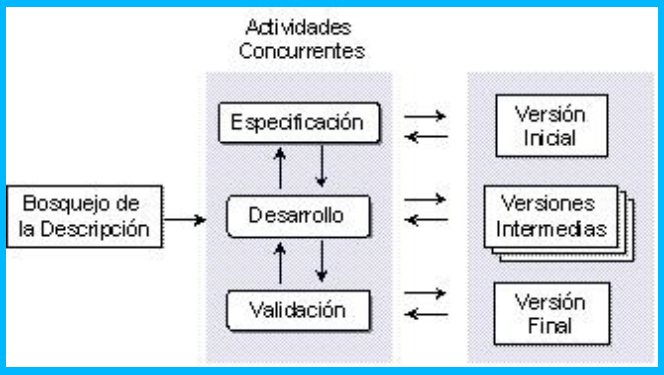

Figura 6: Modelo de desarrollo evolutivo.

Existen dos tipos de desarrollo evolutivo:

- Desarrollo Exploratorio: El objetivo de este enfoque es explorar con el usuario los requisitos hasta llegar a un sistema final. El desarrollo comienza con las partes que se tiene más claras. El sistema evoluciona conforme se añaden nuevas características propuestas por el usuario.
- Enfoque utilizando prototipos: El objetivo es entender los requisitos del usuario y trabajar para mejorar la calidad de los requisitos. A diferencia del desarrollo exploratorio, se comienza por definir los requisitos que no están claros para el usuario y se utiliza un prototipo para experimentar con ellos. El prototipo ayuda a terminar de definir estos requisitos.

Entre los puntos favorables de este modelo están:

- La especificación puede desarrollarse de forma creciente.
- Los usuarios y desarrolladores logran un mejor entendimiento del sistema. Esto se refleja en una mejora de la calidad del software.
- Es más efectivo que el modelo de cascada, ya que cumple con las necesidades inmediatas del cliente.

Desde una perspectiva de ingeniería y administración se identifican los siguientes problemas:

- Proceso no Visible: Los administradores necesitan entregas para medir el progreso. Si el sistema se necesita desarrollar rápido, no es efectivo producir documentos que reflejen cada versión del sistema.
- Sistemas pobremente estructurados: Los cambios continuos pueden ser perjudiciales para la estructura del software haciendo costoso el mantenimiento.
- Se requieren técnicas y herramientas: Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

Este modelo es efectivo en proyectos pequeños (menos de 100.000 líneas de código) o medianos (hasta 500.000 líneas de código) con poco tiempo para su desarrollo y sin generar documentación para cada versión.

Para proyectos largos es mejor combinar lo mejor del modelo de cascada y evolutivo: se puede hacer un prototipo global del sistema y posteriormente reimplementarlo con un acercamiento más estructurado. Los subsistemas con requisitos bien definidos y estables se pueden programar utilizando cascada y la interfaz de usuario se puede especificar utilizando un enfoque exploratorio.

Desarrollo formal de sistemas

Este modelo se basa en transformaciones formales de los requisitos hasta llegar a un programa ejecutable.

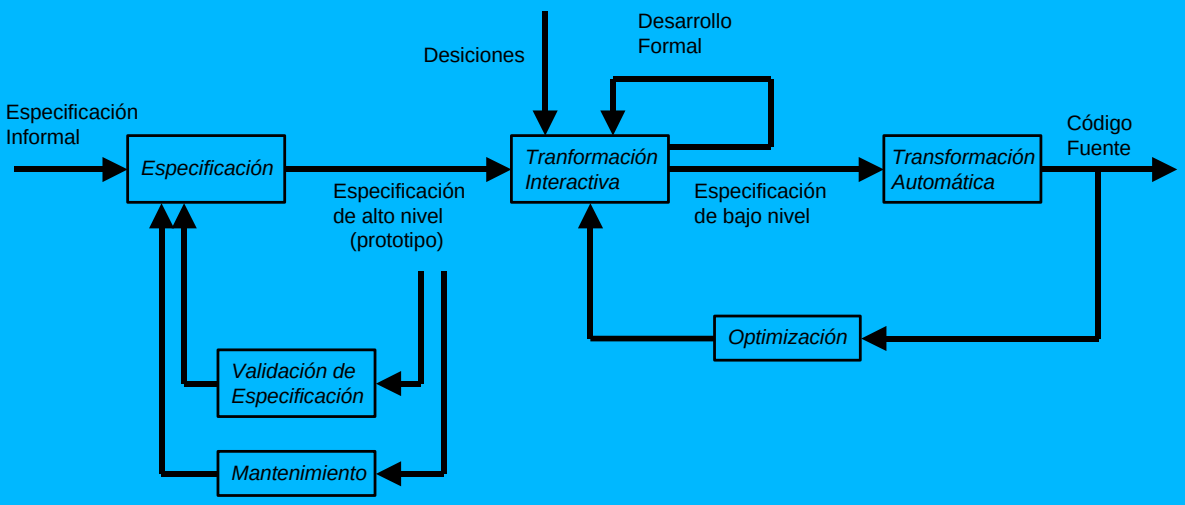

Figura 7: Paradigma de programación automática.

La Figura 7 (obtenida desde [20]) ilustra un paradigma ideal de programación automática. Se distinguen dos fases globales: especificación (incluyendo validación) y transformación. Las características principales de este paradigma son: la especificación es formal y ejecutable constituye el primer prototipo del sistema), la especificación es validada mediante prototipación. Posteriormente, a través de transformaciones formales la especificación se convierte en la implementación del sistema, en el último paso de transformación se obtiene una implementación en un lenguaje de programación determinado. , el mantenimiento se realiza sobre la especificación (no sobre el código fuente), la documentación es generada automáticamente y el mantenimiento es realizado por repetición del proceso (no mediante parches sobre la implementación).

Observaciones sobre el desarrollo formal de sistemas:

- Permite demostrar la corrección del sistema durante el proceso de transformación. Así, las pruebas que verifican la correspondencia con la especificación no son necesarias.
- Es atractivo sobre todo para sistemas donde hay requisitos de seguridad y confiabilidad importantes.
- Requiere desarrolladores especializados y experimentados en este proceso para llevarse a cabo.

Desarrollo basado en reutilización

Como su nombre lo indica, es un modelo fuertemente orientado a la reutilización. Este modelo consta de 4 fases ilustradas en la Figura 9. A continuación se describe cada fase:

1. Análisis de componentes: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.
2. Modificación de requisitos: Se adaptan (en lo posible) los requisitos para concordar con los componentes de la etapa anterior. Si no se puede realizar modificaciones en los requisitos, hay que seguir buscando componentes más adecuados (fase 1).
3. Diseño del sistema con reutilización: Se diseña o reutiliza el marco de trabajo para el sistema. Se debe tener en cuenta los componentes localizados en la fase 2 para diseñar o determinar este marco.
4. Desarrollo e integración: El software que no puede comprarse, se desarrolla. Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

Las ventajas de este modelo son:

- Disminuye el costo y esfuerzo de desarrollo.
- Reduce el tiempo de entrega.
- Disminuye los riesgos durante el desarrollo.

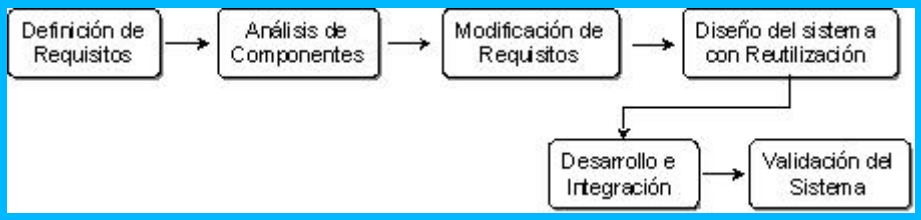

Figura 8: Desarrollo basado en reutilización de componentes

Desventajas de este modelo:

- Los “compromisos” en los requisitos son inevitables, por lo cual puede que el software no cumpla las expectativas del cliente.
- Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

Procesos iterativos

A continuación se expondrán dos enfoques híbridos, especialmente diseñados para el soporte de las iteraciones:

- Desarrollo Incremental.
- Desarrollo en Espiral.

Desarrollo incremental

Mills [9] sugirió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema (ver Figura 10). Es una combinación del Modelo de Cascada y Modelo Evolutivo.

Reduce el rehacer trabajo durante el proceso de desarrollo y da oportunidad para retrasar las decisiones hasta tener experiencia en el sistema.

Durante el desarrollo de cada incremento se puede utilizar el modelo de cascada o evolutivo, dependiendo del conocimiento que se tenga sobre los requisitos a implementar. Si se tiene un buen conocimiento, se puede optar por cascada, si es dudoso, evolutivo.

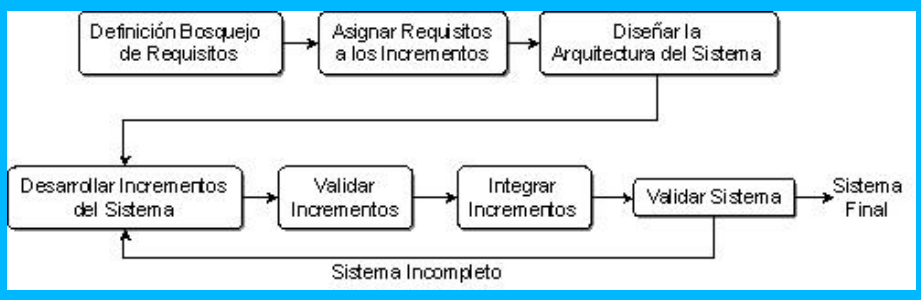

Figura 9: Modelo de desarrollo iterativo incremental.

Entre las ventajas del modelo incremental se encuentran:

- Los clientes no esperan hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento.
- Los clientes pueden aclarar los requisitos que no tengan claros conforme ven las entregas del sistema.
- Se disminuye el riesgo de fracaso de todo el proyecto, ya que se puede distribuir en cada incremento.
- Las partes más importantes del sistema son entregadas primero, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.

Algunas de las desventajas identificadas para este modelo son:

- Cada incremento debe ser pequeño para limitar el riesgo (menos de 20.000 líneas).
- Cada incremento debe aumentar la funcionalidad.
- Es difícil establecer las correspondencias de los requisitos contra los incrementos.
- Es difícil detectar las unidades o servicios genéricos para todo el sistema.
-

Desarrollo en espiral

El modelo de desarrollo en espiral (ver Figura 11) es actualmente uno de los más conocidos y fue propuesto por Boehm [7]. El ciclo de desarrollo se representa como una espiral, en lugar de una serie de actividades sucesivas con retrospectiva de una actividad a otra.

Cada ciclo de desarrollo se divide en cuatro fases:

1. Definición de objetivos: Se definen los objetivos. Se definen las restricciones del proceso y del producto. Se realiza un diseño detallado del plan administrativo. Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de estos.
2. Evaluación y reducción de riesgos: Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir el riesgo de requisitos dudosos. Se llevan a cabo los pasos para reducir los riesgos.
3. Desarrollo y validación: Se escoge el modelo de desarrollo después de la evaluación del riesgo. El modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) depende del riesgo identificado para esa fase.
4. Planificación: Se determina si continuar con otro ciclo. Se planea la siguiente fase del proyecto.

Este modelo a diferencia de los otros toma en consideración explícitamente el riesgo, esta es una actividad importante en la administración del proyecto.

El ciclo de vida inicia con la definición de los objetivos. De acuerdo a las restricciones se determinan distintas alternativas. Se identifican los riesgos al sopesar los objetivos contra las alternativas. Se evalúan los riesgos con actividades como análisis detallado, simulación, prototipos, etc. Se desarrolla un poco el sistema. Se planifica la siguiente fase.

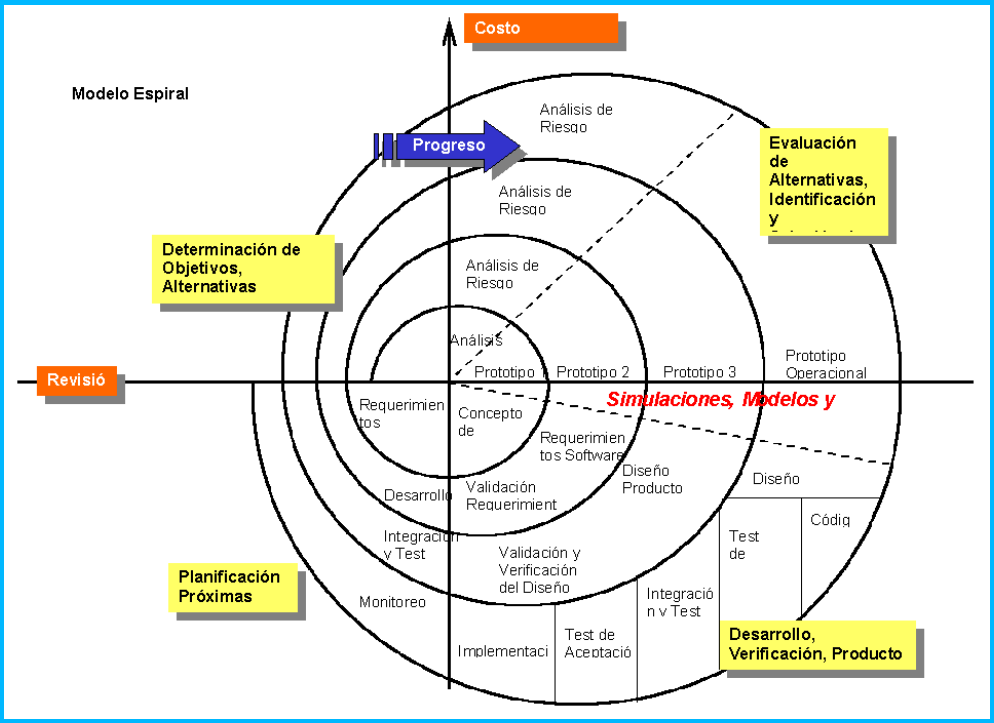

Figura 10: Modelo de desarrollo en Espiral

¿Cuál es el modelo de proceso más adecuado?

Cada proyecto de software requiere de una forma de particular de abordar el problema. Las propuestas comerciales y académicas actuales promueven procesos iterativos, donde en cada iteración puede utilizarse uno u otro modelo de proceso, considerando un conjunto de criterios (Por ejemplo: grado de definición de requisitos, tamaño del proyecto, riesgos identificados, entre otros).

En la Tabla 1 se expone un cuadro comparativo de acuerdo con algunos criterios básicos para la selección de un modelo de proceso [10], la medida utilizada indica el nivel de efectividad del modelo de proceso de acuerdo al criterio (Por ejemplo: El modelo Cascada responde con un nivel de efectividad Bajo cuando los Requisitos y arquitectura no están predefinidos):

Modelo de proceso	Funciona con requisitos y arquitectura no predefinidos	Produce software altamente fiable	Gestión de riesgos	Permite correcciones sobre la marcha	Visión del progreso por el Cliente y el Jefe del proyecto
Codificar y corregir	Bajo	Bajo	Bajo	Alto	Medio
Cascada	Bajo	Alto	Bajo	Bajo	Bajo
Evolutivo exploratorio	Medio o Alto	Medio o Alto	Medio	Medio o Alto	Medio o Alto
Evolutivo prototipado	Alto	Medio	Medio	Alto	Alto
Desarrollo formal de sistemas	Bajo	Alto	Bajo a Medio	Bajo	Bajo
Desarrollo orientado a reutilización	Medio	Bajo a Alto	Bajo a Medio	Alto	Alto
Incremental	Bajo	Alto	Medio	Bajo	Bajo
Espiral	Alto	Alto	Alto	Medio	Medio

Tabla 1: Comparación entre modelos de proceso de software.

Metodologías para desarrollo de software

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 artefactos, 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 artefactos 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 cada una de estas categorías de metodologías.

Metodologías estructuradas

Los métodos estructurados comenzaron a desarrollarse a fines de los 70’s con la Programación Estructurada, luego 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[4] (Francia), MÉTRICA[5] (España), SSADM[6] (Reino Unido). Ejemplos de propuestas de métodos estructurados en el ámbito académico: Gane & Sarson[7], Ward & Mellor[8], Yourdon & DeMarco[9] e Information Engineering[10].

Metodologías orientadas a objetos

Su historia va unida a la evolución de los lenguajes de programación orientada a objeto, los más representativos: a fines de los 60’s SIMULA, a fines de los 70’s Smalltalk-80, la primera versión de C++ por Bjarne Stroustrup en 1981 y actualmente Java[11] o C# de Microsoft. A fines de los 80’s comenzaron a consolidarse algunos métodos Orientadas a Objeto.

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)[12], 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)[13], OPEN[14], MÉTRICA (que también soporta la notación estructurada).

[4] http://perso.club-internet.fr/brouardf/SGBDRmerise.htm (7.5.2002)
[5] http://www.map.es/csi/metrica3/ (7.5.2003)
[6] http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html (7.5.2003)
[7] http://portal.newman.wa.edu.au/technology/12infsys/html/dfdnotes.doc (29.8.2003)
[8] http://www.yourdon.com/books/coolbooks/notes/wardmellor.html (29.8.2003)
[9] http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?Yourdon%2FDemarco (29.8.2003)
[10] http://gantthead.com/Gantthead/process/processMain/1,1289,2-12009-2,00.html (29.8.2003)
[11] http://java.sun.com/ (7.5.2003)
[12] http://www.uml.org/ (7.5.2003)
[13] http://www.rational.com/products/rup/index.jsp (7.5.2003)
[14] http://www.open.org.au/ (17.9.2003)

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. Aunque en el caso particular de RUP, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del proyecto (mediante su configuración previa a aplicarse), realizando una configuración adecuada, podría considerarse Ágil.

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) [11].

Entre las metodologías ágiles identificadas en [11]:

- Extreme Programming [6].
- Scrum ([12], [13]).
- Familia de Metodologías Crystal [14].
- Feature Driven Development [15].
- Proceso Unificado Rational, una configuración ágil ([16]).
- Dynamic Systems Development Method [17].
- Adaptive Software Development [18].
- Open Source Software Development [19].

Referencias

[1] Pressman, R, Ingeniería del Software: Un enfoque práctico, McGraw Hill 1997.
[2] Naur P., Randell B., Software Engineering: A Report on a Conference Sponsored by the NATO Scienc, 1969.
[3] Jacaboson, I., Booch, G., Rumbaugh J., El Proceso Unificado de Desarrollo de Software, Addison Wesley 2000.
[4] Sommerville, I., Ingeniería de Software, Pearson Educación, 2002.
[5] Letelier, P., Proyecto Docente e Investigador, DSIC, 2003.
[6] Beck, K., Una explicación de la Programación Extrema. Aceptar el cambio, Pearson Educación, 2000.
[7] Boehm, B. W., A Spiral Model of Software Develpment and Enhancement, IEEE Computer ,1988.
[8] Royce, W., Managing the developmento of large software systems: concepts and technique, IEEE Westcon, 1970.
[9] Mills, H., O´Neill, D., The Management of Software Engineering, IBM Systems, 1980.
[10] Laboratorio Ing. Soft., Ingeniería de software 2, Departamento de Informática, 2002.
[11] Abrahamsson, P., Salo, O., Ronkainen, J., Agile Software Development Methods. Review and Analysis, VTT, 2002.
[12] Schwaber, K., Scrum Development Process. Workshop on Business Object Design and Implementation, OOPSLA´95, 1995.
[13] Schwaber, K., Beedle, M., Agile Software Development With Scrum, Prentice Hall, 2002.
[14] Cockburn, A., Agile Software Development, Addison Wesley, 2002.
[15] Palmer, S. R., Felsing, J. M., A Practical Guide to Feature Driven Development, Prentice Hall, 2002.
[16] Kruchten, P., A Rational Development Process, Crosstalk, 1996.
[17] Stapleton, J., Dynamic Systems Development Method - The Method in Practice, Addison Wesley, 1997.
[18] Highsmith, J., Adaptive Software Development: A Collaborative Approach, Dorset House, 2000.
[19] O´Reilly, T., Lessons from Open Source Software Development, ACM, 1999.
[20] Balzer R. *A 15 Year Perspective on Automatic Programming*. IEEE Transactions on Software Engineering, vol.11, núm.11, páginas 1257-1268, Noviembre 1985.