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.

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) (comentadas en la Sección 4.5) pueden ayudar a algunas actividades del proceso. Sin embargo, no existe posibilidad alguna, al menos en los próximos años, de una automatización mayor en el diseño creativo del software realizado por los ingenieros relacionados con el proceso del software.

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.*** Se debe definir la funcionalidad del software y las restricciones en su operación.
2. ***Diseño e implementación del software.*** Se debe producir software que cumpla su especificación.
3. ***Validación del software.*** Se debe validar el software para asegurar que hace lo que el cliente desea.
4. ***Evolución del software.*** El software debe evolucionar para cubrir las necesidades cambiantes del cliente.

En este capítulo se tratan brevemente estas actividades y se analizan con más detalle en partes posteriores del libro.

Aunque no existe un proceso del software «ideal», en las organizaciones existen enfoques para mejorarlos. Los procesos pueden incluir técnicas anticuadas o no aprovecharse de las mejores prácticas en la ingeniería del software industrial. De hecho, muchas organizaciones aún no aprovechan los métodos de la ingeniería del software en el desarrollo de su software.

(Los procesos del software se pueden mejorar por la estandarización del proceso donde la diversidad de los procesos del software en una organización sea reducida. 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. En el Capítulo 28 se trata con más detalle la mejora del proceso del software.

4.1 Modelos del proceso del software

Como se explicó en el Capítulo 1, un modelo del proceso del software es una representación abstracta de un proceso del software. Cada modelo de proceso representa un proceso

desde una perspectiva particular, y así proporciona sólo información parcial sobre ese proceso. En esta sección, se introducen varios modelos de proceso muy generales (algunas veces llamados ***paradigmas de proceso)*** y se presentan desde una perspectiva arquitectónica. Esto es, vemos el marco de trabajo del proceso, pero no los detalles de actividades específicas.

Estos modelos generales no son descripciones definitivas de los procesos del software. Más bien, 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.

Los modelos de procesos que se incluyen en este capítulo son:

1. ***El modelo en cascada.*** Considera las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución, y los representa como fases separadas del proceso, tales como la especificación de requerimientos, el diseño del software, la implementación, las pruebas, etcétera.
2. ***Desarrollo evolutivo.*** Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas. Este se refina basándose en las peticiones del cliente para producir un sistema que satisfaga sus necesidades.
3. ***Ingeniería del software basada en componentes.*** Este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en integrar estos componentes en el sistema más que en desarrollarlos desde cero.

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. De hecho, el Proceso Unificado de Rational que se trata en la Sección 4.4 combina elementos de todos estos modelos. Los subsistemas deniro 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.

Se han propuesto todo tipo de variantes de estos procesos genéricos y pueden ser usados en algunas organizaciones. La variante más importante es probablemente el desarrollo formal de sistemas, donde se crea un modelo formal matemático de un sistema. Este modelo se transforma entonces, usando transformaciones matemáticas que preservan su consistencia, en código ejecutable.

El ejemplo más conocido de un proceso de desarrollo formal es el proceso de sala limpia, el cual fue originalmente desarrollado por IBM (Mills ***etal.,*** 1987; Selby ***etai,*** 1987; Linger. 1994: Prowell ***et al.,*** 1999). En el proceso de sala limpia, cada incremento del software se especifica formalmente, y esta especificación se transforma en una implementación. La corrección del software se demuestra utilizando un enfoque formal. No hay pruebas para defectos en el proceso, y las pruebas del sistema se centran en evaluar su fiabilidad.

Tanto el enfoque de sala limpia como otro enfoque para desarrollo formal basado en el método B (Wordsworth, 1996) son particularmente apropiados para el desarrollo de sistemas que tienen estrictos requerimientos de seguridad, fiabilidad o protección. El enfoque formal simplifica la producción de un caso de seguridad o protección que demuestre a los clientes u organismos de certificación que el sistema realmente cumple los requerimientos de seguridad y protección.

Fuera de estos ámbitos especializados, los procesos basados en transformaciones formales no se utilizan en general. Requieren una pericia especializada y, en realidad, para la mayoría de los sistemas este proceso no ofrece ventajas importantes de coste o calidad sobre otros enfoques para el desarrollo de sistemas.

4.1.1 El modelo en cascada

El primer modelo de proceso de desarrollo de software que se publicó se derivó de procesos de ingeniería de sistemas más generales (Royce, 1970). Este modelo se muestra en la Figura 4.1. Debido a la cascada de una fase a otra, dicho modelo se conoce como modelo en cascada o como ciclo de vida del software. Las principales etapas de este modelo se transforman en actividades fundamentales de desarrollo:

1. ***Análisis y definición de requerimientos.*** Los servicios, restricciones y metas del sistema se definen a partir de las consultas con los usuarios. Entonces, se definen en detalle y sirven como una especificación del sistema.
2. ***Diseño del sistema y del software.*** El proceso de diseño del sistema divide los requerimientos en sistemas hardware o software. Establece una arquitectura completa del sistema. El diseño del software identifica y describe las abstracciones fundamentales del sistema software y sus relaciones.
3. ***¡mplementación y prueba de unidades.*** Durante esta etapa, el diseño del software se lleva a cabo como un conjunto o unidades de programas. La prueba de unidades implica verificar que cada una cumpla su especificación.
4. ***Integración y prueba del sistema.*** Los programas o las unidades individuales de programas se integran y prueban como un sistema completo para asegurar que se cumplan los requerimientos del software. Después de las pruebas, el sistema software se entrega al cliente.
5. ***Funcionamiento y mantenimiento.*** Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida. El sistema se instala y se pone en funcionamiento práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida, mejorar la ¡mplementación de las unidades del sistema y resaltar los servicios del sistema una vez que se descubren nuevos requerimientos.

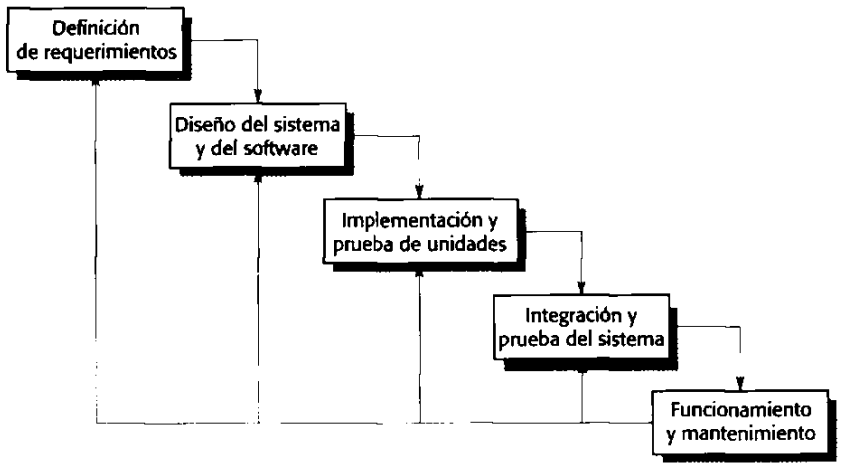


Figura 4.1 El ciclo de vida del software.

En principio, el resultado de cada fase es uno o más documentos aprobados («firmados»). La siguiente fase no debe empezar hasta que la fase previa haya finalizado. En la práctica, estas etapas se superponen y proporcionan información a las otras. Durante el diseño se identifican los problemas con los requerimientos; durante el diseño del código se encuentran problemas, y así sucesivamente. El proceso del software no es un modelo lineal simple, sino que implica una serie de iteraciones de las actividades de desarrollo.

Debido a los costos de producción y aprobación de documentos, las iteraciones son costosas e implican rehacer el trabajo. Por lo tanto, después de un número reducido de iteraciones, es normal congelar partes del desarrollo, como la especificación, y continuar con las siguientes etapas de desarrollo. Los problemas se posponen para su resolución, se pasan por alto o se programan. Este congelamiento prematuro de requerimientos puede implicar que el sistema no haga lo que los usuarios desean. También puede conducir a sistemas mal estructurados debido a que los problemas de diseño se resuelven mediante trucos de implementación.

Durante la fase final del ciclo de vida (funcionamiento y mantenimiento), el software se pone en funcionamiento. Se descubren errores y omisiones en los requerimientos originales del software. Los errores de programación y de diseño emergen y se identifica la necesidad de una nueva funcionalidad. Por tanto, el sistema debe evolucionar para mantenerse útil. Hacer estos cambios (mantenimiento del software) puede implicar repetir etapas previas del proceso.

Las ventajas del modelo en cascada son que la documentación se produce en cada fase y que éste cuadra con otros modelos del proceso de ingeniería. Su principal problema es su inflexibilidad al dividir el proyecto en distintas etapas. Se deben hacer compromisos en las etapas iniciales, lo que hace difícil responder a los cambios en los requerimientos del cliente.

Por lo tanto, el modelo en cascada sólo se debe utilizar cuando los requerimientos se comprendan bien y sea improbable que cambien radicalmente durante el desarrollo del sistema. Sin embargo, el modelo refleja el tipo de modelo de proceso usado en otros proyectos de la ingeniería. Por consiguiente, los procesos del software que se basan en este enfoque se siguen utilizando para el desarrollo de software, particularmente cuando éste es parte de proyectos grandes de ingeniería de sistemas.

4.1.2 Desarrollo evolutivo

El desarrollo evolutivo se basa en la idea de desarrollar una implementación inicial, exponiéndola a los comentarios del usuario y refinándola a través de las diferentes versiones hasla que se desarrolla un sistema adecuado (Figura 4.2). Las actividades de especificación,

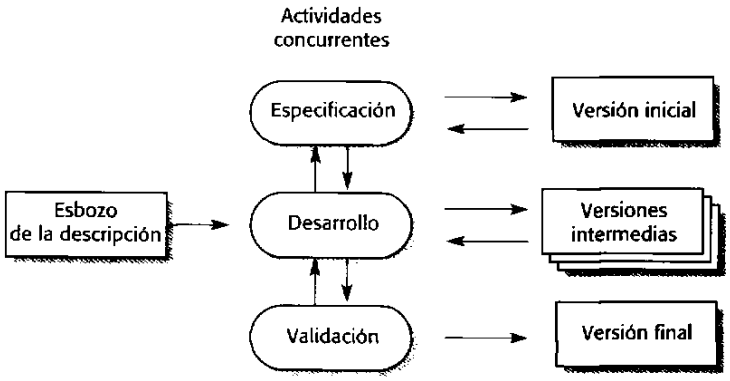


Figura 4.2
Desarrollo evolutivo.

desarrollo y validación se entrelazan en vez de separarse, con una rápida retroalimentación entre éstas.

Existen dos tipos de desarrollo evolutivo:

1. ***Desarrollo exploratorio,*** donde el objetivo del proceso es trabajar con el cliente para explorar sus requerimientos y entregar un sistema final. El desarrollo empieza con las partes del sistema que se comprenden mejor. El sistema evoluciona agregando nuevos atributos propuestos por el cliente.
2. ***Prototipos desechadles,*** donde el objetivo del proceso de desarrollo evolutivo es comprender los requerimientos del cliente y entonces desarrollar una definición mejorada de los requerimientos para el sistema. El prototipo se centra en experimentar con los^ requerimientos del cliente que no se comprenden del todo.

En la producción de sistemas, un enfoque evolutivo para el desarrollo de software suele ser más efectivo que el enfoque en cascada, ya que satisface las necesidades inmediatas de los clientes. La ventaja de un proceso del software que se basa en un enfoque evolutivo es que la especificación se puede desarrollar de forma creciente. Tan pronto como los usuarios desarrollen un mejor entendimiento de su problema, éste se puede reflejar en el sistema software. Sin embargo, desde una perspectiva de ingeniería y de gestión, el enfoque evolutivo tiene dos problemas:

1. ***El proceso no es visible.*** Los administradores tienen que hacer entregas regulares para medir el progreso. Si los sistemas se desarrollan rápidamente, no es rentable producir documentos que reflejen cada versión del sistema.
2. ***,4 menudo los sistemas tienen una estructura deficiente.*** Los cambios continuos tienden a corromper la estructura del software. Incorporar cambios en él se convierte cada vez más en una tarea difícil y costosa.

Para sistemas pequeños y de tamaño medio (hasta 500.000 líneas de código), el enfoque evolutivo de desarrollo es el mejor. Los problemas del desarrollo evolutivo se hacen particularmente agudos para sistemas grandes y complejos con un periodo de vida largo, donde diferentes equipos desarrollan distintas partes del sistema. Es difícil establecer una arquitectura del sistema estable usando este enfoque, el cual hace difícil integrar las contribuciones de los equipos.

Para sistemas grandes, se recomienda un proceso mixto que incorpore las mejores características del modelo en cascada y del desarrollo evolutivo. Esto puede implicar desarrollar un prototipo desechable utilizando un enfoque evolutivo para resolver incertidumbres en la especificación del sistema. Puede entonces reimplementarse utilizando un enfoque más estructurado. Las partes del sistema bien comprendidas se pueden especificar y desarrollar utilizando un proceso basado en el modelo en cascada. Las otras partes del sistema, como la interfaz de usuario, que son difíciles de especificar por adelantado, se deben desarrollar siempre utilizando un enfoque de programación exploratoria.

Los procesos del desarrollo evolutivo y el proceso de apoyo se estudian con más detalle en el Capítulo 17, junto con la construcción de prototipos de sistemas y el desarrollo rápido de aplicaciones. El desarrollo evolutivo también se incorpora en el Proceso Unificado de Rational que se trata más tarde en este capítulo.

4.1.3 Ingeniería del software basada en componentes

En la mayoría de los proyectos de software existe algo de reutilización de software. Por lo general, esto sucede informalmente cuando las personas que trabajan en el proyecto conocen di-

senos o código similares al requerido. Los buscan, los modifican según lo creen necesario y los incorporan en el sistema. En el enfoque evolutivo, descrito en la Sección 4.1.2, la reutilización es a menudo indispensable para el desarrollo rápido de sistemas.

Esta reutilización informal es independiente del proceso de desarrollo que se utilice. Sin embargo, en los últimos años, ha surgido un enfoque de desarrollo de software denominado ingeniería del software basada en componentes (CBSE) que se basa en la reutilización, el cual se está utilizando de forma amplia. Se introduce brevemente este enfoque aquí, pero se estudia con más detalle en el Capítulo 19.

Este enfoque basado en la reutilización se compone de una gran base de componentes software reutilizables y de algunos marcos de trabajo de integración para éstos. Algunas veces estos componentes son sistemas por sí mismos (COTS o sistemas comerciales) que se pueden utilizar para proporcionar una funcionalidad específica, como dar formato al texto o efectuar cálculos numéricos. En la Figura 4.3 se muestra el modelo del proceso genérico para la CBSE.

Aunque la etapa de especificación de requerimientos y la de validación son comparables con otros procesos, las etapas intermedias en el proceso orientado a la reutilización son diferentes. Estas etapas son:

1. ***Análisis de componentes.*** Dada la especificación de requerimientos, se buscan los componentes para implementar esta especificación. Por lo general, no existe una concordancia exacta y los componentes que se utilizan sólo proporcionan parte de la funcionalidad requerida.
2. ***Modificación de requerimientos.*** En esta etapa, los requerimientos se analizan utilizando información acerca de los componentes que se han descubierto. Entonces, estos componentes se modifican para reflejar los componentes disponibles. Si las modificaciones no son posibles, la actividad de análisis de componentes se puede llevar a cabo nuevamente para buscar soluciones alternativas.
3. ***Diseño del sistema con reutilización.*** En esta fase se diseña o se reutiliza un marco de trabajo para el sistema. Los diseñadores tienen en cuenta los componentes que se reutilizan y organizan el marco de trabajo para que los satisfaga. Si los componentes reutilizables no están disponibles, se puede tener que diseñar nuevo software.
4. ***Desarrollo e integración.*** Para crear el sistema, el software que no se puede adquirir externamente se desarrolla, y los componentes y los sistemas COTS se integran. En este modelo, la integración de sistemas es parte del proceso de desarrollo, más que una actividad separada.

La ingeniería del software basada en componentes tiene la ventaja obvia de reducir la cantidad de software a desarrollarse y así reduce los costos y los riesgos. Por lo general, también permite una entrega más rápida del software. Sin embargo, los compromisos en los requerimientos son inevitables, y esto puede dar lugar a un sistema que no cumpla las necesidades

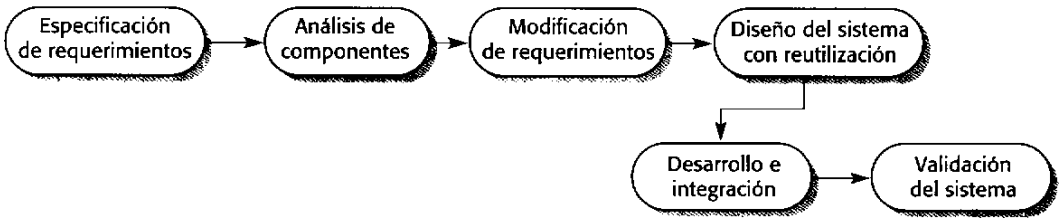


Figura 4.3 Ingeniería del software basada en componentes.

reales de los usuarios. Más aún: si las nuevas versiones de los componentes reutitizables no están bajo el control de la organización que los utiliza, se pierde parte del control sobre la evolución del sistema.

La CBSE tiene mucho en común con un enfoque que está surgiendo para el desarrollo de sistemas que se basa en la integración de servicios web de una serie de proveedores. En el Capítulo 12 se describe este enfoque de desarrollo de servicio céntrico.

4.2 Iteración de procesos

Los cambios son inevitables en todos los proyectos de software grandes. Los requerimientos del sistema cambian cuando el negocio que procura el sistema responde a las presiones externas. Las prioridades de gestión cambian. Cuando se dispone de nuevas tecnologías, cambian los diseños y la implementación. Esto significa que el proceso del software no es un proceso único; más bien, las actividades del proceso se repiten regularmente conforme el sistema se rehace en respuesta a peticiones de cambios.

El desarrollo iterativo es tan fundamental para el software que se le dedica un capítulo completo del libro más adelante (Capítulo 17). En esta sección, se introduce el tema describiendo dos modelos de procesos que han sido diseñados explícitamente para apoyar la iteración de procesos:

1. ***Entrega incrementa!.*** La especificación, el diseño y la implementación del software se dividen en una serie de incrementos, los cuales se desarrollan por turnos;
2. ***Desarrollo en espira!.*** El desarrollo del sistema gira en espiral hacia fuera, empezando con un esbozo inicial y terminando con el desarrollo final del mismo.

La esencia de los procesos iterativos es que la especificación se desarrolla junto con el software. Sin embargo, esto crea conflictos con el modelo de obtención de muchas organizaciones donde la especificación completa del sistema es parte del contrato de desarrollo del mismo. En el enfoque incremental, no existe una especificación completa del sistema hasta que el incremento final se especifica. Esto requiere un nuevo tipo de contrato, que a los clientes grandes como las agencias del gobierno les puede ser difícil de incorporar.

4.2.1 Entrega Incremental

El modelo de desarrollo en cascada requiere que los clientes de un sistema cumplan un conjunto de requerimientos antes de que se inicie el diseño y que el diseñador cumpla estrategias particulares de diseño antes de la implementación. Los cambios de requerimientos implican rehacer el trabajo de captura de éstos, de diseño e implementación. Sin embargo, la separación en el diseño y la implementación deben dar lugar a sistemas bien documentados susceptibles de cambio. En contraste, un enfoque de desarrollo evolutivo permite que los requerimientos y las decisiones de diseño se retrasen, pero también origina un software que puede estar débilmente estructurado y difícil de comprender y mantener.

La entrega incremental (Figura 4.4) es un enfoque intermedio que combina las ventajas de estos modelos. En un proceso de desarrollo incremental. los clientes identifican, a grandes rasgos, los servicios que proporcionará el sistema. Identifican qué servicios son más importantes y cuáles menos. Entonces, se definen varios incrementos en donde cada uno proporciona un subconjunto de la funcionalidad del sistema. La asignación de servicios a los

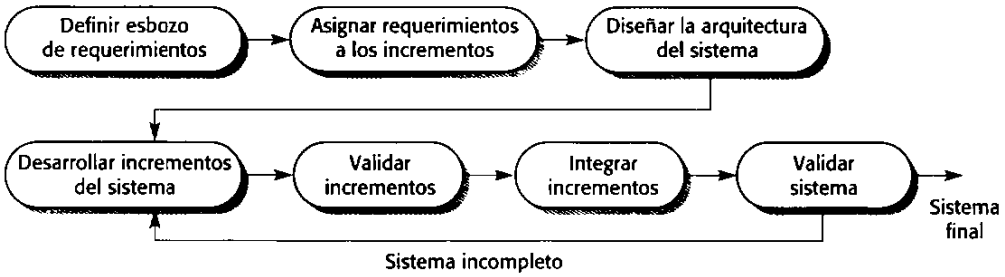


Figura 4.4 Entrega incremental.

incrementos depende de la prioridad del servicio con los servicios de prioridad más alta entregados primero.

Una vez que los incrementos del sistema se han identificado, los requerimientos para los servicios que se van a entregar en el primer incremento se definen en detalle, y éste se desarrolla. Durante el desarrollo, se puede llevar a cabo un análisis adicional de requerimientos para los requerimientos posteriores, pero no se aceptan cambios en los requerimientos para el incremento actual.

Una vez que un incremento se completa y entrega, los clientes pueden ponerlo en servicio. Esto significa que tienen una entrega temprana de parte de la funcionalidad del sistema. Pueden experimentar con el sistema, lo cual les ayuda a clarificar sus requerimientos para los incrementos posteriores y para las últimas versiones del incremento actual. Tan pronto como se completan los nuevos incrementos, se integran en los existentes de tal forma que la funcionalidad del sistema mejora con cada incremento entregado. Los servicios comunes se pueden implementar al inicio del proceso o de forma incremental tan pronto como sean requeridos por un incremento.

Este proceso de desarrollo incremental tiene varias ventajas:

1. Los clientes no tienen que esperar hasta que el sistema completo se entregue para sacar provecho de él. El primer incremento satisface los requerimientos más críticos de tal forma que pueden utilizar el software inmediatamente.
2. Los clientes pueden utilizar los incrementos iniciales como prototipos y obtener experiencia sobre los requerimientos de los incrementos posteriores del sistema.
3. Existe un bajo riesgo de un fallo total del proyecto. Aunque se pueden encontrar problemas en algunos incrementos, lo normal es que el sistema se entregue de forma satisfactoria al cliente.
4. Puesto que los servicios de más alta prioridad se entregan primero, y los incrementos posteriores se integran en ellos, es inevitable que los servicios más importantes del sistema sean a los que se les hagan más pruebas. Esto significa que es menos probable que los clientes encuentren fallos de funcionamiento del software en las partes más importantes del sistema.

Sin embargo, existen algunos problemas en el desarrollo incremental. Los incrementos deben ser relativamente pequeños (no más de 20.000 líneas de código) y cada uno debe entregar alguna funcionalidad del sistema. Puede ser difícil adaptar los requerimientos del cliente a incrementos de tamaño apropiado. Más aún, muchos de los sistemas requieren un conjunto de recursos que se utilizan en diferentes partes del sistema. Puesto que los requerimientos no se definen en detalle hasta que un incremento se implementa, puede ser difícil identificar los recursos comunes que requieren todos los incrementos.

Se ha desarrollado una variante de este enfoque incremental denominada ***programación extrema*** (Beck, 2000). Ésta se basa en el desarrollo y la entrega de incrementos de funcionalidad muy pequeños, en la participación del cliente en el proceso, en la mejora constante del código y en la programación por parejas. En el Capítulo 17 se estudia la programación por parejas y otros llamados métodos ágiles.

4.2.2 Desarrollo en espiral

El modelo en espiral del proceso del software (Figura 4.5) fue originalmente propuesto por Boehm (Boehm, 1988). Más que representar el proceso del software como una secuencia de actividades con retrospectiva de una actividad a otra, se representa como una espiral. Cada ciclo en la espiral representa una fase del proceso del software. Así. el ciclo más interno podría referirse a la viabilidad del sistema, el siguiente ciclo a la definición de requerimientos, el siguiente ciclo al diseño del sistema, y así sucesivamente.

Cada ciclo de la espiral se divide en cuatro sectores:

1. ***Definición de objetivos.*** Para esta fase del proyecto se definen los objetivos específicos. Se identifican las restricciones del proceso y el producto, y se traza un plan detallado de gestión. Se identifican los riesgos del proyecto. Etependiendo de estos riesgos, se planean estrategias alternativas.
2. ***Evaluación y reducción de riesgos.*** Se lleva a cabo un análisis detallado para cada uno de los riesgos del proyecto identificados. Se definen los pasos para reducir dichos riesgo. Por ejemplo, si existe el riesgo de tener requerimientos inapropiados, se puede desarrollar un prototipo del sistema.

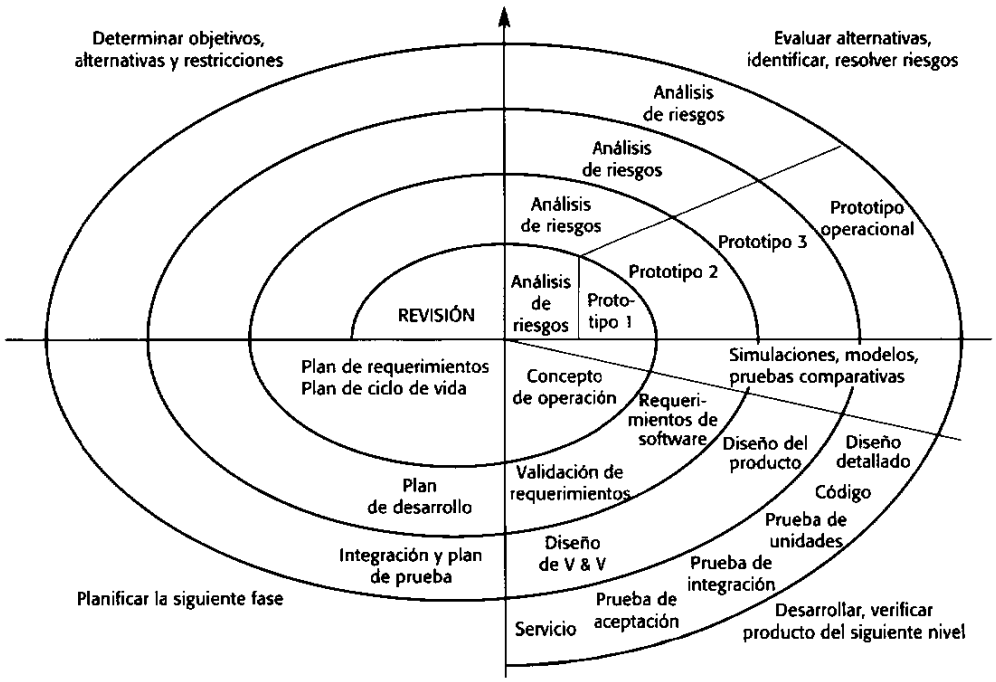


Figura 4.5 Modelo en espiral de Boehm para el proceso del software (©IEEE, 1988).

3. ***Desarrollo y validación.*** Después de la evaluación de riesgos, se elige un modelo para el desarrollo del sistema. Por ejemplo, si los riesgos en la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos. Si los riesgos de seguridad son la principal consideración, un desarrollo basado en transformaciones formales podría ser el más apropiado, y así sucesivamente. El modelo en cascada puede ser el más apropiado para el desarrollo si el mayor riesgo identificado es la integración de los subsistemas.
4. ***Planificación.*** El proyecto se revisa y se toma la decisión de si se debe continuar con un ciclo posterior de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

La diferencia principal entre el modelo en espiral y los otros modelos del proceso del software es la consideración explícita del riesgo en el modelo en espiral. Informalmente, el riesgo significa sencillamente algo que puede ir mal. Por ejemplo, si la intención es utilizar un nuevo lenguaje de programación, un riesgo es que los compiladores disponibles sean poco fiables o que no produzcan código objeto suficientemente eficiente. Los riesgos originan problemas en el proyecto, como los de confección de agendas y excesos en los costos; por lo tanto, la disminución de riesgos es una actividad muy importante en la gestión del proyecto. La gestión de los riesgos, una parte fundamental en la gestión de proyectos, se trata en el Capítulo 5.

Un ciclo de la espiral empieza con la elaboración de objetivos, como el rendimiento y la funcionalidad. Entonces se enumeran formas alternativas de alcanzar estos objetivos y las restricciones impuestas en cada una de ellas. Cada alternativa se evalúa contra cada objetivo y se identifican las fuentes de riesgo del proyecto. El siguiente paso es resolver estos riesgos mediante actividades de recopilación de información como la de detallar más el análisis, la construcción de prototipos y la simulación. Una vez que se han evaluado los riesgos, se lleva a cabo cierto desarrol lo. seguido de una actividad de planificación para la siguiente fase del proceso.

4.3 Actividades del proceso

Las cuatro actividades básicas del proceso de especificación, desarrollo, validación y evolución se organizan de forma distinta en diferentes procesos del desarrollo. En el enfoque en cascada, están organizadas en secuencia, mientras que en el desarrollo evolutivo se entrelazan. Cómo se llevan a cabo estas actividades depende del tipo de software, de las personas y de la estructura organización al implicadas. No hay una forma correcta o incorrecta de organizarestas actividades, y el objetivo de esta sección es simplemente proporcionar una introducción de cómo se pueden organizar.

4.3.1 Especificación del software

La especificación del software o ingeniería de requerimientos es el proceso de comprensión y definición de qué servicios se requieren del sistema y de identificación de las restricciones de funcionamiento y desarrollo del mismo. La ingeniería de requerimientos es una etapa particularmente crítica en el proceso del software ya que los errores en esta etapa originan inevitablemente problemas posteriores en el diseño e implementación del sistema.

En la Figura 4.6 se muestra el proceso de ingeniería de requerimientos. Éste conduce a la producción de un documento de requerimientos, que es la especificación del sistema. Normalmente en este documento los requerimientos se presentan en dos niveles de detalle. Los usuarios finales y los clientes necesitan una declaración de alto nivel de los requerimientos, mientras que los desarrolladores del sistema necesitan una especificación más detallada de éste.

Existen cuatro fases principales en el proceso de ingeniería de requerimientos:

1. ***Estudio de viabilidad.*** Se estima si las necesidades del usuario se pueden satisfacer con las tecnologías actuales de software y hardware. El estudio analiza si el sistema propuesto será rentable desde un punto de vista de negocios y si se puede desarrollar dentro de las restricciones de presupuesto existentes. Este estudio debe ser relativamente económico y rápido de elaborar. El resultado debe informar si se va a continuar con un análisis más detallado.
2. ***Obtención y análisis de requerimientos.*** Es el proceso de obtener los requerimientos del sistema por medio de la observación de los sistemas existentes, discusiones con los usuarios potenciales y proveedores, el análisis de tareas, etcétera. Esto puede implicar el desarrollo de uno o más modelos y prototipos del sistema que ayudan al analista a comprender el sistema a especificar.
3. ***Especificación de requerimientos.*** Es la actividad de traducir la información recopilada durante la actividad de análisis en un documento que define un conjunto de requerimientos. En este documento se pueden incluir dos tipos de requerimientos: los ***requerimientos del usuario,*** que son declaraciones abstractas de los requerimientos del cliente y del usuario final del sistema, y los ***requerimientos del sistema,*** que son una descripción más detallada de la funcionalidad a proporcionar.
4. ***Validación de requerimientos.*** Esta actividad comprueba la veracidad, consistencia y completitud de los requerimientos. Durante este proceso, inevitablemente se descubren errores en el documento de requerimientos. Se debe modificar entonces para corregir estos problemas.

Por supuesto, las actividades en el proceso de requerimientos no se llevan a cabo de forma estrictamente secuencial. El análisis de requerimientos continúa durante la definición y especificación, y a lo largo del proceso surgen nuevos requerimientos. Por lo tanto, las activida-

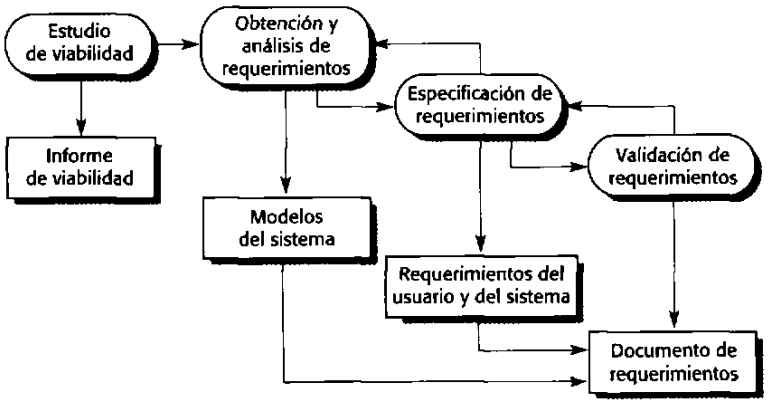


Figura 4.6 El proceso de la ingeniería de requerimientos.

des de análisis, definición y especificación se entrelazan. En los métodos ágiles como la programación extrema, los requerimientos de desarrollan de forma incremental conforme a las prioridades del usuario, y la obtención de requerimientos viene de los usuarios que forman parte del equipo de desarrollo.

4.3.2 Diseño e Implementación del software

La etapa de implementación del desarrollo de software es el proceso de convertir una especificación del sistema en un sistema ejecutable. Siempre implica los procesos de diseño y programación de software, pero, si se utiliza un enfoque evolutivo de desarrollo, también puede implicar un refinamiento de la especificación del software.

Un diseño de software es una descripción de la estructura del software que se va a implementar, los datos que son parte del sistema, las interfaces entre los componentes del sistema y, algunas veces, los algoritmos utilizados. Los diseñadores no llegan inmediatamente a un diseño detallado, sino que lo desarrollan de manera iterativa a través de diversas versiones. El proceso de diseño conlleva agregar formalidad y detalle durante el desarrollo del diseño, y regresar a los diseños anteriores para corregirlos.

El proceso de diseño puede implicar el desarrollo de varios modelos del sistema con diferentes niveles de abstracción. Mientras se descompone un diseño, se descubren errores y omisiones de las etapas previas. Esta retroalimentación permite mejorar los modelos de diseño previos. La Figura 4.7 es un modelo de este proceso que muestra las descripciones de diseño que pueden producirse en varias etapas del diseño. Este diagrama sugiere que las etapas son secuenciales. En realidad, las actividades del proceso de diseño se entrelazan. La retroalimentación entre etapas y la consecuente repetición del trabajo es inevitable en todos los procesos de diseño.

Una especificación para la siguiente etapa es la salida de cada actividad de diseño. Esta especificación puede ser abstracta y formal, realizada para clarificar los requerimientos, o puede ser una especificación para determinar qué parte del sistema se va a construir. Durante todo el proceso de diseño, se detalla cada vez más esta especificación. El resultado final del proceso son especificaciones precisas de los algoritmos y estructuras de datos a implementarse.

Las actividades específicas del proceso de diseño son:

1. ***Diseño arquitectónico.*** Los subsistemas que forman el sistema y sus relaciones se identifican y documentan. En los Capítulos 11, 12 y 13 se trata este importante tema.

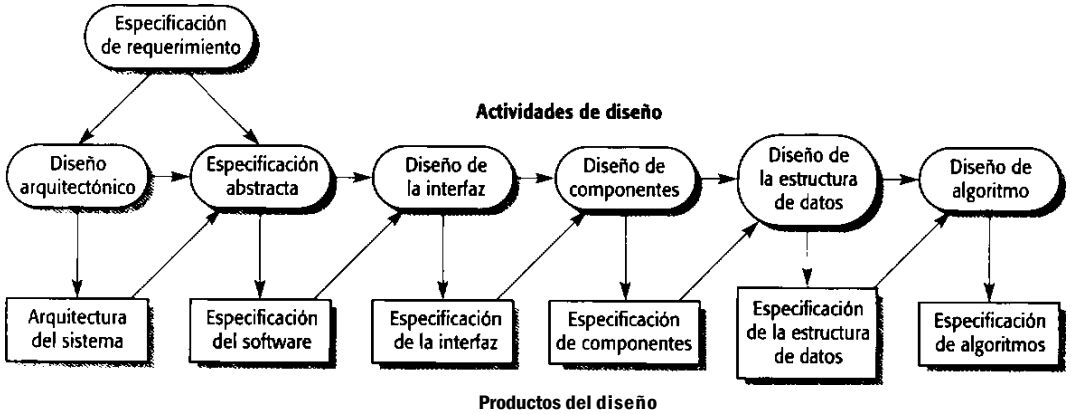


Figura 4.7 Un modelo general del proceso de diseño.

2. ***Especificación abstracta.*** Para cada subsistema se produce una especificación abstracta de sus servicios y las restricciones bajo las cuales debe funcionar.
3. ***Diseño de la interfaz.*** Para cada subsistema se diseña y documenta su interfaz con otros subsistemas. Esta especificación de la interfaz debe ser inequívoca ya que permite que el subsistema se utilice sin conocimiento de su funcionamiento. En esta etapa pueden utilizarse los métodos de especificación formal que se describen en el Capítulo 10.
4. ***Diseño de componentes.*** Se asignan servicios a los componentes y se diseñan sus interfaces.
5. ***Diseño de la estructura de datos.*** Se diseña en detalle y especifica la estructura de datos utilizada en la implementación del sistema.
6. ***Diseño de algoritmos.*** Se diseñan en detalle y especifican los algoritmos utilizados para proporcionar los servicios.

Éste es un modelo general del proceso de diseño, y los procesos reales y prácticos pueden adaptarlo de diversas maneras. He aquí algunas adaptaciones posibles:

1. Las dos últimas etapas —diseño de la estructura de datos y de algoritmos— se pueden retrasar hasta la etapa de implementación.
2. Si se utiliza un enfoque exploratorio de diseño, las interfaces del sistema se pueden diseñar después de que se especifiquen las estructuras de datos.
3. Se puede omitir la etapa de especificación abstracta, aunque normalmente es una parte fundamental del diseño de sistemas críticos.

Cada vez más, cuando se utilizan métodos ágiles de desarrollo ***(véase*** el Capítulo 17), las salidas del proceso de diseño no serán documentos de especificación separados, sino que estarán representadas en el código del programa. Una vez diseñada la arquitectura de un sistema, las etapas posteriores del diseño son increméntales. Cada incremento se representa como código del programa en vez de como un modelo de diseño.

Un enfoque opuesto es dado por los ***métodos estructurados*** que se basan en la producción de modelos gráficos del sistema ***(véase*** el Capítulo 8) y, en muchos casos, código automáticamente generado desde estos modelos. Los métodos estructurados se inventaron en los años 70 en apoyo del diseño orientado a funciones (Constantine y Yourdon, 1979; Gane y Sarson, 1979). Se propusieron varios modelos competentes de apoyo al diseño orientado a objetos (Robinson, 1992; Booch, 1994) y éstos se unificaron en los años 90 para crear el Lenguaje Unificado de Modelado (UML) y el proceso unificado de diseño asociado (Rumbaugh ***etal.,*** 1991; Booch ***etal.,*** 1999; Rumbaugh ***etal.,*** 1999a; Rumbaugh ***etal..*** 1999b). En el momento de escribir este libro, una revisión importante del UML (UML 2.0) está en marcha.

Un método estructurado incluye un modelo del proceso de diseño, notaciones para representar el diseño, formatos de informes, reglas y pautas de diseño. Los métodos estructurados pueden ayudar a alguno o a la totalidad de los siguientes modelos de un sistema:

1. Un modelo que objetos que muestra las clases de objetos utilizadas en el sistema y sus dependencias.
2. Un modelo de secuencias que muestra cómo interactúan los objetos en el sistema cuando éste se ejecuta.
3. Un modelo del estado de transición que muestra los estados del sistema y los disparadores de las transiciones desde un estado a otro.

4. Un modelo estructural en el cual se documentan los componentes del sistema y sus agregaciones.
5. Un modelo de flujo de datos en el que el sistema se modela utilizando la transformación de datos que tiene lugar cuando se procesan. Éste no se utiliza normalmente en los métodos orientados a objetos, pero todavía se utiliza frecuentemente en el diseño de sistemas de tiempo real y de negocio.

En la práctica, los «métodos» estructurados son realmente notaciones estándar que comprenden prácticas aceptables. Si se siguen estos métodos y se aplican las pautas, puede obtenerse un diseño razonable. La creatividad del diseñador aún se requiere para decidir la descomposición del sistema y asegurar que el diseño capte de forma adecuada la especificación del mismo. Los estudios empíricos de los diseñadores (Bansler y Bcídker, 1993) muestran que éstos raramente siguen los métodos convencionales. Seleccionan y eligen de las pautas según circunstancias locales.

El desarrollo de un programa para implementar el sistema se sigue de forma natural del proceso de diseño. Aunque algunos programas, como los de los sistemas críticos de seguridad, se diseñan en detalle antes de que se inicie cualquier implementación. es más común que las primeras etapas de diseño y desarrollo de programas estén entrelazadas. Las herramientas C A S E se pueden utilizar para generar un programa esqueleto a partir de un diseño. Esto incluye código para definir e implementar las interfaces, y en muchos casos el desarrollador sólo necesita agregar detalles del funcionamiento de cada componente del programa.

La programación es una actividad personal y no existe un proceso general que se siga comúnmente. Algunos programadores empiezan con los componentes que comprenden, los desarrollan y después continúan con los que comprenden menos. Otros toman el enfoque opuesto, dejando los componentes que son más familiares hasta el final debido a que saben cómo desarrollarlos. Algunos desarrolladores prefieren definir los datos al inicio del proceso y los utilizan para conducir el desarrollo del programa; otros dejan los datos sin especificar tanto como sea posible.

Normalmente, los programadores llevan a cabo algunas pruebas del código que han desarrollado. A menudo esto muestra defectos en el programa que se deben eliminar del mismo. Esto se denomina ***depuración.*** Las pruebas y la depuración de defectos son procesos diferentes. Las pruebas establecen la existencia de defectos. La depuración comprende la localización y corrección de estos defectos.

La Figura 4.8 ilustra las etapas de la depuración. Los defectos en el código se localizan y el programa se modifica para cumplir los requerimientos. Las pruebas se deben entonces repetir para asegurar que los cambios se han efectuado correctamente. Así, el proceso de depuración es parte tanto del desarrollo como de las pruebas del software.

Al depurar, se generan hipótesis acerca del comportamiento que se observa en el programa; después, se prueban estas hipótesis con la esperanza de encontrar el defecto que origina la anomalía en la salida. Probar las hipótesis puede implicar realizar una traza manual del código del programa. Pueden escribirse nuevos casos de pruebas para localizar el problema. Para ayudar al proceso de depuración, se pueden utilizar herramientas de depuración interactiva que muestran los valores intermedios de las variables del programa y una traza de las sentencias ejecutadas.

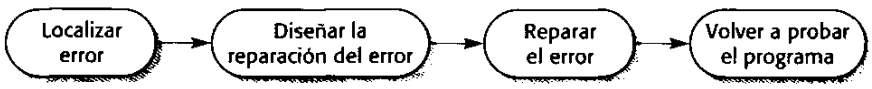


Figura 4.8 El proceso de depuración.

4.3.3 Validación del software

La validación del software o, de forma más general, la verificación y validación (V & V) se utiliza para mostrar que el sistema se ajusta a su especificación y que cumple las expectativas del usuario que lo comprará. Implica procesos de comprobación, como las inspecciones y revisiones (véase el Capítulo 22), en cada etapa del proceso del software desde la definición de requerimientos hasta el desarrollo del programa. Sin embargo, la mayoría de los costos de validación aparecen después de la implementación, cuando se prueba el funcionamiento del sistema (Capítulo 23).

A excepción de los programas pequeños, los sistemas no se deben probar como una simple unidad monolítica. La Figura 4.9 muestra un proceso de pruebas de tres etapas en el cual se prueban los componentes del sistema, la integración del sistema y, finalmente, el sistema con los datos del cliente. En el mejor de los casos, los defectos se descubren en las etapas iniciales del proceso y los problemas con la interfaz, cuando el sistema se integra. Sin embargo, cuando se descubren defectos el programa debe depurarse y esto puede requerir la repetición de otras etapas del proceso de pruebas. Los errores en los componentes del programa pueden descubrirse durante las pruebas del sistema. Por lo tanto, el proceso es iterativo y se retroalimenta tanto de las últimas etapas como de la primera parte del proceso.

Las etapas del proceso de pruebas son:

1. ***Prueba de componentes (o unidades).*** Se prueban los componentes individuales para asegurarse de que funcionan correctamente. Cada uno se prueba de forma independiente, sin los otros componentes del sistema. Los componentes pueden ser entidades simples como funciones o clases de objetos, o pueden ser agrupaciones coherentes de estas entidades.
2. ***Prueba de! sistema.*** Los componentes se integran para formar el sistema. Esle proceso comprende encontrar errores que son el resultado de interacciones no previstas entre los componentes y su interfaz. También comprende validar que el sistema cumpla sus requerimientos funcionales y no funcionales y probar las propiedades emergentes del sistema. Para sistemas grandes, esto puede ser un proceso gradual en el cual los componentes se integran para formar subsistemas que son probados individualmente antes de que ellos mismos se integren para formar el sistema final.
3. ***Prueba de aceptación.*** Es la etapa final en el proceso de pruebas antes de que se acepte que el sistema se ponga en funcionamiento. Este se prueba con los datos proporcionados por el cliente más que con datos de prueba simulados. Debido a la diferencia existente entre los datos reales y los de prueba, la prueba de aceptación puede revelar errores y omisiones en la definición de requerimientos del sistema. También puede revelar problemas en los requerimientos donde los recursos del sistema no cumplen las necesidades del usuario o donde el desempeño del sistema es inaceptable.

Normalmente, el desarrollo de componentes y las pruebas se entrelazan. Los programadores definen sus propios datos de prueba y de forma incremental prueban el código que se

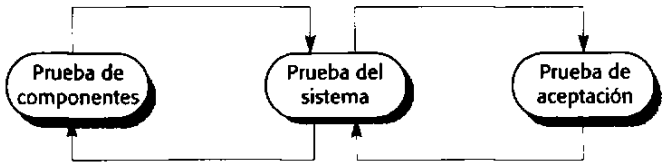


Figura 4.9 El proceso de pruebas.

va desarrollando. Éste es un enfoque económicamente razonable puesto que el programador es el que mejor conoce los componentes y es, por lo tanto, la mejor persona para generar los casos de prueba.

Si se utiliza un enfoque incremental de desarrollo, cada incremento debe ser probado cuando se desarrolla, con estas pruebas basadas en los requerimientos de ese incremento. En la programación extrema, las pruebas se desarrollan junto con los requerimientos antes de que empiece el desarrollo. Esto ayuda a los probadores y desarrolladores a entender los requerimientos y asegurar que no hay retardos pues se crean los casos de prueba.

Las últimas etapas de prueba consisten en integrar el trabajo de los programadores y deben planificarse por adelantado. Un equipo independiente de probadores debe trabajar a partir de planes de prueba que se desarrollan desde de la especificación y diseño del sistema. La Figura 4.10 ilustra cómo los planes de prueba son el vínculo entre las actividades de prueba y de desarrollo.

La prueba de aceptación algunas veces se denomina prueba alfa. Los sistemas personalizados se desarrollan para un único cliente. El proceso de prueba alfa continúa hasta que el desarrollador del sistema y el cliente acuerdan que el sistema que se va a entregar es una implementación aceptable de los requerimientos del sistema.

Cuando un sistema se va a comercializar como un producto de software, a menudo se utiliza un proceso de prueba denominado prueba beta. Ésta comprende la entrega de un sistema a un número potencial de clientes que acuerdan utilizarlo, los cuales informan de los problemas a los desarrolladores del sistema. Esto expone el producto a un uso real y detecta los errores no identificados por los constructores del sistema. Después de esta retroalimenlación, el sistema se modifica y se entrega ya sea para una prueba beta adicional o para la venta.

4.3.4 Evolución del software

La flexibilidad de los sistemas software es una de las principales razones por la que más y más software se incorpora a los sistemas grandes y complejos. Una vez que se decide adquirir hardware, es muy costoso hacer cambios en su diseño. Sin embargo, se pueden hacer cambios al software en cualquier momento durante o después del desarrollo del sistema. Aun cambios importantes son todavía mucho más económicos que los correspondientes de los sistemas hardware.

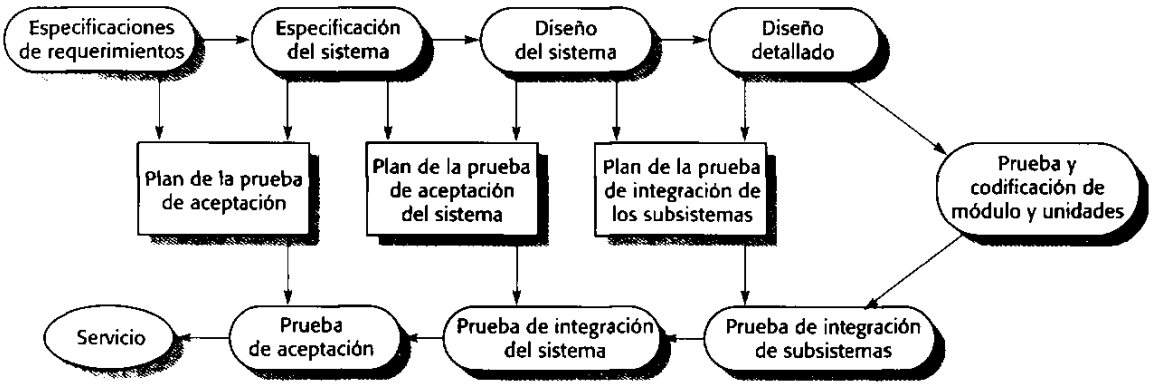


Figura 4.10 Las fases de prueba en el proceso del software.

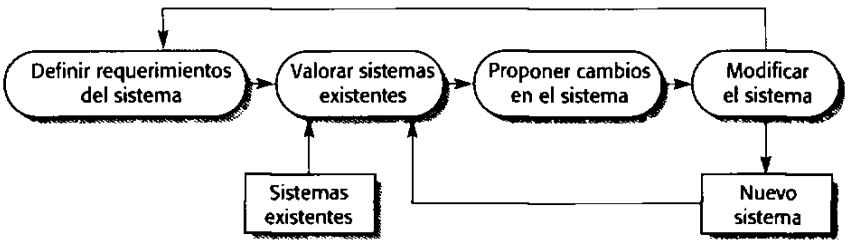


Figura 4.11 Evolución del sistema.

Históricamente, siempre ha existido una separación entre el proceso de desarrollo y el proceso de evolución del software (mantenimiento del software). La gente considera el desarrollo de software como una actividad creativa en la cual un sistema software se desarrolla desde un concepto inicial hasta que se pone en funcionamiento. Sin embargo, a veces consideran el mantenimiento del software como algo aburrido y sin interés. Aunque los costos de «mantenimiento» son a menudo varias veces los costos iniciales de desarrollo, el proceso de mantenimiento se considera a veces menos problemático que el desarrollo del software original.

Esta distinción entre el desarrollo y el mantenimiento es cada vez más irrelevante. Hoy en día, pocos sistemas software son completamente nuevos, lo que implica que tiene más sentido ver el desarrollo y el mantenimiento como actividades continuas. Más que dos procesos separados, es más realista considerar a la ingeniería del software como un proceso evolutivo (Figura 4.11) en el cual el software se cambia continuamente durante su periodo de vida como respuesta a los requerimientos cambiantes y necesidades del usuario.

4.4 El Proceso Unificado de Rational

El Proceso Unificado de Rational (RUP) es un ejemplo de un modelo de proceso moderno que proviene del trabajo en el U M L y el asociado Proceso Unificado de Desarrollo de Software (Rumbaugh ***et ai.,*** 1999b)^Se ha incluido aquí una descripción ya que es un buen ejemplo de un modelo de proceso híbrido. Reúne elementos de todos los modelos de procesos genéricos (Sección 4.1), iteraciones de apoyo (Sección 4.2) e ilustra buenas prácticas en la especificación y el diseño (Sección 4.3).

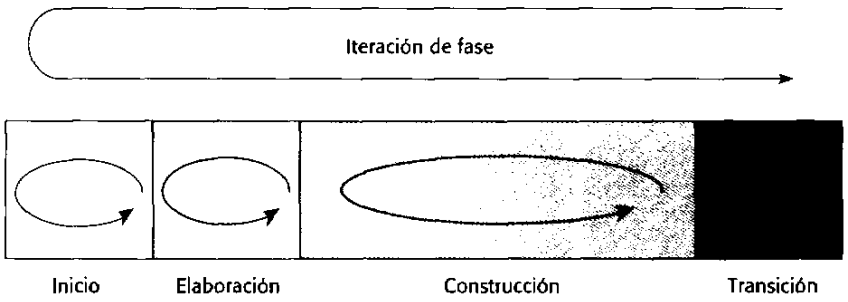
El RUP reconoce que los modelos de procesos genéricos presentan un sola enfoque del proceso. En contraste, el RUP se describe normalmente desde tres perspectivas:

1. Una perspectiva dinámica que muestra las fases del modelo sobre el tiempo.
2. Una perspectiva estática que muestra las actividades del proceso que se representan.
3. Una perspectiva práctica que sugiere buenas prácticas a utilizar durante el proceso.

La mayor parte de las descripciones del RUP intentan combinar las perspectivas estática y dinámica es un único diagrama (Krutchen. 2000). Esto hace el proceso más difícil de entender, por lo que aquí se utilizan descripciones separadas de cada una de estas perspectivas.

El RUP es un modelo en fases que identifica cuatro fases diferentes en el proceso del software. Sin embargo, a diferencia del modelo en cascada donde las fases se equiparan con las actividades del proceso, las fases en el RUP están mucho más relacionadas con asuntos de negocio más que técnicos. La Figura 4.12 muestra las fases en el RUP. Estas son:

1. ***Inicio.*** El objetivo de la fase de inicio es el de establecer un caso de negocio para el sistema. Se deben identificar todas las entidades externas (personas y sistemas) que



inleractuarán con el sistema y definir estas interacciones. Esta información se utiliza entonces para evaluar la aportación que el sistema hace al negocio. Si esta aportación es de poca importancia, se puede cancelar el proyecto después de esta fase.

2. ***Elaboración.*** Los objetivos de la fase de elaboración son desarrollar una comprensión del dominio del problema, establecer un marco de trabajo arquitectónico para el sistema, desarrollar el plan del proyecto e identificar los riesgos clave del proyecto. Al terminar esta fase, se debe tener un modelo de los requerimientos del sistema (se especifican los casos de uso UML), una descripción arquitectónica y un plan de desarrollo del software.
3. ***Construcción.*** La fase de construcción fundamentalmente comprende el diseno del sistema, la programación y las pruebas. Durante esta fase se desarrollan e integran las partes del sistema. Al terminar esta fase, debe tener un sistema software operativo y la documentación correspondiente lista para entregarla a los usuarios.
4. ***Transición.*** La fase final del RUP se ocupa de mover el sistema desde la comunidad de desarrollo a la comunidad del usuario y hacerlo trabajar en un entorno real. Esto se deja de lado en la mayor parte de los modelos de procesos del software pero es, en realidad, una actividad de alto costo y a veces problemática. Al terminar esta fase, se debe tener un sistema software documentado que funciona correctamente en su entorno operativo.

La iteración dentro del RUP es apoyada de dos formas, como se muestra en la Figura 4.12. Cada fase se puede representar de un modo iterativo con los resultados desarrollados incrementalmente. Además, el conjunto entero de fases puede también representarse de forma incremental, como se muestra en la citada figura por la flecha en forma de bucle desde la Transición hasta el Inicio.

La vista estática del RUP se centra en las actividades que tienen lugar durante el proceso de desarrollo. Estas se denominan ***flujos de trabajo*** en la descripción del RUP. Existen seis principales flujos de trabajo del proceso identificados en el proceso y tres principales flujos de trabajo de soporte. El RUP se ha diseñado conjuntamente con el UML —un lenguaje de modelado orientado a objetos—, por lo que la descripción del flujo de trabajo se orienta alrededor de los modelos UML asociados. En la Figura 4.13 se describen los principales flujos de trabajo de ingeniería y de soporte.

La ventaja de presentar perspectivas dinámicas y estáticas es que las fases del proceso de desarrollo no están asociadas con flujos de trabajo específicos. AI menos en principio, todos los flujos de trabajo del RUP pueden estar activos en todas las etapas del proceso. Por supuesto, la mayor parte del esfuerzo se realizará en flujos de trabajo tales como el modelado del negocio y los requerimientos en las primeras fases del proceso y en las pruebas y despliegue en las fases posteriores.

Modelado del negocio	Los procesos del negocio se modelan utilizando casos de uso de negocio.
Requerimientos	Se definen los actores que interactúan con el sistema y se desarrollan casos de uso para modelar los requerimientos del sistema.
Análisis y diseño	Se crea y documenta un modelo del diseño utilizando modelos arquitectónicos, modelos de componentes, modelos de objetos y modelos de secuencias.
Implementación	Se implementan y estructuran en subsistemas los componentes del sistema. La generación automática de código de los modelos del diseño ayuda a acelerar este proceso.
Pruebas	Las pruebas son un proceso iterativo que se llevan a cabo conjuntamente con la implementación. A la finalización de la implementación tienen lugar las pruebas del sistema.
Despliegue	Se crea una release del producto, se distribuye a los usuarios y se instala en su lugar de trabajo.
Configuración y cambios de gestión	Este flujo de trabajo de soporte gestiona los cambios del sistema (véase el Cap. 29).
Gestión del proyecto	Este flujo de trabajo de soporte gestiona el desarrollo del sistema (véase el Cap. 5).
Entorno	Este flujo de trabajo se refiere a hacer herramientas software apropiadas disponibles para los equipos de desarrollo de software.

Figura 4.13 Flujos de trabajo estáticos en el Proceso Unificado de Rational.

La perspectiva práctica en el RUP describe buenas prácticas de la ingeniería del software que son aconsejables en el desarrollo de sistemas. Se recomiendan seis buenas prácticas fundamentales:

1. ***Desarrolle el software de forma iterativa.*** Planifique incrementos del sistema basado en las prioridades del usuario y desarrollo y entregue las características del sistema de más alta prioridad al inicio del proceso de desarrollo.
2. ***Gestione los requerimientos.*** Documente explícitamente los requerimientos del cliente y manténgase al tanto de los cambios de estos requerimientos. Analice el impacto de los cambios en el sistema antes de aceptarlos.
3. ***Utilice arquitecturas basadas en componentes.*** Estructure la arquitectura del sistema en componentes como se indicó anteriormente en este capítulo.
4. ***Modele el software visualmente.*** Utilice modelos gráficos UML para presentar vistas estáticas y dinámicas del software.
5. ***Verifique la calidad del software.*** Asegure que el software cumple los estándares de calidad organizacionales.
6. ***Controle los cambios del software.*** Gestione los cambios del software usando un sistema de gestión de cambios y procedimientos y herramientas de gestión de configuraciones (véase el Capítulo 29).

El RUP no es un proceso apropiado para todos los tipos de desarrollo sino que representa una nueva generación de procesos genéricos. Las innovaciones más importantes son la separación de fases y los flujos de trabajo, y el reconocimiento de que la utilización del software en un entorno del usuario es parte del proceso. Las fases son dinámicas y tienen objetivos. Los flujos de trabajo son estáticos y son actividades técnicas que no están asociadas con fases únicas sino que pueden utilizarse durante el desarrollo para alcanzar los objetivos de cada fase.