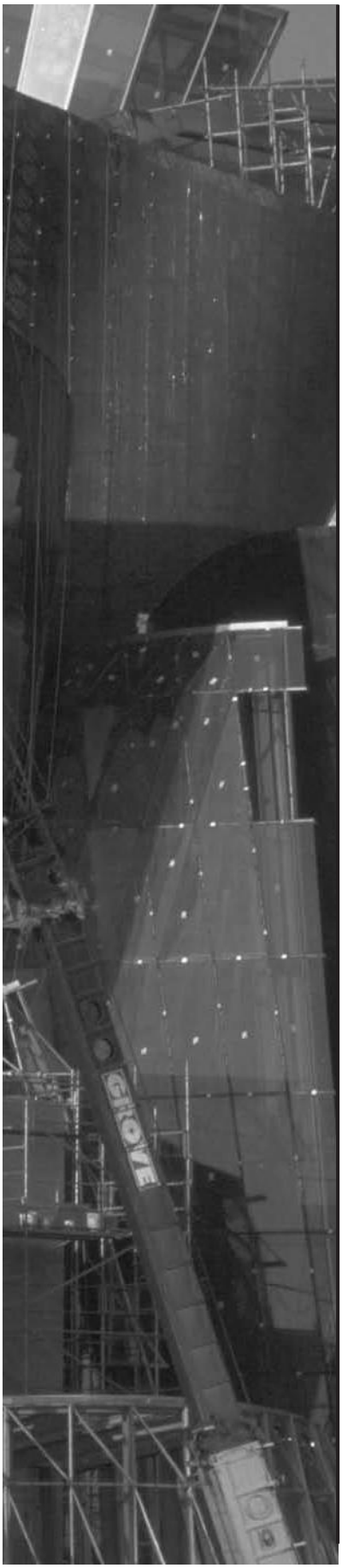

2

Procesos de software

Objetivos

El objetivo de este capítulo es introducirlo hacia la idea de un proceso de software: un conjunto coherente de actividades para la producción de software. Al estudiar este capítulo:

- comprenderá los conceptos y modelos sobre procesos de software;
- se introducirá en los tres modelos de proceso de software genérico y sabrá cuándo usarlos;
- entenderá las principales actividades del proceso de ingeniería de requerimientos de software, así como del desarrollo, las pruebas y la evolución del software;
- comprenderá por qué deben organizarse los procesos para enfrentar los cambios en los requerimientos y el diseño de software;
- entenderá cómo el Proceso Unificado Racional (Rational Unified Process, RUP) integra buenas prácticas de ingeniería de software para crear procesos de software adaptables.

Contenido

Un proceso de software es una serie de actividades relacionadas que conduce a la elaboración de un producto de software. Estas actividades pueden incluir el desarrollo de software desde cero en un lenguaje de programación estándar como Java o C. Sin embargo, las aplicaciones de negocios no se desarrollan precisamente de esta forma. El nuevo software empresarial con frecuencia ahora se desarrolla extendiendo y modificando los sistemas existentes, o configurando e integrando el software comercial o componentes del sistema.

Existen muchos diferentes procesos de software, pero todos deben incluir cuatro actividades que son fundamentales para la ingeniería de software:

1. *Especificación del software* Tienen que definirse tanto la funcionalidad del software como las restricciones de su operación.
2. *Diseño e implementación del software* Debe desarrollarse el software para cumplir con las especificaciones.
3. *Validación del software* Hay que validar el software para asegurarse de que cumple lo que el cliente quiere.
4. *Evolución del software* El software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

En cierta forma, tales actividades forman parte de todos los procesos de software. Por supuesto, en la práctica éstas son actividades complejas en sí mismas e incluyen subactividades tales como la validación de requerimientos, el diseño arquitectónico, la prueba de unidad, etcétera. También existen actividades de soporte al proceso, como la documentación y el manejo de la configuración del software.

Cuando los procesos se discuten y describen, por lo general se habla de actividades como especificar un modelo de datos, diseñar una interfaz de usuario, etcétera, así como del orden de dichas actividades. Sin embargo, al igual que las actividades, también las descripciones de los procesos deben incluir:

1. Productos, que son los resultados de una actividad del proceso. Por ejemplo, el resultado de la actividad del diseño arquitectónico es un modelo de la arquitectura de software.
2. Roles, que reflejan las responsabilidades de la gente que interviene en el proceso. Ejemplos de roles: gerente de proyecto, gerente de configuración, programador, etcétera.
3. Precondiciones y postcondiciones, que son declaraciones válidas antes y después de que se realice una actividad del proceso o se cree un producto. Por ejemplo, antes de comenzar el diseño arquitectónico, una precondición es que el cliente haya aprobado todos los requerimientos; después de terminar esta actividad, una postcondición podría ser que se revisen aquellos modelos UML que describen la arquitectura.

Los procesos de software son complejos y, como todos los procesos intelectuales y creativos, se apoyan en personas con capacidad de juzgar y tomar decisiones. No hay un proceso ideal; además, la mayoría de las organizaciones han diseñado sus propios procesos de desarrollo de software. Los procesos han evolucionado para beneficiarse de las capacidades de la gente en una organización y de las características específicas de los

sistemas que se están desarrollando. Para algunos sistemas, como los sistemas críticos, se requiere de un proceso de desarrollo muy estructurado. Para los sistemas empresariales, con requerimientos rápidamente cambiantes, es probable que sea más efectivo un proceso menos formal y flexible.

En ocasiones, los procesos de software se clasifican como dirigidos por un plan (*plan-driven*) o como procesos ágiles. Los procesos dirigidos por un plan son aquellos donde todas las actividades del proceso se planean por anticipado y el avance se mide contra dicho plan. En los procesos ágiles, que se estudiarán en el capítulo 3, la planeación es incremental y es más fácil modificar el proceso para reflejar los requerimientos cambiantes del cliente. Como plantean Boehm y Turner (2003), cada enfoque es adecuado para diferentes tipos de software. Por lo general, uno necesita encontrar un equilibrio entre procesos dirigidos por un plan y procesos ágiles.

Aunque no hay un proceso de software "ideal", en muchas organizaciones sí existe un ámbito para mejorar el proceso de software. Los procesos quizás incluyan técnicas obsoletas o tal vez no aprovechen las mejores prácticas en la industria de la ingeniería de software. En efecto, muchas organizaciones aún no sacan ventaja de los métodos de la ingeniería de software en su desarrollo de software.

Los procesos de software pueden mejorarse con la estandarización de los procesos, donde se reduce la diversidad en los procesos de software en una organización. Esto conduce a mejorar la comunicación, a reducir el tiempo de capacitación, y a que el soporte de los procesos automatizados sea más económico. La estandarización también representa un primer paso importante tanto en la introducción de nuevos métodos y técnicas de ingeniería de software, como en sus buenas prácticas. En el capítulo 26 se analiza con más detalle la mejora en el proceso de software.

2.1 Modelos de proceso de software

Como se explicó en el capítulo 1, un modelo de proceso de software es una representación simplificada de este proceso. Cada modelo del proceso representa a otro desde una particular perspectiva y, por lo tanto, ofrece sólo información parcial acerca de dicho proceso. Por ejemplo, un modelo de actividad del proceso muestra las actividades y su secuencia, pero quizá sin presentar los roles de las personas que intervienen en esas actividades. En esta sección se introducen algunos modelos de proceso muy generales (en ocasiones llamados "paradigmas de proceso") y se muestran desde una perspectiva arquitectónica. En otras palabras, se ve el marco (framework) del proceso, pero no los detalles de las actividades específicas.

Tales modelos genéricos no son descripciones definitivas de los procesos de software. Más bien, son abstracciones del proceso que se utilizan para explicar los diferentes enfoques del desarrollo de software. Se pueden considerar marcos del proceso que se extienden y se adaptan para crear procesos más específicos de ingeniería de software.

Los modelos del proceso que se examinan aquí son:

1. *El modelo en cascada* (*waterfall*) Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y, luego, los representa como fases separadas del proceso, tal como especificación de requerimientos, diseño de software, implementación, pruebas, etcétera.

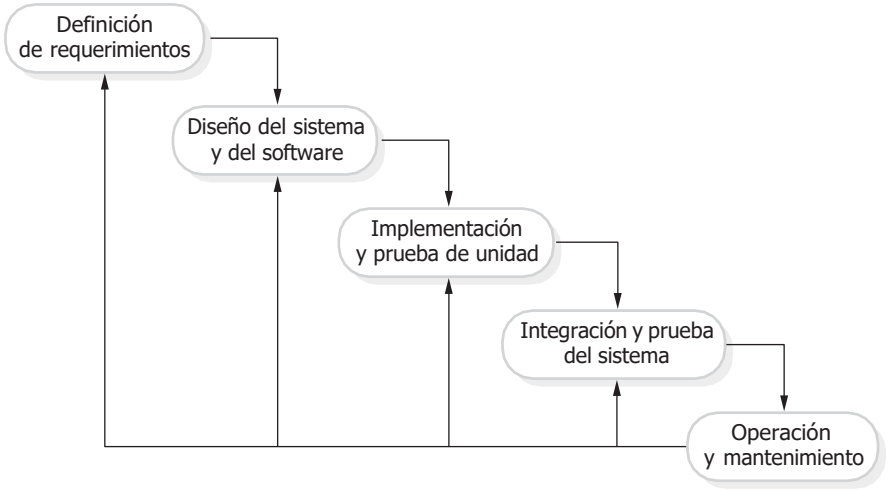

Figura 2.1 El modelo en cascada

2. *Desarrollo incremental* Este enfoque vincula las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones (incrementos), y cada versión añade funcionalidad a la versión anterior.

3. *Ingeniería de software orientada a la reutilización* Este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en la integración de estos componentes en un sistema, en vez de desarrollarlo desde cero.

Dichos modelos no son mutuamente excluyentes y con frecuencia se usan en conjunto, sobre todo para el desarrollo de grandes sistemas. Para este tipo de sistemas, tiene sentido combinar algunas de las mejores características de los modelos de desarrollo en cascada e incremental. Se necesita contar con información sobre los requerimientos esenciales del sistema para diseñar la arquitectura de software que apoye dichos requerimientos. No puede desarrollarse de manera incremental. Los subsistemas dentro de un sistema más grande se desarrollan usando diferentes enfoques. Partes del sistema que son bien comprendidas pueden especificarse y desarrollarse al utilizar un proceso basado en cascada. Partes del sistema que por adelantado son difíciles de especificar, como la interfaz de usuario, siempre deben desarrollarse con un enfoque incremental.

2.1.1 El modelo en cascada

El primer modelo publicado sobre el proceso de desarrollo de software se derivó a partir de procesos más generales de ingeniería de sistemas (Royce, 1970). Este modelo se ilustra en la figura 2.1. Debido al paso de una fase en cascada a otra, este modelo se conoce como "modelo en cascada" o ciclo de vida del software. El modelo en cascada es un ejemplo de un proceso dirigido por un plan; en principio, usted debe planear y programar todas las actividades del proceso, antes de comenzar a trabajar con ellas.

Las principales etapas del modelo en cascada reflejan directamente las actividades fundamentales del desarrollo:

1. *Análisis y definición de requerimientos* Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Luego, se definen con detalle y sirven como una especificación del sistema.
2. *Diseño del sistema y del software* El proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global. El diseño del software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.
3. *Implementación y prueba de unidad* Durante esta etapa, el diseño de software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.
4. *Integración y prueba de sistema* Las unidades del programa o los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.
5. *Operación y mantenimiento* Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

En principio, el resultado de cada fase consiste en uno o más documentos que se autorizaron ("firmaron"). La siguiente fase no debe comenzar sino hasta que termine la fase previa. En la práctica, dichas etapas se traslapan y se nutren mutuamente de información. Durante el diseño se identifican los problemas con los requerimientos. En la codificación se descubren problemas de diseño, y así sucesivamente. El proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra. Entonces, es posible que los documentos generados en cada fase deban modificarse para reflejar los cambios que se realizan.

Debido a los costos de producción y aprobación de documentos, las iteraciones suelen ser onerosas e implicar un rediseño significativo. Por lo tanto, después de un pequeño número de iteraciones, es normal detener partes del desarrollo, como la especificación, y continuar con etapas de desarrollo posteriores. Los problemas se dejan para una resolución posterior, se ignoran o se programan. Este freno prematuro de los requerimientos quizá signifique que el sistema no hará lo que el usuario desea. También podría conducir a sistemas mal estructurados conforme los problemas de diseño se evadan con la implementación de trucos.

Durante la fase final del ciclo de vida (operación y mantenimiento), el software se pone en servicio. Se descubren los errores y las omisiones en los requerimientos originales del software. Surgen los errores de programa y diseño, y se detecta la necesidad de nueva funcionalidad. Por lo tanto, el sistema debe evolucionar para mantenerse útil. Hacer tales cambios (mantenimiento de software) puede implicar la repetición de etapas anteriores del proceso.

Ingeniería de software de cuarto limpio

Un ejemplo del proceso de desarrollo formal, diseñado originalmente por IBM, es el proceso de cuarto limpio (*cleanroom*). En el proceso de cuarto limpio, cada incremento de software se especifica formalmente y tal especificación se transforma en una implementación. La exactitud del software se demuestra mediante un enfoque formal. No hay prueba de unidad para defectos en el proceso y la prueba del sistema se enfoca en la valoración de la fiabilidad del sistema.

El objetivo del proceso de cuarto limpio es obtener un software con cero defectos, de modo que los sistemas que se entreguen cuenten con un alto nivel de fiabilidad.

http://www.SoftwareEngineering-9.com/Web/Cleanroom/

El modelo en cascada es consecuente con otros modelos del proceso de ingeniería y en cada fase se produce documentación. Esto hace que el proceso sea visible, de modo que los administradores monitoricen el progreso contra el plan de desarrollo. Su principal problema es la partición inflexible del proyecto en distintas etapas. Tienen que establecerse compromisos en una etapa temprana del proceso, lo que dificulta responder a los requerimientos cambiantes del cliente.

En principio, el modelo en cascada sólo debe usarse cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sistema. Sin embargo, el modelo en cascada refleja el tipo de proceso utilizado en otros proyectos de ingeniería. Como es más sencillo emplear un modelo de gestión común durante todo el proyecto, aún son de uso común los procesos de software basados en el modelo en cascada.

Una variación importante del modelo en cascada es el desarrollo de sistemas formales, donde se crea un modelo matemático para una especificación del sistema. Después se corrige este modelo, mediante transformaciones matemáticas que preservan su consistencia en un código ejecutable. Con base en la suposición de que son correctas sus transformaciones matemáticas, se puede aseverar, por lo tanto, que un programa generado de esta forma es consecuente con su especificación.

Los procesos formales de desarrollo, como el que se basa en el método B (Schneider, 2001; Wordsworth, 1996) son muy adecuados para el desarrollo de sistemas que cuenten con rigurosos requerimientos de seguridad, fiabilidad o protección. El enfoque formal simplifica la producción de un caso de protección o seguridad. Esto demuestra a los clientes o reguladores que el sistema en realidad cumple sus requerimientos de protección o seguridad.

Los procesos basados en transformaciones formales se usan por lo general sólo en el desarrollo de sistemas críticos para protección o seguridad. Requieren experiencia especializada. Para la mayoría de los sistemas, este proceso no ofrece costo/beneficio significativos sobre otros enfoques en el desarrollo de sistemas.

2.1.2 Desarrollo incremental

El desarrollo incremental se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado (figura 2.2). Las actividades de especificación, desarrollo

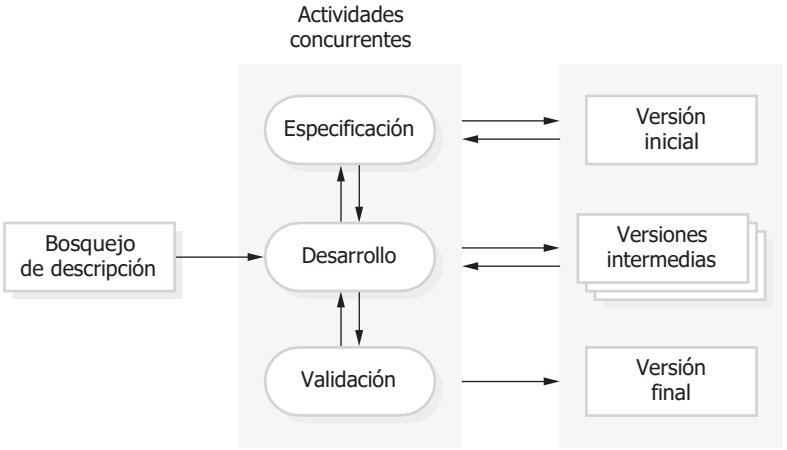

Figura 2.2 Desarrollo incremental

y validación están entrelazadas en vez de separadas, con rápida retroalimentación a través de las actividades.

El desarrollo de software incremental, que es una parte fundamental de los enfoques ágiles, es mejor que un enfoque en cascada para la mayoría de los sistemas empresariales, de comercio electrónico y personales. El desarrollo incremental refleja la forma en que se resuelven problemas. Rara vez se trabaja por adelantado una solución completa del problema, más bien se avanza en una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores. Al desarrollar el software de manera incremental, resulta más barato y fácil realizar cambios en el software conforme éste se diseña.

Cada incremento o versión del sistema incorpora algunas de las funciones que necesita el cliente. Por lo general, los primeros incrementos del sistema incluyen la función más importante o la más urgente. Esto significa que el cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana, para constatar si se entrega lo que se requiere. En caso contrario, sólo el incremento actual debe cambiarse y, posiblemente, definir una nueva función para incrementos posteriores.

Comparado con el modelo en cascada, el desarrollo incremental tiene tres beneficios importantes:

1. Se reduce el costo de adaptar los requerimientos cambiantes del cliente. La cantidad de análisis y la documentación que tiene que reelaborarse son mucho menores de lo requerido con el modelo en cascada.
2. Es más sencillo obtener retroalimentación del cliente sobre el trabajo de desarrollo que se realizó. Los clientes pueden comentar las demostraciones del software y darse cuenta de cuánto se ha implementado. Los clientes encuentran difícil juzgar el avance a partir de documentos de diseño de software.
3. Es posible que sea más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad. Los clientes tienen posibilidad de usar y ganar valor del software más temprano de lo que sería posible con un proceso en cascada.

Problemas con el desarrollo incremental

Aunque el desarrollo incremental tiene muchas ventajas, no está exento de problemas. La principal causa de la dificultad es el hecho de que las grandes organizaciones tienen procedimientos burocráticos que han evolucionado con el tiempo y pueden suscitar falta de coordinación entre dichos procedimientos y un proceso iterativo o ágil más informal.

En ocasiones, tales procedimientos se hallan ahí por buenas razones: por ejemplo, pueden existir procedimientos para garantizar que el software implementa de manera adecuada regulaciones externas (en Estados Unidos, por ejemplo, las regulaciones de contabilidad Sarbanes-Oxley). El cambio de tales procedimientos podría resultar imposible, de manera que los conflictos son inevitables.

http://www.SoftwareEngineering-9.com/Web/IncrementalDev/

El desarrollo incremental ahora es en cierta forma el enfoque más común para el desarrollo de sistemas de aplicación. Este enfoque puede estar basado en un plan, ser ágil o, más usualmente, una mezcla de dichos enfoques. En un enfoque basado en un plan se identifican por adelantado los incrementos del sistema; si se adopta un enfoque ágil, se detectan los primeros incrementos, aunque el desarrollo de incrementos posteriores depende del avance y las prioridades del cliente.

Desde una perspectiva administrativa, el enfoque incremental tiene dos problemas:

1. El proceso no es visible. Los administradores necesitan entregas regulares para medir el avance. Si los sistemas se desarrollan rápidamente, resulta poco efectivo en términos de costos producir documentos que reflejen cada versión del sistema.
2. La estructura del sistema tiende a degradarse conforme se tienen nuevos incrementos. A menos que se gaste tiempo y dinero en la refactorización para mejorar el software, el cambio regular tiende a corromper su estructura. La incorporación de más cambios de software se vuelve cada vez más difícil y costosa.

Los problemas del desarrollo incremental se tornan particularmente agudos para sistemas grandes, complejos y de larga duración, donde diversos equipos desarrollan diferentes partes del sistema. Los grandes sistemas necesitan de un marco o una arquitectura estable y es necesario definir con claridad, respecto a dicha arquitectura, las responsabilidades de los distintos equipos que trabajan en partes del sistema. Esto debe planearse por adelantado en vez de desarrollarse de manera incremental.

Se puede desarrollar un sistema incremental y exponerlo a los clientes para su comentario, sin realmente entregarlo e implementarlo en el entorno del cliente. La entrega y la implementación incrementales significan que el software se usa en procesos operacionales reales. Esto no siempre es posible, ya que la experimentación con un nuevo software llega a alterar los procesos empresariales normales. En la sección 2.3.2 se estudian las ventajas y desventajas de la entrega incremental.

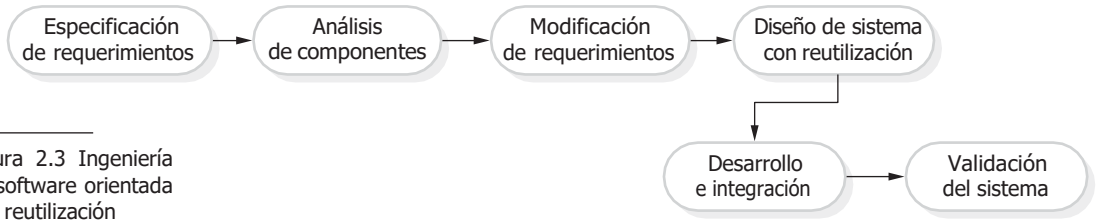

Figura 2.3 Ingeniería de software orientada a la reutilización

2.1.3 Ingeniería de software orientada a la reutilización

En la mayoría de los proyectos de software hay cierta reutilización de software. Sucede con frecuencia de manera informal, cuando las personas que trabajan en el proyecto conocen diseños o códigos que son similares a lo que se requiere. Los buscan, los modifican según se necesite y los incorporan en sus sistemas.

Esta reutilización informal ocurre independientemente del proceso de desarrollo que se emplee. Sin embargo, en el siglo XXI, los procesos de desarrollo de software que se enfocaban en la reutilización de software existente se utilizan ampliamente. Los enfoques orientados a la reutilización se apoyan en una gran base de componentes de software reutilizable y en la integración de marcos para la composición de dichos componentes. En ocasiones, tales componentes son sistemas por derecho propio (sistemas comerciales, off-the-shelf o COTS) que pueden mejorar la funcionalidad específica, como el procesador de textos o la hoja de cálculo.

En la figura 2.3 se muestra un modelo del proceso general para desarrollo basado en reutilización. Aunque la etapa inicial de especificación de requerimientos y la etapa de validación se comparan con otros procesos de software en un proceso orientado a la reutilización, las etapas intermedias son diferentes. Dichas etapas son:

1. *Análisis de componentes* Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación. Por lo general, no hay coincidencia exacta y los componentes que se usan proporcionan sólo parte de la funcionalidad requerida.

2. *Modificación de requerimientos* Durante esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles. Donde las modificaciones son imposibles, puede regresarse a la actividad de análisis de componentes para buscar soluciones alternativas.

3. *Diseño de sistema con reutilización* Durante esta fase se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente. Los creadores toman en cuenta los componentes que se reutilizan y organizan el marco de referencia para atenderlo. Es posible que deba diseñarse algo de software nuevo, si no están disponibles los componentes reutilizables.

4. *Desarrollo e integración* Se diseña el software que no puede procurarse de manera externa, y se integran los componentes y los sistemas COTS para crear el nuevo sistema. La integración del sistema, en este modelo, puede ser parte del proceso de desarrollo, en vez de una actividad independiente.

Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a la reutilización:

1. Servicios Web que se desarrollan en concordancia para atender servicios estándares y que están disponibles para la invocación remota.
2. Colecciones de objetos que se desarrollan como un paquete para su integración con un marco de componentes como .NET o J2EE.
3. Sistemas de software independientes que se configuran para usar en un entorno particular.

La ingeniería de software orientada a la reutilización tiene la clara ventaja de reducir la cantidad de software a desarrollar y, por lo tanto, la de disminuir costos y riesgos; por lo general, también conduce a entregas más rápidas del software. Sin embargo, son inevitables los compromisos de requerimientos y esto conduciría hacia un sistema que no cubra las necesidades reales de los usuarios. Más aún, se pierde algo de control sobre la evolución del sistema, conforme las nuevas versiones de los componentes reutilizables no estén bajo el control de la organización que los usa.

La reutilización de software es muy importante y en la tercera parte del libro se dedican varios capítulos a este tema. En el capítulo 16 se tratan los conflictos generales de la reutilización de software y la reutilización de COTS, en los capítulos 17 y 18 se estudia la ingeniería de software basada en componentes, y en el capítulo 19 se explican los sistemas orientados al servicio.