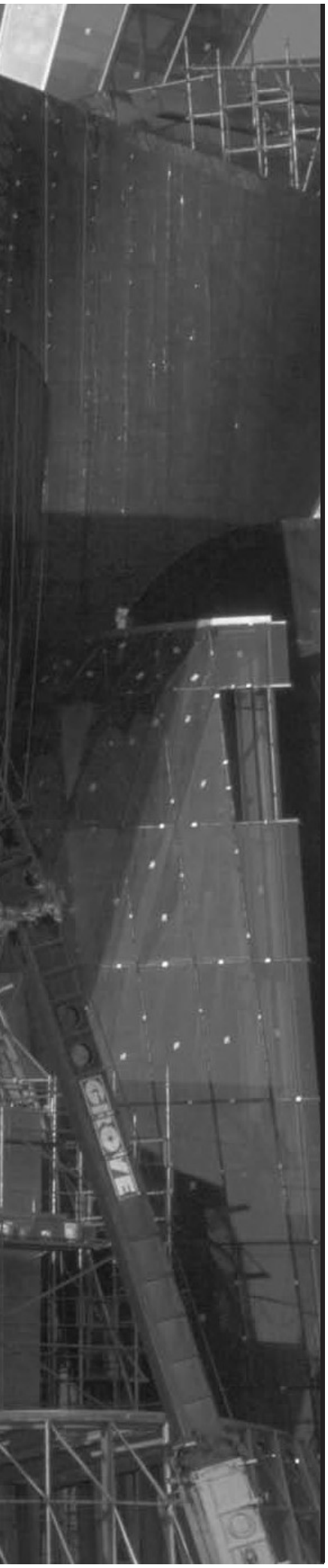


2

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.

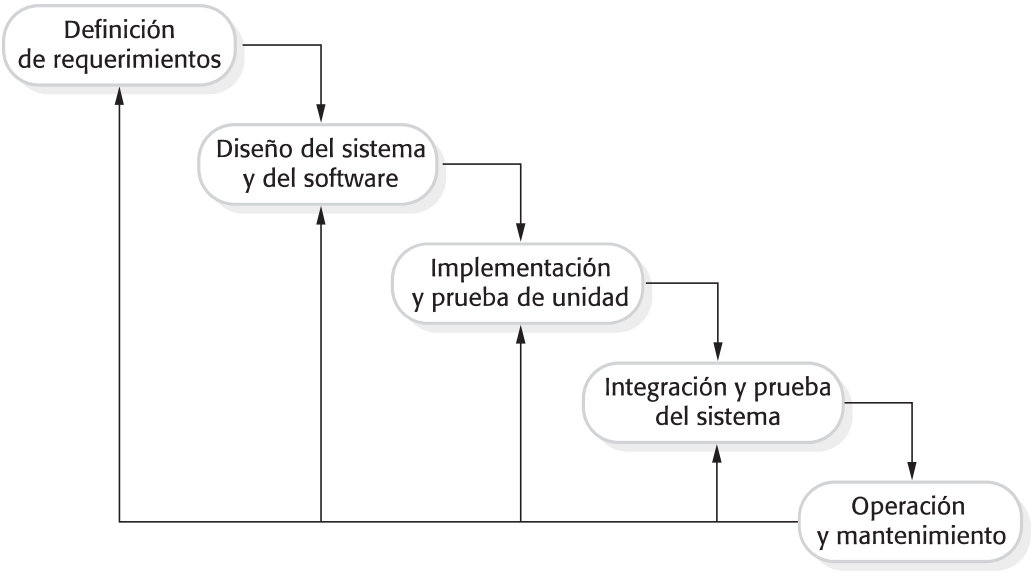


Figura 2.1 El modelo en cascada

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

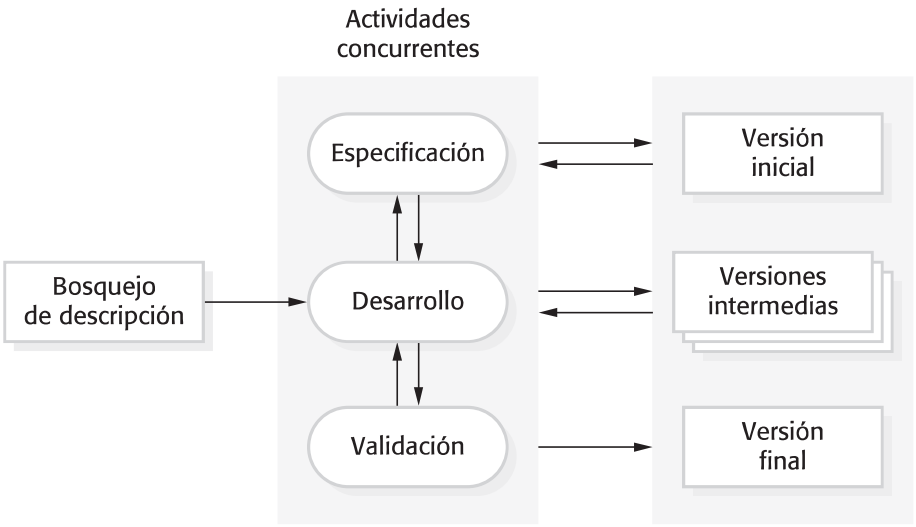


Figura 2.2 Desarrollo incremental

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.

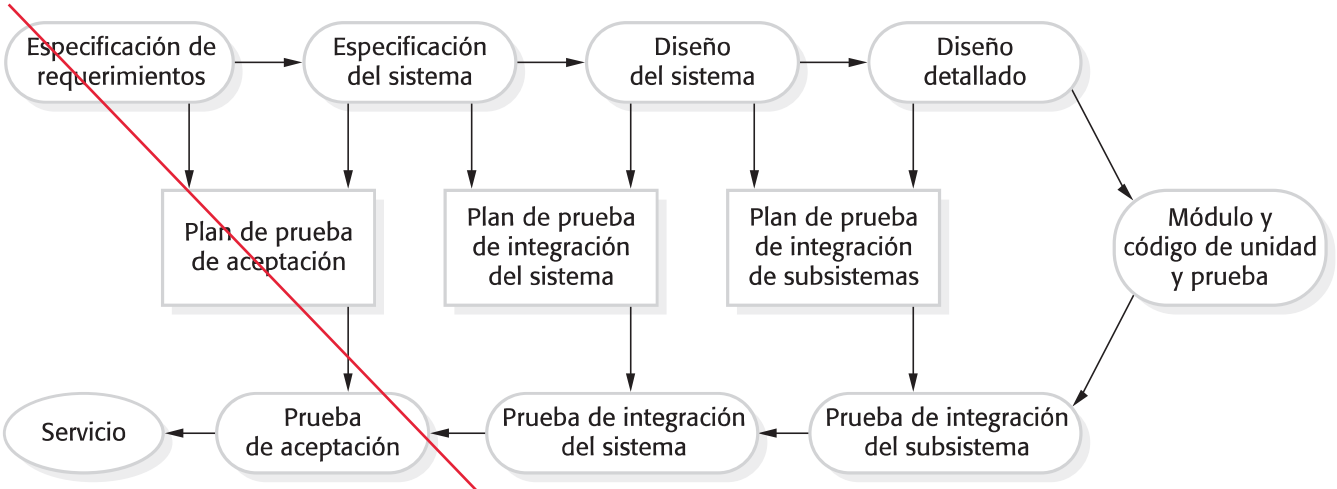


Figura 2.7 Probando fases en un proceso de software dirigido por un plan

2.2.4 Evolución del software

La flexibilidad de los sistemas de software es una de las razones principales por las que cada vez más software se incorpora en los sistemas grandes y complejos. Una vez tomada la decisión de fabricar hardware, resulta muy costoso hacer cambios a su diseño. Sin embargo, en cualquier momento durante o después del desarrollo del sistema, pueden hacerse cambios al software. Incluso los cambios mayores son todavía más baratos que los correspondientes cambios al hardware del sistema.

En la historia, siempre ha habido división entre el proceso de desarrollo del software y el proceso de evolución del software (mantenimiento de software). Las personas consideran el desarrollo de software como una actividad creativa, en la cual se diseña un sistema de software desde un concepto inicial y a través de un sistema de trabajo. No obstante, consideran en ocasiones el mantenimiento del software como insulso y poco interesante. Aunque en la mayoría de los casos los costos del mantenimiento son varias veces los costos iniciales de desarrollo, los procesos de mantenimiento se consideran en ocasiones como menos desafiantes que el desarrollo de software original.

Esta distinción entre desarrollo y mantenimiento es cada vez más irrelevante. Es muy difícil que cualquier sistema de software sea un sistema completamente nuevo, y tiene mucho más sentido ver el desarrollo y el mantenimiento como un continuo. En lugar de dos procesos separados, es más realista pensar en la ingeniería de software como un proceso evolutivo (figura 2.8), donde el software cambia continuamente a lo largo de su vida, en función de los requerimientos y las necesidades cambiantes del cliente.

2.3 Cómo enfrentar el cambio

El cambio es inevitable en todos los grandes proyectos de software. Los requerimientos del sistema varían conforme la empresa procura que el sistema responda a presiones externas y se modifican las prioridades administrativas. A medida que se ponen a disposición nuevas tecnologías, surgen nuevas posibilidades de diseño e implementación. Por ende, cualquiera que sea el modelo del proceso de software utilizado, es esencial que ajuste los cambios al software a desarrollar.

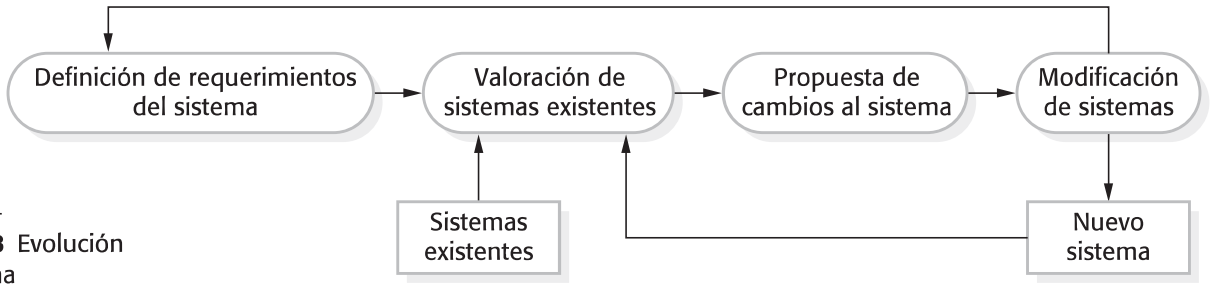


Figura 2.8 Evolución del sistema

El cambio se agrega a los costos del desarrollo de software debido a que, por lo general, significa que el trabajo ya terminado debe volver a realizarse. A esto se le llama rehacer. Por ejemplo, si se analizaron las relaciones entre los requerimientos en un sistema y se identifican nuevos requerimientos, parte o todo el análisis de requerimientos tiene que repetirse. Entonces, es necesario rediseñar el sistema para entregar los nuevos requerimientos, cambiar cualquier programa que se haya desarrollado y volver a probar el sistema.

Existen dos enfoques relacionados que se usan para reducir los costos del rehacer:

1. Evitar el cambio, donde el proceso de software incluye actividades que anticipan cambios posibles antes de requerirse la labor significativa de rehacer. Por ejemplo, puede desarrollarse un sistema prototipo para demostrar a los clientes algunas características clave del sistema. Ellos podrán experimentar con el prototipo y refinar sus requerimientos, antes de comprometerse con mayores costos de producción de software.
2. Tolerancia al cambio, donde el proceso se diseña de modo que los cambios se ajusten con un costo relativamente bajo. Por lo general, esto comprende algunas formas de desarrollo incremental. Los cambios propuestos pueden implementarse en incrementos que aún no se desarrollan. Si no es posible, entonces tal vez sólo un incremento (una pequeña parte del sistema) tendría que alterarse para incorporar el cambio.

En esta sección se estudian dos formas de enfrentar el cambio y los requerimientos cambiantes del sistema. Se trata de lo siguiente:

1. Prototipo de sistema, donde rápidamente se desarrolla una versión del sistema o una parte del mismo, para comprobar los requerimientos del cliente y la factibilidad de algunas decisiones de diseño. Esto apoya el hecho de evitar el cambio, al permitir que los usuarios experimenten con el sistema antes de entregarlo y así refinar sus requerimientos. Como resultado, es probable que se reduzca el número de propuestas de cambio de requerimientos posterior a la entrega.
2. Entrega incremental, donde los incrementos del sistema se entregan al cliente para su comentario y experimentación. Esto apoya tanto al hecho de evitar el cambio como a tolerar el cambio. Por un lado, evita el compromiso prematuro con los requerimientos para todo el sistema y, por otro, permite la incorporación de cambios en incrementos mayores a costos relativamente bajos.

La noción de refactorización, esto es, el mejoramiento de la estructura y organización de un programa, es también un mecanismo importante que apoya la tolerancia al cambio. Este tema se explica en el capítulo 3, que se ocupa de los métodos ágiles.

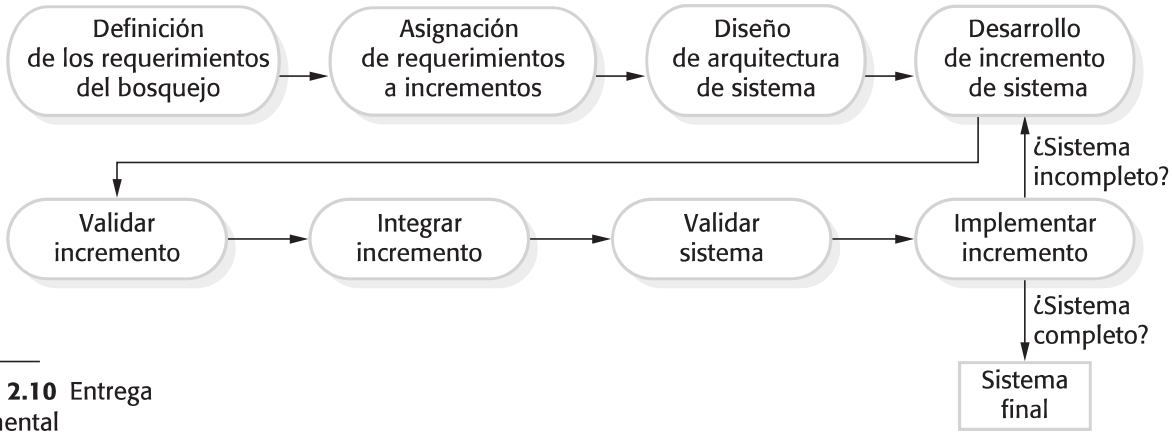


Figura 2.10 Entrega incremental

2.3.2 Entrega incremental

La entrega incremental (figura 2.10) es un enfoque al desarrollo de software donde algunos de los incrementos diseñados se entregan al cliente y se implementan para usarse en un entorno operacional. En un proceso de entrega incremental, los clientes identifican, en un bosquejo, los servicios que proporciona el sistema. Identifican cuáles servicios son más importantes y cuáles son menos significativos para ellos. Entonces, se define un número de incrementos de entrega, y cada incremento proporciona un subconjunto de la funcionalidad del sistema. La asignación de servicios por incrementos depende de la prioridad del servicio, donde los servicios de más alta prioridad se implementan y entregan primero.

Una vez identificados los incrementos del sistema, se definen con detalle los requerimientos de los servicios que se van a entregar en el primer incremento, y se desarrolla ese incremento. Durante el desarrollo, puede haber un mayor análisis de requerimientos para incrementos posteriores, aun cuando se rechacen cambios de requerimientos para el incremento actual.

Una vez completado y entregado el incremento, los clientes lo ponen en servicio. Esto significa que toman la entrega anticipada de la funcionalidad parcial del sistema. Pueden experimentar con el sistema que les ayuda a clarificar sus requerimientos, para posteriores incrementos del sistema. A medida que se completan nuevos incrementos, se integran con los incrementos existentes, de modo que con cada incremento entregado mejore la funcionalidad del sistema.

La entrega incremental tiene algunas ventajas:

1. Los clientes pueden usar los primeros incrementos como prototipos y adquirir experiencia que informe sobre sus requerimientos, para posteriores incrementos del sistema. A diferencia de los prototipos, éstos son parte del sistema real, de manera que no hay reaprendizaje cuando está disponible el sistema completo.
2. Los clientes deben esperar hasta la entrega completa del sistema, antes de ganar valor del mismo. El primer incremento cubre sus requerimientos más críticos, de modo que es posible usar inmediatamente el software.
3. El proceso mantiene los beneficios del desarrollo incremental en cuanto a que debe ser relativamente sencillo incorporar cambios al sistema.
4. Puesto que primero se entregan los servicios de mayor prioridad y luego se integran los incrementos, los servicios de sistema más importantes reciben mayores pruebas.

Esto significa que los clientes tienen menos probabilidad de encontrar fallas de software en las partes más significativas del sistema.

Sin embargo, existen problemas con la entrega incremental:

1. La mayoría de los sistemas requieren de una serie de recursos que se utilizan para diferentes partes del sistema. Dado que los requerimientos no están definidos con detalle sino hasta que se implementa un incremento, resulta difícil identificar recursos comunes que necesiten todos los incrementos.
2. Asimismo, el desarrollo iterativo resulta complicado cuando se diseña un sistema de reemplazo. Los usuarios requieren de toda la funcionalidad del sistema antiguo, ya que es común que no deseen experimentar con un nuevo sistema incompleto. Por lo tanto, es difícil conseguir retroalimentación útil del cliente.
3. La esencia de los procesos iterativos es que la especificación se desarrolla en conjunto con el software. Sin embargo, esto se puede contradecir con el modelo de adquisiciones de muchas organizaciones, donde la especificación completa del sistema es parte del contrato de desarrollo del sistema. En el enfoque incremental, no hay especificación completa del sistema, sino hasta que se define el incremento final. Esto requiere una nueva forma de contrato que los grandes clientes, como las agencias gubernamentales, encontrarían difícil de adoptar.

Existen algunos tipos de sistema donde el desarrollo incremental y la entrega no son el mejor enfoque. Hay sistemas muy grandes donde el desarrollo incluye equipos que trabajan en diferentes ubicaciones, algunos sistemas embebidos donde el software depende del desarrollo de hardware y algunos sistemas críticos donde todos los requerimientos tienen que analizarse para comprobar las interacciones que comprometan la seguridad o protección del sistema.

Estos sistemas, desde luego, enfrentan los mismos problemas de incertidumbre y requerimientos cambiantes. En consecuencia, para solucionar tales problemas y obtener algunos de los beneficios del desarrollo incremental, se utiliza un proceso donde un prototipo del sistema se elabore iterativamente y se utilice como plataforma, para experimentar con los requerimientos y el diseño del sistema. Con la experiencia obtenida del prototipo, pueden concertarse los requerimientos definitivos.

2.3.3 Modelo en espiral de Boehm

Boehm (1988) propuso un marco del proceso de software dirigido por el riesgo (el modelo en espiral), que se muestra en la figura 2.11. Aquí, el proceso de software se representa como una espiral, y no como una secuencia de actividades con cierto retroceso de una actividad a otra. Cada ciclo en la espiral representa una fase del proceso de software. Por ende, el ciclo más interno puede relacionarse con la factibilidad del sistema, el siguiente ciclo con la definición de requerimientos, el ciclo que sigue con el diseño del sistema, etcétera. El modelo en espiral combina el evitar el cambio con la tolerancia al cambio. Lo anterior supone que los cambios son resultado de riesgos del proyecto e incluye actividades de gestión de riesgos explícitas para reducir tales riesgos.

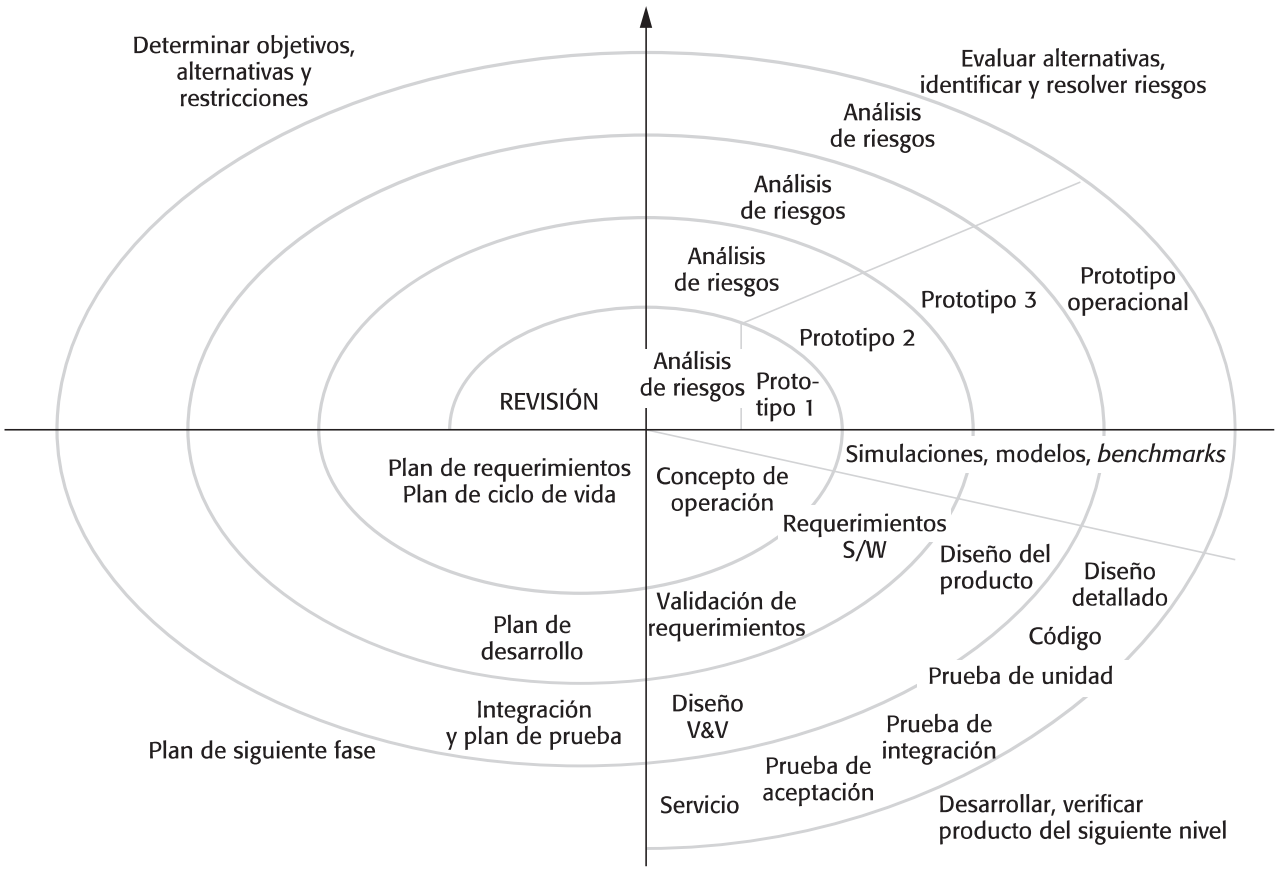


Figura 2.11 Modelo en espiral de Boehm del proceso de software (© IEEE, 1988)

Cada ciclo en la espiral se divide en cuatro sectores:

1. *Establecimiento de objetivos* Se definen objetivos específicos para dicha fase del proyecto. Se identifican restricciones en el proceso y el producto, y se traza un plan de gestión detallado. Se identifican los riesgos del proyecto. Pueden planearse estrategias alternativas, según sean los riesgos.

2. *Valoración y reducción del riesgo* En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso. Se dan acciones para reducir el riesgo. Por ejemplo, si existe un riesgo de que los requerimientos sean inadecuados, puede desarrollarse un sistema prototipo.

3. *Desarrollo y validación* Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema. Por ejemplo, la creación de prototipos desechables sería el mejor enfoque de desarrollo, si predominan los riesgos en la interfaz del usuario. Si la principal consideración son los riesgos de seguridad, el desarrollo con base en transformaciones formales sería el proceso más adecuado, entre otros. Si el principal riesgo identificado es la integración de subsistemas, el modelo en cascada sería el mejor modelo de desarrollo a utilizar.

4. *Planeación* El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral. Si se opta por continuar, se trazan los planes para la siguiente fase del proyecto.

La diferencia principal entre el modelo en espiral con otros modelos de proceso de software es su reconocimiento explícito del riesgo. Un ciclo de la espiral comienza por elaborar objetivos como rendimiento y funcionalidad. Luego, se numeran formas alternativas de alcanzar dichos objetivos y de lidiar con las restricciones en cada uno de ellos. Cada alternativa se valora contra cada objetivo y se identifican las fuentes de riesgo del proyecto. El siguiente paso es resolver dichos riesgos, mediante actividades de recopilación de información, como análisis más detallado, creación de prototipos y simulación.

Una vez valorados los riesgos se realiza cierto desarrollo, seguido por una actividad de planeación para la siguiente fase del proceso. De manera informal, el riesgo significa simplemente algo que podría salir mal. Por ejemplo, si la intención es usar un nuevo lenguaje de programación, un riesgo sería que los compiladores disponibles no sean confiables o no produzcan un código-objeto suficientemente eficaz. Los riesgos conducen a propuestas de cambios de software y a problemas de proyecto como exceso en las fechas y el costo, de manera que la minimización del riesgo es una actividad muy importante de administración del proyecto. En el capítulo 22 se tratará la gestión del riesgo, una parte esencial de la administración del proyecto.

2.4 El Proceso Unificado Racional

El Proceso Unificado Racional (RUP, por las siglas de *Rational Unified Process*) (Krutchen, 2003) es un ejemplo de un modelo de proceso moderno que se derivó del trabajo sobre el UML y el proceso asociado de desarrollo de software unificado (Rumbaugh *et al*., 1999; Arlow y Neustadt, 2005). Aquí se incluye una descripción, pues es un buen ejemplo de un modelo de proceso híbrido. Conjunta elementos de todos los modelos de proceso genéricos (sección 2.1), ilustra la buena práctica en especificación y diseño (sección 2.2), y apoya la creación de prototipos y entrega incremental (sección 2.3).

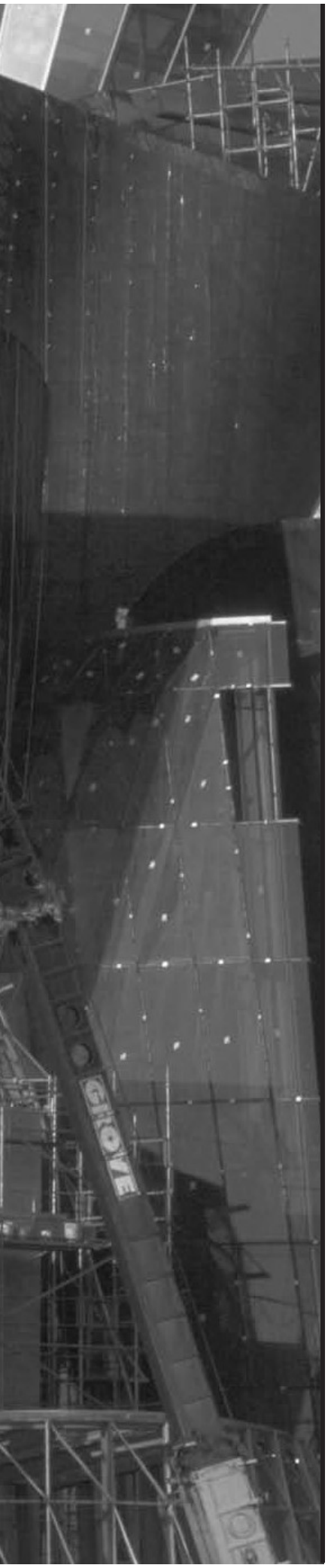
El RUP reconoce que los modelos de proceso convencionales presentan una sola visión del proceso. En contraste, el RUP por lo general se describe desde tres perspectivas:

1. Una perspectiva dinámica que muestra las fases del modelo a través del tiempo.
2. Una perspectiva estática que presenta las actividades del proceso que se establecen.
3. Una perspectiva práctica que sugiere buenas prácticas a usar durante el proceso.

La mayoría de las descripciones del RUP buscan combinar las perspectivas estática y dinámica en un solo diagrama (Krutchen, 2003). Esto hace que el proceso resulte más difícil de entender, por lo que en este texto se usan descripciones separadas de cada una de estas perspectivas.

El RUP es un modelo en fases que identifica cuatro fases discretas en el proceso de software. Sin embargo, a diferencia del modelo en cascada, donde las fases se igualan con actividades del proceso, las fases en el RUP están más estrechamente vinculadas con la empresa que con las preocupaciones técnicas. La figura 2.11 muestra las fases del RUP. Éstas son:

1. *Concepción* La meta de la fase de concepción es establecer un caso empresarial para el sistema. Deben identificarse todas las entidades externas (personas y sistemas)



3

Desarrollo ágil de software

Objetivos

El objetivo de este capítulo es introducirlo a los métodos de desarrollo ágil de software. Al estudiar este capítulo:

- comprenderá las razones de los métodos de desarrollo ágil de software, el manifiesto ágil, así como las diferencias entre el desarrollo ágil y el dirigido por un plan;
- conocerá las prácticas clave en la programación extrema y cómo se relacionan con los principios generales de los métodos ágiles;
- entenderá el enfoque de Scrum para la administración de un proyecto ágil;
- reconocerá los conflictos y problemas de escalar los métodos de desarrollo ágil para el diseño de sistemas de software grandes.

Contenido

Las empresas operan ahora en un entorno global que cambia rápidamente. En ese sentido, deben responder frente a nuevas oportunidades y mercados, al cambio en las condiciones económicas, así como al surgimiento de productos y servicios competitivos. El software es parte de casi todas las operaciones industriales, de modo que el nuevo software se desarrolla rápidamente para aprovechar las actuales oportunidades, con la finalidad de responder ante la amenaza competitiva. En consecuencia, en la actualidad la entrega y el desarrollo rápidos son por lo general el requerimiento fundamental de los sistemas de software. De hecho, muchas empresas están dispuestas a negociar la calidad del software y el compromiso con los requerimientos, para lograr con mayor celeridad la implementación que necesitan del software.

Debido a que dichos negocios funcionan en un entorno cambiante, a menudo es prácticamente imposible derivar un conjunto completo de requerimientos de software estable. Los requerimientos iniciales cambian de modo inevitable, porque los clientes encuentran imposible predecir cómo un sistema afectará sus prácticas operacionales, cómo interactuará con otros sistemas y cuáles operaciones de usuarios se automatizarán. Es posible que sea sólo hasta después de entregar un sistema, y que los usuarios adquieran experiencia con éste, cuando se aclaren los requerimientos reales. Incluso, es probable que debido a factores externos, los requerimientos cambien rápida e impredeciblemente. En tal caso, el software podría ser obsoleto al momento de entregarse.

Los procesos de desarrollo de software que buscan especificar por completo los requerimientos y, luego, diseñar, construir y probar el sistema, no están orientados al desarrollo rápido de software. A medida que los requerimientos cambian, o se descubren problemas en los requerimientos, el diseño o la implementación del sistema tienen que reelaborarse y probarse de nuevo. En consecuencia, un proceso convencional en cascada o uno basado en especificación se prolongan con frecuencia, en tanto que el software final se entrega al cliente mucho después de lo que se especificó originalmente.

En algunos tipos de software, como los sistemas de control críticos para la seguridad, donde es esencial un análisis completo del sistema, resulta oportuno un enfoque basado en un plan. Sin embargo, en un ambiente empresarial de rápido movimiento, esto llega a causar verdaderos problemas. Al momento en que el software esté disponible para su uso, la razón original para su adquisición quizás haya variado tan radicalmente que el software sería inútil a todas luces. Por lo tanto, para sistemas empresariales, son esenciales en particular los procesos de diseño que se enfocan en el desarrollo y la entrega de software rápidos.

Durante algún tiempo, se reconoció la necesidad de desarrollo y de procesos de sistema rápidos que administraran los requerimientos cambiantes. IBM introdujo el desarrollo incremental en la década de 1980 (Mills *et al.*, 1980). La entrada de los llamados lenguajes de cuarta generación, también en la misma década, apoyó la idea del software de desarrollo y entrega rápidos (Martin, 1981). Sin embargo, la noción prosperó realmente a finales de la década de 1990, con el desarrollo de la noción de enfoques ágiles como el DSDM (Stapleton, 1997), Scrum (Schwaber y Beedle, 2001) y la programación extrema (Beck, 1999; Beck, 2000).

Los procesos de desarrollo del software rápido se diseñan para producir rápidamente un software útil. El software no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema. Aun cuando existen muchos enfoques para el desarrollo de software rápido, comparten algunas características fundamentales:

1. Los procesos de especificación, diseño e implementación están entrelazados. No existe una especificación detallada del sistema, y la documentación del diseño se

minimiza o es generada automáticamente por el entorno de programación que se usa para implementar el sistema. El documento de requerimientos del usuario define sólo las características más importantes del sistema.

2. El sistema se desarrolla en diferentes versiones. Los usuarios finales y otros colaboradores del sistema intervienen en la especificación y evaluación de cada versión. Ellos podrían proponer cambios al software y nuevos requerimientos que se implementen en una versión posterior del sistema.

3. Las interfaces de usuario del sistema se desarrollan usando con frecuencia un sistema de elaboración interactivo, que permita que el diseño de la interfaz se cree rápidamente en cuanto se dibujan y colocan iconos en la interfaz. En tal situación, el sistema puede generar una interfaz basada en la Web para un navegador o una interfaz para una plataforma específica, como Microsoft Windows.

Los métodos ágiles son métodos de desarrollo incremental donde los incrementos son mínimos y, por lo general, se crean las nuevas liberaciones del sistema, y cada dos o tres semanas se ponen a disposición de los clientes. Involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes. Minimizan la cantidad de documentación con el uso de comunicaciones informales, en vez de reuniones formales con documentos escritos.

3.1 Métodos ágiles

En la década de 1980 y a inicios de la siguiente, había una visión muy difundida de que la forma más adecuada para lograr un mejor software era mediante una cuidadosa planeación del proyecto, aseguramiento de calidad formalizada, el uso de métodos de análisis y el diseño apoyado por herramientas CASE, así como procesos de desarrollo de software rigurosos y controlados. Esta percepción proviene de la comunidad de ingeniería de software, responsable del desarrollo de grandes sistemas de software de larga duración, como los sistemas aeroespaciales y gubernamentales.

Este software lo desarrollaron grandes equipos que trabajaban para diferentes compañías. A menudo los equipos estaban geográficamente dispersos y laboraban por largos periodos en el software. Un ejemplo de este tipo de software es el sistema de control de una aeronave moderna, que puede tardar hasta 10 años desde la especificación inicial hasta la implementación. Estos enfoques basados en un plan incluyen costos operativos significativos en la planeación, el diseño y la documentación del sistema. Dichos gastos se justifican cuando debe coordinarse el trabajo de múltiples equipos de desarrollo, cuando el sistema es un sistema crítico y cuando numerosas personas intervendrán en el mantenimiento del software a lo largo de su vida.

Sin embargo, cuando este engorroso enfoque de desarrollo basado en la planeación se aplica a sistemas de negocios pequeños y medianos, los costos que se incluyen son tan grandes que dominan el proceso de desarrollo del software. Se invierte más tiempo en diseñar el sistema, que en el desarrollo y la prueba del programa. Conforme cambian los requerimientos del sistema, resulta esencial la reelaboración y, en principio al menos, la especificación y el diseño deben modificarse con el programa.

En la década de 1990 el descontento con estos enfoques engorrosos de la ingeniería de software condujo a algunos desarrolladores de software a proponer nuevos "métodos

ágiles", los cuales permitieron que el equipo de desarrollo se enfocara en el software en lugar del diseño y la documentación. Los métodos ágiles se apoyan universalmente en el enfoque incremental para la especificación, el desarrollo y la entrega del software. Son más adecuados para el diseño de aplicaciones en que los requerimientos del sistema cambian, por lo general, rápidamente durante el proceso de desarrollo. Tienen la intención de entregar con prontitud el software operativo a los clientes, quienes entonces propondrán requerimientos nuevos y variados para incluir en posteriores iteraciones del sistema. Se dirigen a simplificar el proceso burocrático al evitar trabajo con valor dudoso a largo plazo, y a eliminar documentación que quizá nunca se emplee.

La filosofía detrás de los métodos ágiles se refleja en el manifiesto ágil, que acordaron muchos de los desarrolladores líderes de estos métodos. Este manifiesto afirma:

> *Estamos descubriendo mejores formas para desarrollar software, al hacerlo y al ayudar a otros a hacerlo. Gracias a este trabajo llegamos a valorar:*
>
> *A los individuos y las interacciones sobre los procesos y las herramientas*
>
> *Al software operativo sobre la documentación exhaustiva*
>
> *La colaboración con el cliente sobre la negociación del contrato*
>
> *La respuesta al cambio sobre el seguimiento de un plan*
>
> *Esto es, aunque exista valor en los objetos a la derecha, valoraremos más los de la izquierda.*

Probablemente el método ágil más conocido sea la programación extrema (Beck, 1999; Beck, 2000), descrita más adelante en este capítulo. Otros enfoques ágiles incluyen los de Scrum (Cohn, 2009; Schwaber, 2004; Schwaber y Beedle, 2001), de Crystal (Cockburn, 2001; Cockburn, 2004), de desarrollo de software adaptativo (Highsmith, 2000), de DSDM (Stapleton, 1997; Stapleton, 2003) y el desarrollo dirigido por características (Palmer y Felsing, 2002). El éxito de dichos métodos condujo a cierta integración con métodos más tradicionales de desarrollo, basados en el modelado de sistemas, lo cual resulta en la noción de modelado ágil (Ambler y Jeffries, 2002) y ejemplificaciones ágiles del Proceso Racional Unificado (Larman, 2002).

Aunque todos esos métodos ágiles se basan en la noción del desarrollo y la entrega incrementales, proponen diferentes procesos para lograrlo. Sin embargo, comparten una serie de principios, según el manifiesto ágil y, por ende, tienen mucho en común. Dichos principios se muestran en la figura 3.1. Diferentes métodos ágiles ejemplifican esos principios en diversas formas; sin embargo, no se cuenta con espacio suficiente para discutir todos los métodos ágiles. En cambio, este texto se enfoca en dos de los métodos usados más ampliamente: programación extrema (sección 3.3) y de Scrum (sección 3.4).

Los métodos ágiles han tenido mucho éxito para ciertos tipos de desarrollo de sistemas:

1. Desarrollo del producto, donde una compañía de software elabora un producto pequeño o mediano para su venta.
2. Diseño de sistemas a la medida dentro de una organización, donde hay un claro compromiso del cliente por intervenir en el proceso de desarrollo, y donde no existen muchas reglas ni regulaciones externas que afecten el software.

Principio	Descripción
Participación del cliente	Los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.
Entrega incremental	El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.
Personas, no procesos	Tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos.
Adoptar el cambio	Esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios.
Mantener simplicidad	Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.

Figura 3.1 Los principios de los métodos ágiles

Como se analiza en la sección final de este capítulo, el éxito de los métodos ágiles se debe al interés considerable por usar dichos métodos para otros tipos de desarrollo del software. No obstante, dado su enfoque en equipos reducidos firmemente integrados, hay problemas en escalarlos hacia grandes sistemas. También se ha experimentado en el uso de enfoques ágiles para la ingeniería de sistemas críticos (Drobna *et al.*, 2004). Sin embargo, a causa de las necesidades de seguridad, protección y análisis de confiabilidad en los sistemas críticos, los métodos ágiles requieren modificaciones significativas antes de usarse cotidianamente con la ingeniería de sistemas críticos.

En la práctica, los principios que subyacen a los métodos ágiles son a veces difíciles de cumplir:

1. Aunque es atractiva la idea del involucramiento del cliente en el proceso de desarrollo, su éxito radica en tener un cliente que desee y pueda pasar tiempo con el equipo de desarrollo, y éste represente a todos los participantes del sistema. Los representantes del cliente están comúnmente sujetos a otras presiones, así que no intervienen por completo en el desarrollo del software.
2. Quizás algunos miembros del equipo no cuenten con la personalidad adecuada para la participación intensa característica de los métodos ágiles y, en consecuencia, no podrán interactuar adecuadamente con los otros integrantes del equipo.
3. Priorizar los cambios sería extremadamente difícil, sobre todo en sistemas donde existen muchos participantes. Cada uno por lo general ofrece diversas prioridades a diferentes cambios.
4. Mantener la simplicidad requiere trabajo adicional. Bajo la presión de fechas de entrega, es posible que los miembros del equipo carezcan de tiempo para realizar las simplificaciones deseables al sistema.

5. Muchas organizaciones, especialmente las grandes compañías, pasan años cambiando su cultura, de tal modo que los procesos se definan y continúen. Para ellas, resulta difícil moverse hacia un modelo de trabajo donde los procesos sean informales y estén definidos por equipos de desarrollo.

Otro problema que no es técnico, es decir, que consiste en un problema general con el desarrollo y la entrega incremental, ocurre cuando el cliente del sistema acude a una organización externa para el desarrollo del sistema. Por lo general, el documento de requerimientos del software forma parte del contrato entre el cliente y el proveedor. Como la especificación incremental es inherente en los métodos ágiles, quizá sea difícil elaborar contratos para este tipo de desarrollo.

Como resultado, los métodos ágiles deben apoyarse en contratos, en los cuales el cliente pague por el tiempo requerido para el desarrollo del sistema, en vez de hacerlo por el desarrollo de un conjunto específico de requerimientos. En tanto todo marche bien, esto beneficia tanto al cliente como al desarrollador. No obstante, cuando surgen problemas, sería difícil discutir acerca de quién es culpable y quién debería pagar por el tiempo y los recursos adicionales requeridos para solucionar las dificultades.

La mayoría de los libros y ensayos que describen los métodos ágiles y las experiencias con éstos hablan del uso de dichos métodos para el desarrollo de nuevos sistemas. Sin embargo, como se explica en el capítulo 9, una enorme cantidad de esfuerzo en ingeniería de software se usa en el mantenimiento y la evolución de los sistemas de software existentes. Hay sólo un pequeño número de reportes de experiencia sobre el uso de métodos ágiles para el mantenimiento de software (Poole y Huisman, 2001). Se presentan entonces dos preguntas que deberían considerarse junto con los métodos y el mantenimiento ágiles:

1. ¿Los sistemas que se desarrollan usando un enfoque ágil se mantienen, a pesar del énfasis en el proceso de desarrollo de minimizar la documentación formal?
2. ¿Los métodos ágiles pueden usarse con efectividad para evolucionar un sistema como respuesta a requerimientos de cambio por parte del cliente?

Se estima que la documentación formal describe el sistema y, por lo tanto, facilita la comprensión a quienes cambian el sistema. Sin embargo, en la práctica, con frecuencia la documentación formal no se conserva actualizada y, por ende, no refleja con precisión el código del programa. Por esta razón, los apasionados de los métodos ágiles argumentan que escribir esta documentación es una pérdida de tiempo y que la clave para implementar software mantenible es producir un código legible de alta calidad. De esta manera, las prácticas ágiles enfatizan la importancia de escribir un código bien estructurado y destinar el esfuerzo en mejorar el código. En consecuencia, la falta de documentación no debe representar un problema para mantener los sistemas desarrollados con el uso de un enfoque ágil.

No obstante, según la experiencia del autor con el mantenimiento de sistemas, éste sugiere que el documento clave es el documento de requerimientos del sistema, el cual indica al ingeniero de software lo que se supone que debe hacer el sistema. Sin tal conocimiento, es difícil valorar el efecto de los cambios propuestos al sistema. Varios métodos ágiles recopilan los requerimientos de manera informal e incremental, aunque sin crear un documento coherente de requerimientos. A este respecto, es probable

que el uso de métodos ágiles haga más difícil y costoso el mantenimiento posterior del sistema.

Es factible que las prácticas ágiles, usadas en el proceso de mantenimiento en sí, resulten efectivas, ya sea que se utilice o no se utilice un enfoque ágil para el desarrollo del sistema. La entrega incremental, el diseño para el cambio y el mantenimiento de la simplicidad tienen sentido cuando se modifica el software. De hecho, se pensaría tanto en un proceso de desarrollo ágil como en un proceso de evolución del software.

Sin embargo, quizá la principal dificultad luego de entregar el software sea mantener al cliente interviniendo en el proceso. Aunque un cliente justifique la participación de tiempo completo de un representante durante el desarrollo del sistema, esto es menos probable en el mantenimiento, cuando los cambios no son continuos. Es posible que los representantes del cliente pierdan interés en el sistema. En consecuencia, es previsible que se requieran mecanismos alternativos, como las propuestas de cambio, descritas en el capítulo 25, para establecer los nuevos requerimientos del sistema.

El otro problema potencial tiene que ver con mantener la continuidad del equipo de desarrollo. Los métodos ágiles se apoyan en aquellos miembros del equipo que comprenden los aspectos del sistema sin que deban consultar la documentación. Si se separa un equipo de desarrollo ágil, entonces se pierde este conocimiento implícito y es difícil que los nuevos miembros del equipo acumulen la misma percepción del sistema y sus componentes.

Quienes apoyan los métodos ágiles han creído fielmente en la promoción de su uso y tienden a pasar por alto sus limitaciones. Esto alienta una respuesta igualmente extrema que, para el autor, exagera los problemas con este enfoque (Stephens y Rosenberg, 2003). Críticos más razonables como DeMarco y Boehm (DeMarco y Boehm, 2002) destacan tanto las ventajas como las desventajas de los métodos ágiles. Proponen un enfoque híbrido donde los métodos ágiles que incorporan algunas técnicas del desarrollo dirigido por un plan son la mejor forma de avanzar.

3.2 Desarrollo dirigido por un plan y desarrollo ágil

Los enfoques ágiles en el desarrollo de software consideran el diseño y la implementación como las actividades centrales en el proceso del software. Incorporan otras actividades en el diseño y la implementación, como la adquisición de requerimientos y pruebas. En contraste, un enfoque basado en un plan para la ingeniería de software identifica etapas separadas en el proceso de software con salidas asociadas a cada etapa. Las salidas de una etapa se usan como base para planear la siguiente actividad del proceso. La figura 3.2 muestra las distinciones entre los enfoques ágil y el basado en un plan para la especificación de sistemas.

En un enfoque basado en un plan, la iteración ocurre dentro de las actividades con documentos formales usados para comunicarse entre etapas del proceso. Por ejemplo, los requerimientos evolucionarán y, a final de cuentas, se producirá una especificación de aquéllos. Esto entonces es una entrada al proceso de diseño y la implementación. En un enfoque ágil, la iteración ocurre a través de las actividades. Por lo tanto, los requerimientos y el diseño se desarrollan en conjunto, no por separado.

Un proceso de software dirigido por un plan soporta el desarrollo y la entrega incrementales. Es perfectamente factible asignar requerimientos y planear tanto la fase de diseño y desarrollo como una serie de incrementos. Un proceso ágil no está inevitable-

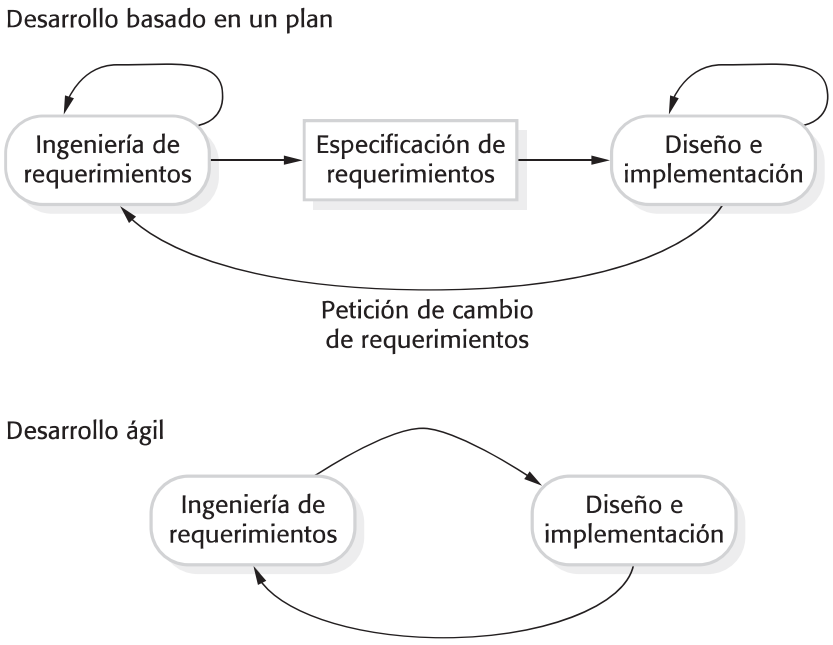


Figura 3.2
Especificación ágil y dirigida por un plan

mente enfocado al código y puede producir cierta documentación de diseño. Como se expone en la siguiente sección, el equipo de desarrollo ágil puede incluir un "pico" de documentación donde, en vez de producir una nueva versión de un sistema, el equipo generará documentación del sistema.

De hecho, la mayoría de los proyectos de software incluyen prácticas de los enfoques ágil y basado en un plan. Para decidir sobre el equilibrio entre un enfoque basado en un plan y uno ágil, se deben responder algunas preguntas técnicas, humanas y organizacionales:

1. ¿Es importante tener una especificación y un diseño muy detallados antes de dirigirse a la implementación? Siendo así, probablemente usted tenga que usar un enfoque basado en un plan.
2. ¿Es práctica una estrategia de entrega incremental, donde se dé el software a los clientes y se obtenga así una rápida retroalimentación de ellos? De ser el caso, considere el uso de métodos ágiles.
3. ¿Qué tan grande es el sistema que se desarrollará? Los métodos ágiles son más efectivos cuando el sistema logra diseñarse con un pequeño equipo asignado que se comunique de manera informal. Esto sería imposible para los grandes sistemas que precisan equipos de desarrollo más amplios, de manera que tal vez se utilice un enfoque basado en un plan.
4. ¿Qué tipo de sistema se desarrollará? Los sistemas que demandan mucho análisis antes de la implementación (por ejemplo, sistema en tiempo real con requerimientos de temporización compleja), por lo general, necesitan un diseño bastante detallado para realizar este análisis. En tales circunstancias, quizá sea mejor un enfoque basado en un plan.
5. ¿Cuál es el tiempo de vida que se espera del sistema? Los sistemas con lapsos de vida prolongados podrían requerir más documentación de diseño, para comunicar al equipo de apoyo los propósitos originales de los desarrolladores del sistema. Sin embargo,

los defensores de los métodos ágiles argumentan acertadamente que con frecuencia la documentación no se conserva actualizada, ni se usa mucho para el mantenimiento del sistema a largo plazo.

6. ¿Qué tecnologías se hallan disponibles para apoyar el desarrollo del sistema? Los métodos ágiles se auxilian a menudo de buenas herramientas para seguir la pista de un diseño en evolución. Si se desarrolla un sistema con un IDE sin contar con buenas herramientas para visualización y análisis de programas, entonces posiblemente se requiera más documentación de diseño.

7. ¿Cómo está organizado el equipo de desarrollo? Si el equipo de desarrollo está distribuido, o si parte del desarrollo se subcontrata, entonces tal vez se requiera elaborar documentos de diseño para comunicarse a través de los equipos de desarrollo. Quizá se necesite planear por adelantado cuáles son.

8. ¿Existen problemas culturales que afecten el desarrollo del sistema? Las organizaciones de ingeniería tradicionales presentan una cultura de desarrollo basada en un plan, pues es una norma en ingeniería. Esto requiere comúnmente una amplia documentación de diseño, en vez del conocimiento informal que se utiliza en los procesos ágiles.

9. ¿Qué tan buenos son los diseñadores y programadores en el equipo de desarrollo? Se argumenta en ocasiones que los métodos ágiles requieren niveles de habilidad superiores a los enfoques basados en un plan, en que los programadores simplemente traducen un diseño detallado en un código. Si usted tiene un equipo con niveles de habilidad relativamente bajos, es probable que necesite del mejor personal para desarrollar el diseño, siendo otros los responsables de la programación.

10. ¿El sistema está sujeto a regulación externa? Si un regulador externo tiene que aprobar el sistema (por ejemplo, la Agencia de Aviación Federal [FAA] estadounidense aprueba el software que es crítico para la operación de una aeronave), entonces, tal vez se le requerirá documentación detallada como parte del sistema de seguridad.

En realidad, es irrelevante el conflicto sobre si un proyecto puede considerarse dirigido por un plan o ágil. A final de cuentas, la principal inquietud de los compradores de un sistema de software es si cuentan o no con un sistema de software ejecutable, que cubra sus necesidades y realice funciones útiles para el usuario de manera individual o dentro de una organización. En la práctica, muchas compañías que afirman haber usado métodos ágiles adoptaron algunas habilidades ágiles y las integraron con sus procesos dirigidos por un plan.

3.3 Programación extrema

La programación extrema (XP) es quizás el método ágil mejor conocido y más ampliamente usado. El nombre lo acuñó Beck (2000) debido a que el enfoque se desarrolló llevando a niveles "extremos" las prácticas reconocidas, como el desarrollo iterativo. Por ejemplo, en la XP muchas versiones actuales de un sistema pueden desarrollarse mediante diferentes programadores, integrarse y ponerse a prueba en un solo día.

la mitad del código que dos individuos que trabajen solos. Hay varios estudios de la productividad de los programadores en pares con resultados mixtos. Al usar estudiantes voluntarios, Williams y sus colaboradores (Cockburn y Williams, 2001; Williams *et al.*, 2000) descubrieron que la productividad con la programación en pares es comparable con la de dos individuos que trabajan de manera independiente. Las razones sugeridas son que los pares discuten el software antes de desarrollarlo, de modo que probablemente tengan menos salidas en falso y menos rediseño. Más aún, el número de errores que se evitan por la inspección informal es tal que se emplea menos tiempo en reparar los *bugs* descubiertos durante el proceso de pruebas.

Sin embargo, los estudios con programadores más experimentados (Arisholm *et al.*, 2007; Parrish *et al.*, 2004) no replican dichos resultados. Hallaron que había una pérdida de productividad significativa comparada con dos programadores que trabajan individualmente. Hubo algunos beneficios de calidad, pero no compensaron por completo los costos de la programación en pares. No obstante, el intercambio de conocimiento que ocurre durante la programación en pares es muy importante, pues reduce los riesgos globales de un proyecto cuando salen miembros del equipo. En sí mismo, esto hace que la programación de este tipo valga la pena.

3.4 Administración de un proyecto ágil

La responsabilidad principal de los administradores del proyecto de software es dirigir el proyecto, de modo que el software se entregue a tiempo y con el presupuesto planeado para ello. Supervisan el trabajo de los ingenieros de software y monitorizan el avance en el desarrollo del software.

El enfoque estándar de la administración de proyectos es el basado en un plan. Como se estudia en el capítulo 23, los administradores se apoyan en un plan para el proyecto que muestra lo que se debe entregar y cuándo, así como quién trabajará en el desarrollo de los entregables del proyecto. Un enfoque basado en un plan requiere en realidad que un administrador tenga una visión equilibrada de todo lo que debe diseñarse y de los procesos de desarrollo. Sin embargo, no funciona bien con los métodos ágiles, donde los requerimientos se desarrollan incrementalmente, donde el software se entrega en rápidos incrementos cortos, y donde los cambios a los requerimientos y el software son la norma.

Como cualquier otro proceso de diseño de software profesional, el desarrollo ágil tiene que administrarse de tal modo que se busque el mejor uso del tiempo y de los recursos disponibles para el equipo. Esto requiere un enfoque diferente a la administración del proyecto, que se adapte al desarrollo incremental y a las fortalezas particulares de los métodos ágiles.

Aunque el enfoque de Scrum (Schwaber, 2004; Schwaber y Beedle, 2001) es un método ágil general, su enfoque está en la administración iterativa del desarrollo, y no en enfoques técnicos específicos para la ingeniería de software ágil. La figura 3.8 representa un diagrama del proceso de administración de Scrum. Este proceso no prescribe el uso de prácticas de programación, como la programación en pares y el desarrollo de primera prueba. Por lo tanto, puede usarse con enfoques ágiles más técnicos, como XP, para ofrecer al proyecto un marco administrativo.

Existen tres fases con Scrum. La primera es la planeación del bosquejo, donde se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software.

lugar de ello, el "maestro de Scrum" es el facilitador que ordena las reuniones diarias, rastrea el atraso del trabajo a realizar, registra las decisiones, mide el progreso del atraso, y se comunica con los clientes y administradores fuera del equipo.

Todo el equipo asiste a las reuniones diarias, que en ocasiones son reuniones en las que los participantes no se sientan, para hacerlas breves y enfocadas. Durante la reunión, todos los miembros del equipo comparten información, describen sus avances desde la última reunión, los problemas que han surgido y los planes del día siguiente. Ello significa que todos en el equipo conocen lo que acontece y, si surgen problemas, replantean el trabajo en el corto plazo para enfrentarlo. Todos participan en esta planeación; no hay dirección descendente desde el maestro de Scrum.

En la Web existen muchos reportes anecdóticos del uso exitoso del Scrum. Rising y Janoff (2000) discuten su uso exitoso en un entorno de desarrollo de software para telecomunicaciones y mencionan sus ventajas del modo siguiente:

1. El producto se desglosa en un conjunto de piezas manejables y comprensibles.
2. Los requerimientos inestables no retrasan el progreso.
3. Todo el equipo tiene conocimiento de todo y, en consecuencia, se mejora la comunicación entre el equipo.
4. Los clientes observan la entrega a tiempo de los incrementos y obtienen retroalimentación sobre cómo funciona el producto.
5. Se establece la confianza entre clientes y desarrolladores, a la vez que se crea una cultura positiva donde todos esperan el triunfo del proyecto.

Scrum, como originalmente se designó, tenía la intención de usarse con equipos coasignados, donde todos los miembros del equipo pudieran congregarse a diario en reuniones breves. Sin embargo, mucho del desarrollo del software implica ahora equipos distribuidos con miembros del equipo ubicados en diferentes lugares alrededor del mundo. En consecuencia, hay varios experimentos en marcha con la finalidad de desarrollar el Scrum para entornos de desarrollo distribuidos (Smits y Pshigoda, 2007; Sutherland *et al*., 2007).

3.5 Escalamiento de métodos ágiles

Los métodos ágiles se desarrollaron para usarse en pequeños equipos de programación, que podían trabajar juntos en la misma habitación y comunicarse de manera informal. Por lo tanto, los métodos ágiles se emplean principalmente para el diseño de sistemas pequeños y medianos. Desde luego, la necesidad de entrega más rápida del software, que es más adecuada para las necesidades del cliente, se aplica también a sistemas más grandes. Por consiguiente, hay un enorme interés en escalar los métodos ágiles para enfrentar los sistemas de mayor dimensión, desarrollados por grandes organizaciones.

Denning y sus colaboradores (2008) argumentan que la única forma de evitar los problemas comunes de la ingeniería de software, como los sistemas que no cubren las necesidades del cliente y exceden el presupuesto, es encontrar maneras de hacer que los métodos ágiles funcionen para grandes sistemas. Leffingwell (2007) discute cuáles prácticas ágiles se escalan al desarrollo de grandes sistemas. Moore y Spens (2008) reportan su experiencia al usar un enfoque ágil para desarrollar un gran sistema médico, con 300 desarrolladores que trabajaban en equipos distribuidos geográficamente.

El desarrollo de grandes sistemas de software difiere en algunas formas del desarrollo de sistemas pequeños:

1. Los grandes sistemas son, por lo general, colecciones de sistemas separados en comunicación, donde equipos separados desarrollan cada sistema. Dichos equipos trabajan con frecuencia en diferentes lugares, en ocasiones en otras zonas horarias. Es prácticamente imposible que cada equipo tenga una visión de todo el sistema. En consecuencia, sus prioridades son generalmente completar la parte del sistema sin considerar asuntos de los sistemas más amplios.
2. Los grandes sistemas son "sistemas abandonados" (Hopkins y Jenkins, 2008); esto es, incluyen e interactúan con algunos sistemas existentes. Muchos de los requerimientos del sistema se interesan por su interacción y, por lo tanto, en realidad no se prestan a la flexibilidad y al desarrollo incremental. Aquí también podrían ser relevantes los conflictos políticos y a menudo la solución más sencilla a un problema es cambiar un sistema existente. Sin embargo, esto requiere negociar con los administradores de dicho sistema para convencerlos de que los cambios pueden implementarse sin riesgo para la operación del sistema.
3. Donde muchos sistemas se integran para crear un solo sistema, una fracción significativa del desarrollo se ocupa en la configuración del sistema, y no en el desarrollo del código original. Esto no necesariamente es compatible con el desarrollo incremental y la integración frecuente del sistema.
4. Los grandes sistemas y sus procesos de desarrollo por lo común están restringidos por reglas y regulaciones externas, que limitan la forma en que pueden desarrollarse, lo cual requiere de ciertos tipos de documentación del sistema que se va a producir, etcétera.
5. Los grandes sistemas tienen un tiempo prolongado de adquisición y desarrollo. Es difícil mantener equipos coherentes que conozcan el sistema durante dicho periodo, pues resulta inevitable que algunas personas se cambien a otros empleos y proyectos.
6. Los grandes sistemas tienen por lo general un conjunto variado de participantes. Por ejemplo, cuando enfermeras y administradores son los usuarios finales de un sistema médico, el personal médico ejecutivo, los administradores del hospital, etcétera, también son participantes en el sistema. En realidad es imposible involucrar a todos estos participantes en el proceso de desarrollo.

Existen dos perspectivas en el escalamiento de los métodos ágiles:

1. Una perspectiva de "expansión" (*scaling up*), que se interesa por el uso de dichos métodos para el desarrollo de grandes sistemas de software que no logran desarrollarse con equipos pequeños.

2. Una perspectiva de "ampliación" (*scaling out*), que se interesa por que los métodos ágiles se introduzcan en una organización grande con muchos años de experiencia en el desarrollo de software.

Los métodos ágiles tienen que adaptarse para enfrentar la ingeniería de los sistemas grandes. Leffingwell (2007) explica que es esencial mantener los fundamentos de los métodos ágiles: planeación flexible, liberación frecuente del sistema, integración continua, desarrollo dirigido por pruebas y buena comunicación del equipo. El autor considera que las siguientes adaptaciones son críticas y deben introducirse:

1. Para el desarrollo de grandes sistemas no es posible enfocarse sólo en el código del sistema. Es necesario hacer más diseño frontal y documentación del sistema. Debe diseñarse la arquitectura de software y producirse documentación para describir los aspectos críticos del sistema, como esquemas de bases de datos, división del trabajo entre los equipos, etcétera.
2. Tienen que diseñarse y usarse mecanismos de comunicación entre equipos. Esto debe incluir llamadas telefónicas regulares, videoconferencias entre los miembros del equipo y frecuentes reuniones electrónicas breves, para que los equipos se actualicen mutuamente del avance. Hay que ofrecer varios canales de comunicación (como correo electrónico, mensajería instantánea, wikis y sistemas de redes sociales) para facilitar las comunicaciones.
3. La integración continua, donde todo el sistema se construya cada vez que un desarrollador verifica un cambio, es prácticamente imposible cuando muchos programas separados deben integrarse para crear el sistema. Sin embargo, resulta esencial mantener construcciones del sistema frecuentes y liberaciones del sistema regulares. Esto podría significar la introducción de nuevas herramientas de gestión de configuración que soporten el desarrollo de software por parte de múltiples equipos.

Las compañías de software pequeñas que desarrollan productos de software están entre quienes adoptan con más entusiasmo los métodos ágiles. Dichas compañías no están restringidas por burocracias organizacionales o estándares de procesos, y son capaces de cambiar rápidamente para acoger nuevas ideas. Desde luego, las compañías más grandes también experimentan en proyectos específicos con los métodos ágiles; sin embargo, para ellas es mucho más difícil "ampliar" dichos métodos en toda la organización. Lindvall y sus colaboradores (2004) analizan algunos de los problemas al escalar los métodos ágiles en cuatro grandes compañías tecnológicas.

Es difícil introducir los métodos ágiles en las grandes compañías por algunas razones:

1. Los gerentes del proyecto carecen de experiencia con los métodos ágiles; pueden ser reticentes para aceptar el riesgo de un nuevo enfoque, pues no saben cómo afectará sus proyectos particulares.
2. Las grandes organizaciones tienen a menudo procedimientos y estándares de calidad que se espera sigan todos los proyectos y, dada su naturaleza burocrática, es probable que sean incompatibles con los métodos ágiles. En ocasiones, reciben apoyo de herramientas de software (por ejemplo, herramientas de gestión de requerimientos), y el uso de dichas herramientas es obligatorio para todos los proyectos.

3. Los métodos ágiles parecen funcionar mejor cuando los miembros del equipo tienen un nivel de habilidad relativamente elevado. Sin embargo, dentro de grandes organizaciones, probablemente haya una amplia gama de habilidades y destrezas, y los individuos con niveles de habilidad inferiores quizá no sean miembros de equipos efectivos en los procesos ágiles.

4. Quizás haya resistencia cultural contra los métodos ágiles, en especial en aquellas organizaciones con una larga historia de uso de procesos convencionales de ingeniería de sistemas.

Los procedimientos de gestión de cambio y de pruebas son ejemplos de procedimientos de la compañía que podrían no ser compatibles con los métodos ágiles. La administración del cambio es el proceso que controla los cambios a un sistema, de modo que el efecto de los cambios sea predecible y se controlen los costos. Antes de realizarse, todos los cambios deben aprobarse y esto entra en conflicto con la noción de refactorización. En XP, cualquier desarrollador puede mejorar cualquier código sin conseguir aprobación externa. Para sistemas grandes, también existen estándares de pruebas, donde una construcción del sistema se envía a un equipo de pruebas externo. Esto entraría en conflicto con los enfoques de primera prueba y prueba frecuente utilizados en XP.

Introducir y sostener el uso de los métodos ágiles a lo largo de una organización grande es un proceso de cambio cultural. El cambio cultural tarda mucho tiempo en implementarse y a menudo requiere un cambio de administración antes de llevarse a cabo. Las compañías que deseen usar métodos ágiles necesitan promotores para alentar el cambio. Tienen que dedicar recursos significativos para el proceso del cambio. Al momento de escribir este texto, unas cuantas compañías clasificadas como grandes han realizado una transición exitosa al desarrollo ágil a lo largo de la organización.

PUNTOS CLAVE

- Los métodos ágiles son métodos de desarrollo incremental que se enfocan en el diseño rápido, liberaciones frecuentes del software, reducción de gastos en el proceso y producción de código de alta calidad. Hacen que el cliente intervenga directamente en el proceso de desarrollo.
- La decisión acerca de si se usa un enfoque de desarrollo ágil o uno basado en un plan depende del tipo de software que se va a elaborar, las capacidades del equipo de desarrollo y la cultura de la compañía que diseña el sistema.
- La programación extrema es un método ágil bien conocido que integra un rango de buenas prácticas de programación, como las liberaciones frecuentes del software, el mejoramiento continuo del software y la participación del cliente en el equipo de desarrollo.
- Una fortaleza particular de la programación extrema, antes de crear una característica del programa, es el desarrollo de pruebas automatizadas. Todas las pruebas deben ejecutarse con éxito cuando un incremento se integra en un sistema.