2.3 Modelos de proceso prescriptivo

Los modelos de proceso prescriptivo fueron propuestos originalmente para poner orden en el caos del desarrollo de software. La historia indica que estos modelos tradicionales han dado cierta estructura útil al trabajo de ingeniería de software y que constituyen un mapa razonablemente eficaz para los equipos de software. Sin embargo, el trabajo de ingeniería de software y el producto que genera siguen "al borde del caos".

Cita:

"Si el proceso está bien, los resultados cuidarán de sí mismos."

Takashi Osada

En un artículo intrigante sobre la extraña relación entre el orden y el caos en el mundo del software, Nogueira y sus colegas [Nog00] afirman lo siguiente:

> El borde del caos se define como "el estado natural entre el orden y el caos, un compromiso grande entre la estructura y la sorpresa" [Kau95]. El borde del caos se visualiza como un estado inestable y parcialmente estructurado [...] Es inestable debido a que se ve atraído constantemente hacia el caos o hacia el orden absoluto.
>
> Tenemos la tendencia de pensar que el orden es el estado ideal de la naturaleza. Esto podría ser un error [...] las investigaciones apoyan la teoría de que la operación que se aleja del equilibrio genera creatividad, procesos autoorganizados y rendimientos crecientes [Roo96]. El orden absoluto significa ausencia de variabilidad, que podría ser una ventaja en los ambientes impredecibles. El cambio ocurre cuando hay cierta estructura que permita que el cambio pueda organizarse, pero que no sea tan rígida como para que no pueda suceder. Por otro lado, demasiado caos hace imposible la coordinación y la coherencia. La falta de estructura no siempre significa desorden.

PUNTO CLAVE

Los modelos de proceso prescriptivo definen un conjunto prescrito de elementos del proceso y un flujo predecible para el trabajo del proceso.

Las implicaciones filosóficas de este argumento son significativas para la ingeniería de software. Si los modelos de proceso prescriptivo[5] buscan generar estructura y orden, ¿son inapropiados para el mundo del software, que se basa en el cambio? Pero si rechazamos los modelos de proceso tradicional (y el orden que implican) y los reemplazamos con algo menos estructurado, ¿hacemos imposible la coordinación y coherencia en el trabajo de software?

No hay respuestas fáciles para estas preguntas, pero existen alternativas disponibles para los ingenieros de software. En las secciones que siguen se estudia el enfoque de proceso prescriptivo en el que los temas dominantes son el orden y la consistencia del proyecto. El autor los llama "prescriptivos" porque prescriben un conjunto de elementos del proceso: actividades estructurales, acciones de ingeniería de software, tareas, productos del trabajo, aseguramiento de la calidad y mecanismos de control del cambio para cada proyecto. Cada modelo del proceso también prescribe un flujo del proceso (también llamado *flujo de trabajo*), es decir, la manera en la que los elementos del proceso se relacionan entre sí.

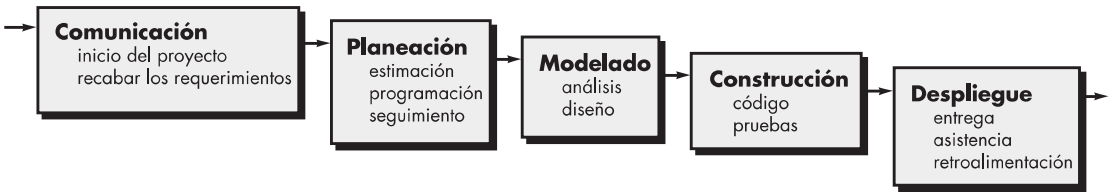
Todos los modelos del proceso del software pueden incluir las actividades estructurales generales descritas en el capítulo 1, pero cada una pone distinto énfasis en ellas y define en forma diferente el flujo de proceso que invoca cada actividad estructural (así como acciones y tareas de ingeniería de software).

2.3.1 Modelo de la cascada

Hay veces en las que los requerimientos para cierto problema se comprenden bien: cuando el trabajo desde la **comunicación** hasta el **despliegue** fluye en forma razonablemente lineal. Esta situación se encuentra en ocasiones cuando deben hacerse adaptaciones o mejoras bien definidas a un sistema ya existente (por ejemplo, una adaptación para software de contabilidad que es obligatorio hacer debido a cambios en las regulaciones gubernamentales). También ocurre en cierto número limitado de nuevos esfuerzos de desarrollo, pero sólo cuando los requerimientos están bien definidos y tienen una estabilidad razonable.

5 Los modelos de proceso prescriptivo en ocasiones son denominados modelos de proceso "tradicional".

FIGURA 2.3 Modelo de la cascada



El *modelo de la cascada*, a veces llamado *ciclo de vida clásico*, sugiere un enfoque sistemático y secuencial[6] para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue, para concluir con el apoyo del software terminado (véase la figura 2.3).

PUNTO CLAVE

El modelo en V ilustra la forma en la que se asocian las acciones de verificación y validación con las primeras acciones de ingeniería.

Una variante de la representación del modelo de la cascada se denomina *modelo en V*. En la figura 2.4 se ilustra el modelo en V [Buc99], donde se aprecia la relación entre las acciones para el aseguramiento de la calidad y aquellas asociadas con la comunicación, modelado y construcción temprana. A medida que el equipo de software avanza hacia abajo desde el lado izquierdo de la V, los requerimientos básicos del problema mejoran hacia representaciones técnicas cada vez más detalladas del problema y de su solución. Una vez que se ha generado el código, el equipo sube por el lado derecho de la V, y en esencia ejecuta una serie de pruebas (acciones para asegurar la calidad) que validan cada uno de los modelos creados cuando el equipo fue hacia abajo por el lado izquierdo.[7] En realidad, no hay diferencias fundamentales entre el ciclo de vida clásico y el modelo en V. Este último proporciona una forma de visualizar el modo de aplicación de las acciones de verificación y validación al trabajo de ingeniería inicial.

El modelo de la cascada es el paradigma más antiguo de la ingeniería de software. Sin embargo, en las últimas tres décadas, las críticas hechas al modelo han ocasionado que incluso sus defensores más obstinados cuestionen su eficacia [Han95]. Entre los problemas que en ocasiones surgen al aplicar el modelo de la cascada se encuentran los siguientes:

? ¿Por qué a veces falla el modelo de la cascada?

1. Es raro que los proyectos reales sigan el flujo secuencial propuesto por el modelo. Aunque el modelo lineal acepta repeticiones, lo hace en forma indirecta. Como resultado, los cambios generan confusión conforme el equipo del proyecto avanza.
2. A menudo, es difícil para el cliente enunciar en forma explícita todos los requerimientos. El modelo de la cascada necesita que se haga y tiene dificultades para aceptar la incertidumbre natural que existe al principio de muchos proyectos.
3. El cliente debe tener paciencia. No se dispondrá de una versión funcional del(de los) programa(s) hasta que el proyecto esté muy avanzado. Un error grande sería desastroso si se detectara hasta revisar el programa en funcionamiento.

Cita:

"Con demasiada frecuencia, el trabajo de software sigue la primera ley del ciclismo: no importa hacia dónde te dirijas, vas hacia arriba y contra el viento."

Anónimo

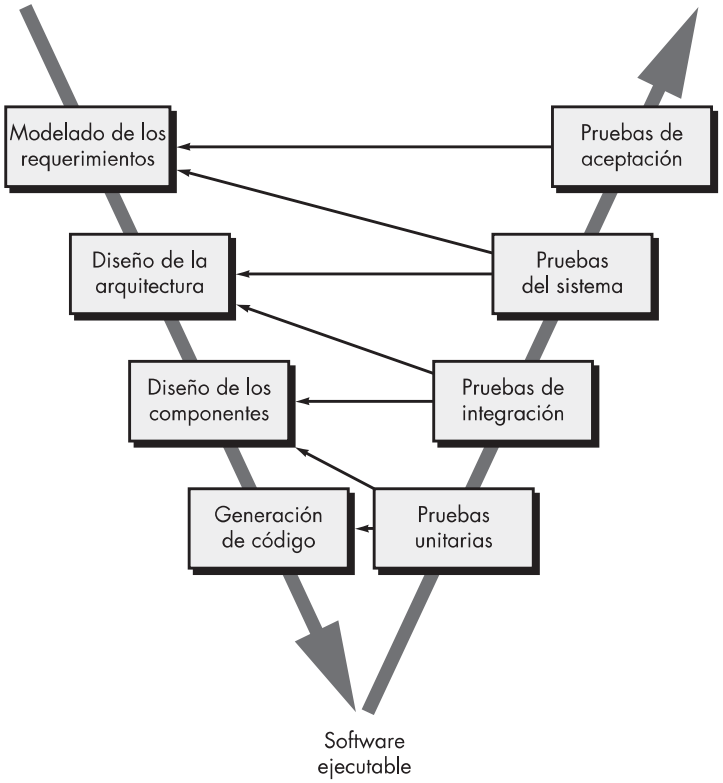
En un análisis interesante de proyectos reales, Bradac [Bra94] encontró que la naturaleza lineal del ciclo de vida clásico llega a "estados de bloqueo" en los que ciertos miembros del equipo de proyecto deben esperar a otros a fin de terminar tareas interdependientes. En realidad, ¡el tiempo de espera llega a superar al dedicado al trabajo productivo! Los estados de bloqueo tienden a ocurrir más al principio y al final de un proceso secuencial lineal.

Hoy en día, el trabajo de software es acelerado y está sujeto a una corriente sin fin de cambios (en las características, funciones y contenido de información). El modelo de la cascada suele ser

6 Aunque el modelo de la cascada propuesto originalmente por Winston Royce [Roy70] prevé los "bucles de retroalimentación", la gran mayoría de organizaciones que aplican este modelo de proceso lo tratan como si fuera estrictamente lineal.

7 En la parte 3 del libro se estudian con detalle las acciones de aseguramiento de la calidad.

FIGURA 2.4
El modelo en V



inapropiado para ese tipo de labor. No obstante, sirve como un modelo de proceso útil en situaciones en las que los requerimientos son fijos y el trabajo avanza en forma lineal hacia el final.

2.3.2 Modelos de proceso incremental

PUNTO CLAVE

El modelo incremental ejecuta una serie de avances, llamados incrementos, que en forma progresiva dan más funcionalidad al cliente conforme se le entrega cada incremento.

Hay muchas situaciones en las que los requerimientos iniciales del software están razonablemente bien definidos, pero el alcance general del esfuerzo de desarrollo imposibilita un proceso lineal. Además, tal vez haya una necesidad imperiosa de dar rápidamente cierta funcionalidad limitada de software a los usuarios y aumentarla en las entregas posteriores de software. En tales casos, se elige un modelo de proceso diseñado para producir el software en incrementos.

El modelo *incremental* combina elementos de los flujos de proceso lineal y paralelo estudiados en la sección 2.1. En relación con la figura 2.5, el modelo incremental aplica secuencias lineales en forma escalonada a medida que avanza el calendario de actividades. Cada secuencia lineal produce "incrementos" de software susceptibles de entregarse [McD93] de manera parecida a los incrementos producidos en un flujo de proceso evolutivo (sección 2.3.3).

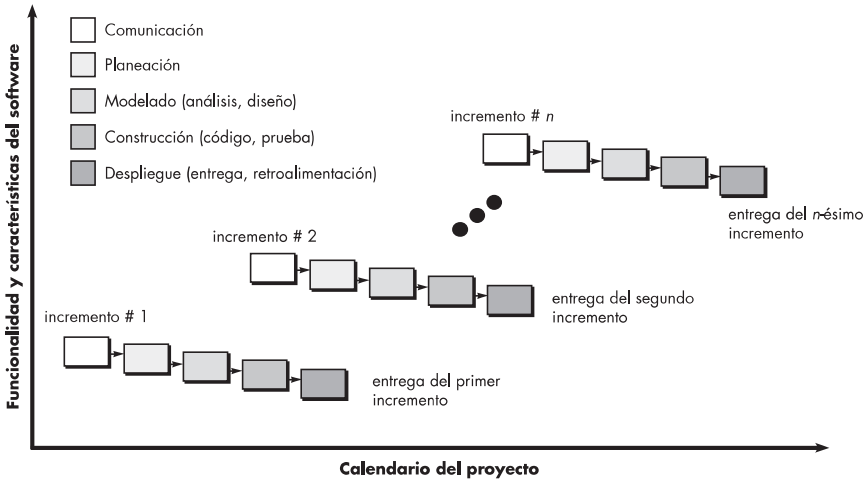
CONSEJO

Su cliente solicita la entrega para una fecha que es imposible de cumplir. Sugiera entregar uno o más incrementos en la fecha que pide, y el resto del software (incrementos adicionales) en un momento posterior.

Por ejemplo, un software para procesar textos que se elabore con el paradigma incremental quizá entregue en el primer incremento las funciones básicas de administración de archivos, edición y producción del documento; en el segundo dará herramientas más sofisticadas de edición y producción de documentos; en el tercero habrá separación de palabras y revisión de la ortografía; y en el cuarto se proporcionará la capacidad para dar formato avanzado a las páginas. Debe observarse que el flujo de proceso para cualquier incremento puede incorporar el paradigma del prototipo.

Cuando se utiliza un modelo incremental, es frecuente que el primer incremento sea el *producto fundamental*. Es decir, se abordan los requerimientos básicos, pero no se proporcionan muchas características suplementarias (algunas conocidas y otras no). El cliente usa el producto fundamental (o lo somete a una evaluación detallada). Como resultado del uso y/o evaluación,

FIGURA 2.5
El modelo incremental



se desarrolla un plan para el incremento que sigue. El plan incluye la modificación del producto fundamental para cumplir mejor las necesidades del cliente, así como la entrega de características adicionales y más funcionalidad. Este proceso se repite después de entregar cada incremento, hasta terminar el producto final.

El modelo de proceso incremental se centra en que en cada incremento se entrega un producto que ya opera. Los primeros incrementos son versiones desnudas del producto final, pero proporcionan capacidad que sirve al usuario y también le dan una plataforma de evaluación.[8]

El desarrollo incremental es útil en particular cuando no se dispone de personal para la implementación completa del proyecto en el plazo establecido por el negocio. Los primeros incrementos se desarrollan con pocos trabajadores. Si el producto básico es bien recibido, entonces se agrega más personal (si se requiere) para que labore en el siguiente incremento. Además, los incrementos se planean para administrar riesgos técnicos. Por ejemplo, un sistema grande tal vez requiera que se disponga de hardware nuevo que se encuentre en desarrollo y cuya fecha de entrega sea incierta. En este caso, tal vez sea posible planear los primeros incrementos de forma que eviten el uso de dicho hardware, y así proporcionar una funcionalidad parcial a los usuarios finales sin un retraso importante.

2.3.3 Modelos de proceso evolutivo

PUNTO CLAVE
El modelo del proceso evolutivo genera en cada iteración una versión final cada vez más completa del software.

El software, como todos los sistemas complejos, evoluciona en el tiempo. Es frecuente que los requerimientos del negocio y del producto cambien conforme avanza el desarrollo, lo que hace que no sea realista trazar una trayectoria rectilínea hacia el producto final; los plazos apretados del mercado hacen que sea imposible la terminación de un software perfecto, pero debe lanzarse una versión limitada a fin de aliviar la presión de la competencia o del negocio; se comprende bien el conjunto de requerimientos o el producto básico, pero los detalles del producto o extensiones del sistema aún están por definirse. En estas situaciones y otras parecidas se necesita un modelo de proceso diseñado explícitamente para adaptarse a un producto que evoluciona con el tiempo.

Los modelos evolutivos son iterativos. Se caracterizan por la manera en la que permiten desarrollar versiones cada vez más completas del software. En los párrafos que siguen se presentan dos modelos comunes de proceso evolutivo.

8 Es importante observar que para todos los modelos de proceso "ágiles" que se estudian en el capítulo 3 también se usa la filosofía incremental.

Cita:

"Planee para lanzar uno. De todos modos hará eso. Su única elección es si tratará de vender a sus clientes lo que lanzó."

Frederick P. Brooks

Hacer prototipos. Es frecuente que un cliente defina un conjunto de objetivos generales para el software, pero que no identifique los requerimientos detallados para las funciones y características. En otros casos, el desarrollador tal vez no esté seguro de la eficiencia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debe adoptar la interacción entre el humano y la máquina. En estas situaciones, y muchas otras, el *paradigma de hacer prototipos* tal vez ofrezca el mejor enfoque.

Aunque es posible hacer prototipos como un modelo de proceso aislado, es más común usarlo como una técnica que puede implementarse en el contexto de cualquiera de los modelos de proceso descritos en este capítulo. Sin importar la manera en la que se aplique, el paradigma de hacer prototipos le ayudará a usted y a otros participantes a mejorar la comprensión de lo que hay que elaborar cuando los requerimientos no están claros.

CONSEJO

Cuando su cliente tiene una necesidad legítima, pero ignora los detalles, como primer paso desarrolle un prototipo.

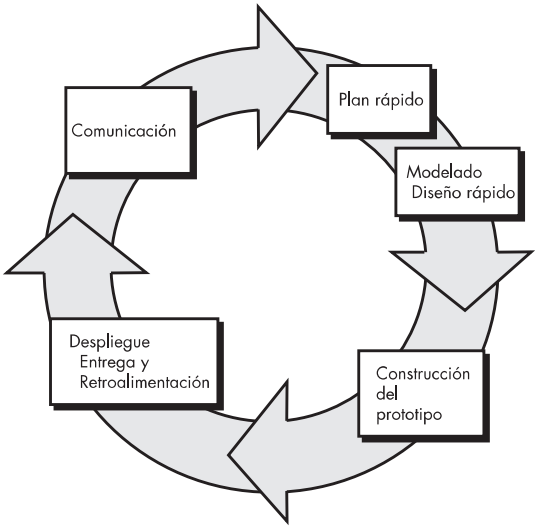
El paradigma de hacer prototipos (véase la figura 2.6) comienza con comunicación. Usted se reúne con otros participantes para definir los objetivos generales del software, identifica cualesquiera requerimientos que conozca y detecta las áreas en las que es imprescindible una mayor definición. Se planea rápidamente una iteración para hacer el prototipo, y se lleva a cabo el modelado (en forma de un "diseño rápido"). Éste se centra en la representación de aquellos aspectos del software que serán visibles para los usuarios finales (por ejemplo, disposición de la interfaz humana o formatos de la pantalla de salida). El diseño rápido lleva a la construcción de un prototipo. Éste se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. La iteración ocurre a medida de que el prototipo es afinado para satisfacer las necesidades de distintos participantes, y al mismo tiempo le permite a usted entender mejor lo que se necesita hacer.

El ideal es que el prototipo sirva como mecanismo para identificar los requerimientos del software. Si va a construirse un prototipo, pueden utilizarse fragmentos de programas existentes o aplicar herramientas (por ejemplo, generadores de reportes y administradores de ventanas) que permitan generar rápidamente programas que funcionen.

Pero, ¿qué hacer con el prototipo cuando ya sirvió para el propósito descrito? Brooks [Bro95] da una respuesta:

> En la mayoría de proyectos es raro que el primer sistema elaborado sea utilizable. Tal vez sea muy lento, muy grande, difícil de usar o todo a la vez. No hay más alternativa que comenzar de nuevo, con más inteligencia, y construir una versión rediseñada en la que se resuelvan los problemas.

FIGURA 2.6
El paradigma de hacer prototipos



El prototipo sirve como "el primer sistema". Lo que Brooks recomienda es desecharlo. Pero esto quizá sea un punto de vista idealizado. Aunque algunos prototipos se construyen para ser "desechables", otros son evolutivos; es decir, poco a poco se transforman en el sistema real.

Tanto a los participantes como a los ingenieros de software les gusta el paradigma de hacer prototipos. Los usuarios adquieren la sensación del sistema real, y los desarrolladores logran construir algo de inmediato. No obstante, hacer prototipos llega a ser problemático por las siguientes razones:

Resista la presión para convertir un prototipo burdo en un producto terminado. Como resultado de ello, casi siempre disminuye la calidad.

1. Los participantes ven lo que parece ser una versión funcional del software, sin darse cuenta de que el prototipo se obtuvo de manera caprichosa; no perciben que en la prisa por hacer que funcionara, usted no consideró la calidad general del software o la facilidad de darle mantenimiento a largo plazo. Cuando se les informa que el producto debe rehacerse a fin de obtener altos niveles de calidad, los participantes gritan que es usted un tonto y piden "unos cuantos arreglos" para hacer del prototipo un producto funcional. Con demasiada frecuencia, el gerente de desarrollo del software cede.
2. Como ingeniero de software, es frecuente que llegue a compromisos respecto de la implementación a fin de hacer que el prototipo funcione rápido. Quizá utilice un sistema operativo inapropiado, o un lenguaje de programación tan sólo porque cuenta con él y lo conoce; tal vez implementó un algoritmo ineficiente sólo para demostrar capacidad. Después de un tiempo, quizá se sienta cómodo con esas elecciones y olvide todas las razones por las que eran inadecuadas. La elección de algo menos que lo ideal ahora ha pasado a formar parte del sistema.

Aunque puede haber problemas, hacer prototipos es un paradigma eficaz para la ingeniería de software. La clave es definir desde el principio las reglas del juego; es decir, todos los participantes deben estar de acuerdo en que el prototipo sirva como el mecanismo para definir los requerimientos. Después se descartará (al menos en parte) y se hará la ingeniería del software real con la mirada puesta en la calidad.

CasaSegura

Selección de un modelo de proceso, parte 1

La escena: Sala de juntas del grupo de ingeniería de software de CPI Corporation, compañía (ficticia) que manufactura artículos de consumo para el hogar y para uso comercial.

Participantes: Lee Warren, gerente de ingeniería; Doug Miller, gerente de ingeniería de software; Jamie Lazar, miembro del equipo de software; Vinod Raman, miembro del equipo de software; y Ed Robbins, miembro del equipo de software.

La conversación:

Lee: Recapitulemos. He dedicado algún tiempo al análisis de la línea de productos *CasaSegura*, según la vemos hasta el momento. No hay duda de que hemos efectuado mucho trabajo tan sólo para definir el concepto, pero me gustaría que ustedes comenzaran a pensar en cómo van a enfocar la parte del software de este proyecto.

Doug: Pareciera que en el pasado hemos estado muy desorganizados en nuestro enfoque del software.

Ed: No sé, Doug, siempre sacamos el producto.

Doug: Es cierto, pero no sin muchos sobresaltos, y este proyecto parece más grande y complejo que cualquier cosa que hayamos hecho antes.

Jamie: No parece tan mal, pero estoy de acuerdo… nuestro enfoque *ad hoc* de los proyectos anteriores no funcionará en éste, en particular si tenemos una fecha de entrega muy apretada.

Doug (sonríe): Quiero ser un poco más profesional en nuestro enfoque. La semana pasada asistí a un curso breve y aprendí mucho sobre ingeniería de software… algo bueno. Aquí necesitamos un proceso.

Jamie (con el ceño fruncido): Mi trabajo es producir programas de computadora, no papel.

Doug: Den una oportunidad antes de ser tan negativos conmigo. Lo que quiero decir es esto: [Doug pasa a describir la estructura del proceso vista en este capítulo y los modelos de proceso prescriptivo presentados hasta el momento.]

Doug: De cualquier forma, parece que un modelo lineal no es para nosotros… pues supone que conocemos todos los requerimientos y, conociendo esta empresa, eso no parece probable.

Vinod: Sí, y parece demasiado orientado a las tecnologías de información… tal vez sea bueno para hacer un sistema de control de inventarios o algo así, pero no parece bueno para *CasaSegura*.

Doug: Estoy de acuerdo.

Ed: Ese enfoque de hacer prototipos parece bueno. En todo caso, se asemeja mucho a lo que hacemos aquí.

Vinod: Eso es un problema. Me preocupa que no nos dé suficiente estructura.

Doug: No te preocupes. Tenemos muchas opciones más, y quisiera que ustedes, muchachos, elijan la que sea mejor para el equipo y para el proyecto.

El modelo espiral. Propuesto en primer lugar por Barry Boehm [Boe88], el *modelo espiral* es un modelo evolutivo del proceso del software y se acopla con la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistémicos del modelo de cascada. Tiene el potencial para hacer un desarrollo rápido de versiones cada vez más completas. Boehm [Boe01a] describe el modelo del modo siguiente:

> El modelo de desarrollo espiral es un generador de *modelo de proceso* impulsado por el *riesgo*, que se usa para guiar la ingeniería concurrente con participantes múltiples de sistemas intensivos en software. Tiene dos características distintivas principales. La primera es el enfoque *cíclico* para el crecimiento incremental del grado de definición de un sistema y su implementación, mientras que disminuye su grado de riesgo. La otra es un conjunto de *puntos de referencia de anclaje puntual* para asegurar el compromiso del participante con soluciones factibles y mutuamente satisfactorias.

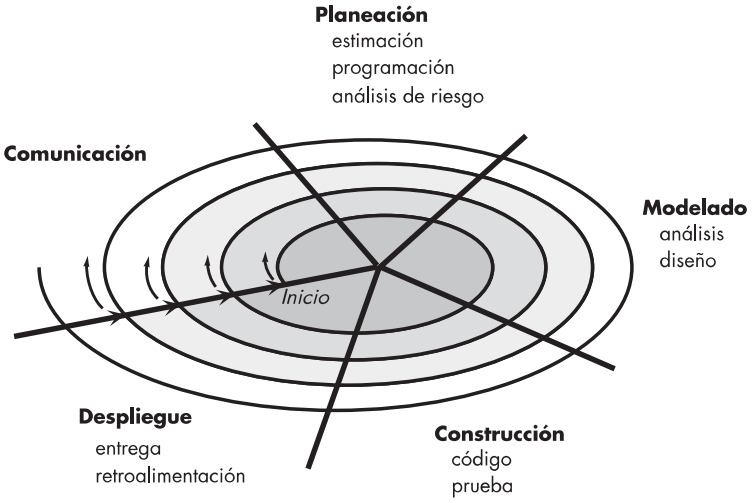
PUNTO CLAVE

El modelo en espiral se adapta para emplearse a lo largo de todo el ciclo de vida de una aplicación, desde el desarrollo del concepto hasta el mantenimiento.

Con el empleo del modelo espiral, el software se desarrolla en una serie de entregas evolutivas. Durante las primeras iteraciones, lo que se entrega puede ser un modelo o prototipo. En las iteraciones posteriores se producen versiones cada vez más completas del sistema cuya ingeniería se está haciendo.

Un modelo en espiral es dividido por el equipo de software en un conjunto de actividades estructurales. Para fines ilustrativos, se utilizan las actividades estructurales generales ya analizadas.[9] Cada una de ellas representa un segmento de la trayectoria espiral ilustrada en la figura 2.7. Al comenzar el proceso evolutivo, el equipo de software realiza actividades implícitas en un

FIGURA 2.7
Modelo de espiral común



9 El modelo espiral estudiado en esta sección es una variante del propuesto por Boehm. Para más información acerca del modelo espiral original, consulte [Boe88]. En [Boe98] se encuentra un análisis más reciente del modelo espiral del mismo autor.

circuito alrededor de la espiral en el sentido horario, partiendo del centro. El riesgo se considera conforme se desarrolla cada revolución (capítulo 28). En cada paso evolutivo se marcan *puntos de referencia puntuales*: combinación de productos del trabajo y condiciones que se encuentran a lo largo de la trayectoria de la espiral.

El primer circuito alrededor de la espiral da como resultado el desarrollo de una especificación del producto; las vueltas sucesivas se usan para desarrollar un prototipo y, luego, versiones cada vez más sofisticadas del software. Cada paso por la región de planeación da como resultado ajustes en el plan del proyecto. El costo y la programación de actividades se ajustan con base en la retroalimentación obtenida del cliente después de la entrega. Además, el gerente del proyecto ajusta el número planeado de iteraciones que se requieren para terminar el software.

WebRef

En la dirección **www.sei.cmu.edu/publications/documents/00.reports/00sr008.html** se encuentra información útil sobre el modelo espiral.

A diferencia de otros modelos del proceso que finalizan cuando se entrega el software, el modelo espiral puede adaptarse para aplicarse a lo largo de toda la vida del software de cómputo. Entonces, el primer circuito alrededor de la espiral quizá represente un "proyecto de desarrollo del concepto" que comienza en el centro de la espiral y continúa por iteraciones múltiples[10] hasta que queda terminado el desarrollo del concepto. Si el concepto va a desarrollarse en un producto real, el proceso sigue hacia fuera de la espiral y comienza un "proyecto de desarrollo de producto nuevo". El nuevo producto evolucionará a través de cierto número de iteraciones alrededor de la espiral. Más adelante puede usarse un circuito alrededor de la espiral para que represente un "proyecto de mejora del producto". En esencia, la espiral, cuando se caracteriza de este modo, sigue operativa hasta que el software se retira. Hay ocasiones en las que el proceso está inmóvil, pero siempre que se inicia un cambio comienza en el punto de entrada apropiado (por ejemplo, mejora del producto).

Si su administración pide un desarrollo apegado al presupuesto (mala idea, por lo general), la espiral se convierte en un problema. El costo se revisa y modifica cada vez que se termina un circuito.

El modelo espiral es un enfoque realista para el desarrollo de sistemas y de software a gran escala. Como el software evoluciona a medida que el proceso avanza, el desarrollador y cliente comprenden y reaccionan mejor ante los riesgos en cada nivel de evolución. El modelo espiral usa los prototipos como mecanismo de reducción de riesgos, pero, más importante, permite aplicar el enfoque de hacer prototipos en cualquier etapa de la evolución del producto. Mantiene el enfoque de escalón sistemático sugerido por el ciclo de vida clásico, pero lo incorpora en una estructura iterativa que refleja al mundo real en una forma más realista. El modelo espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto y, si se aplica de manera apropiada, debe reducir los riesgos antes de que se vuelvan un problema.

Cita:

"Sólo voy aquí y sólo el mañana me guía."

Dave Matthews Band

Pero, como otros paradigmas, el modelo espiral no es una panacea. Es difícil convencer a los clientes (en particular en situaciones bajo contrato) de que el enfoque evolutivo es controlable. Demanda mucha experiencia en la evaluación del riesgo y se basa en ella para llegar al éxito. No hay duda de que habrá problemas si algún riesgo importante no se descubre y administra.

2.3.4 Modelos concurrentes

El *modelo de desarrollo concurrente*, en ocasiones llamado *ingeniería concurrente*, permite que un equipo de software represente elementos iterativos y concurrentes de cualquiera de los modelos de proceso descritos en este capítulo. Por ejemplo, la actividad de modelado definida para el modelo espiral se logra por medio de invocar una o más de las siguientes acciones de software: hacer prototipos, análisis y diseño.[11]

Con frecuencia, el modelo concurrente es más apropiado para proyectos de ingeniería de productos en los que se involucran varios equipos de trabajo.

La figura 2.8 muestra la representación esquemática de una actividad de ingeniería de software dentro de la actividad de modelado con el uso del enfoque de modelado concurrente. La

10 Las flechas que apuntan hacia dentro a lo largo del eje que separa la región del **despliegue** de la de **comunicación** indican un potencial para la iteración local a lo largo de la misma trayectoria espiral.

11 Debe observarse que el análisis y diseño son tareas complejas que requieren mucho análisis. La parte 2 de este libro considera en detalle dichos temas.

CasaSegura

Selección de un modelo de proceso, parte 2

La escena: Sala de juntas del grupo de ingeniería de software de CPI Corporation, compañía que manufactura productos de consumo para uso doméstico y comercial.

Participantes: Lee Warren, gerente de ingeniería; Doug Miller, gerente de ingeniería de software; Vinod y Jamie, miembros del equipo de ingeniería de software.

La conversación: [Doug describe las opciones de proceso evolutivo.]

Jamie: Ahora me doy cuenta de algo. El enfoque incremental tiene sentido, y en verdad me gusta el flujo del modelo en espiral. Es bastante realista.

Vinod: De acuerdo. Entregamos un incremento, aprendemos de la retroalimentación del cliente, volvemos a planear y luego entregamos otro incremento. También se ajusta a la naturaleza del producto. Podemos lanzar con rapidez algo al mercado y luego agregar funcionalidad con cada versión, digo… con cada incremento.

Lee: Un momento. Doug, ¿dijiste que volveríamos a hacer el plan a cada vuelta de la espiral? Eso no es nada bueno; necesitamos un plan, un programa de actividades y apegarnos a ellos.

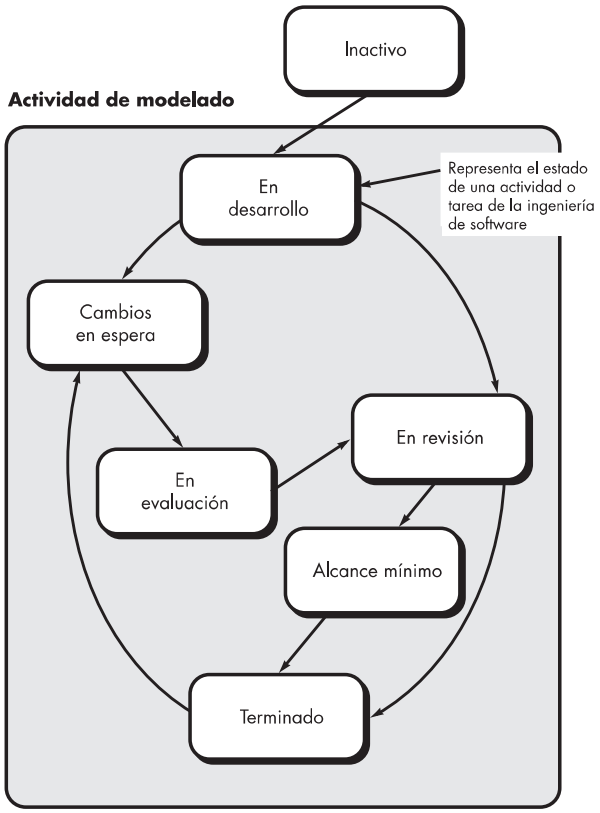
Doug: Ésa es la vieja escuela, Lee. Como dijeron los chicos, tenemos que hacerlo apegado a la realidad. Afirmo que es mejor afinar el plan a medida de que aprendamos más y conforme se soliciten cambios. Eso es más realista. ¿Qué sentido tiene un plan si no refleja la realidad?

Lee (con el ceño fruncido): Supongo, pero… a la alta dirección no le va a gustar… quieren un plan fijo.

Doug (sonriente): Entonces tendrás que reeducarlos, amigo.

actividad **—modelado—** puede estar en cualquiera de los estados[12] mencionados en un momento dado. En forma similar, es posible representar de manera análoga otras actividades, acciones o tareas (por ejemplo, **comunicación** o **construcción**). Todas las actividades de ingeniería de software existen de manera concurrente, pero se hallan en diferentes estados.

Figura 2.8
Un elemento del modelo de proceso concurrente



12 Un *estado* es algún modo de comportamiento observable externamente.

Por ejemplo, la actividad de comunicación (no se muestra en la figura) termina su primera iteración al principio de un proyecto y existe en el estado de **cambios en espera**. La actividad de modelado (que existía en estado **inactivo** mientras concluía la comunicación inicial, ahora hace una transición al estado **en desarrollo**. Sin embargo, si el cliente indica que deben hacerse cambios en los requerimientos, la actividad de modelado pasa del estado **en desarrollo** al de **cambios en espera**.

El modelado concurrente define una serie de eventos que desencadenan transiciones de un estado a otro para cada una de las actividades, acciones o tareas de la ingeniería de software. Por ejemplo, durante las primeras etapas del diseño (acción importante de la ingeniería de software que ocurre durante la actividad de modelado), no se detecta una inconsistencia en el modelo de requerimientos. Esto genera el evento *corrección del modelo de análisis*, que disparará la acción de análisis de requerimientos del estado **terminado** al de **cambios en espera**.

Cita:

"Todo proceso en su organización tiene un cliente, y un proceso sin cliente no tiene propósito."

V. Daniel Hunt

El modelado concurrente es aplicable a todos los tipos de desarrollo de software y proporciona un panorama apropiado del estado actual del proyecto. En lugar de confinar las actividades, acciones y tareas de la ingeniería de software a una secuencia de eventos, define una red del proceso. Cada actividad, acción o tarea de la red existe simultáneamente con otras actividades, acciones o tareas. Los eventos generados en cierto punto de la red del proceso desencadenan transiciones entre los estados.

2.3.5 Una última palabra acerca de los procesos evolutivos

Ya se dijo que el software de cómputo moderno se caracteriza por el cambio continuo, por tiempos de entrega muy apretados y por una necesidad apremiante de la satisfacción del cliente o usuario. En muchos casos, el tiempo para llegar al mercado es el requerimiento administrativo más importante. Si se pierde un nicho de mercado, todo el proyecto de software podría carecer de sentido.[13]

Los modelos de proceso evolutivo fueron concebidos para cumplir esos requisitos, pero, aun así, como clase general de modelos de proceso tienen demasiadas debilidades, que fueron resumidas por Nogueira y sus colegas [Nog00]:

> A pesar de los beneficios incuestionables de los procesos evolutivos de software, existen algunas preocupaciones. La primera es que hacer prototipos (y otros procesos evolutivos más sofisticados) plantea un problema para la planeación del proyecto debido a la incertidumbre en el número de ciclos que se requieren para elaborar el producto. La mayor parte de técnicas de administración y estimación de proyectos se basa en un planteamiento lineal de las actividades, por lo que no se ajustan por completo.
>
> En segundo lugar, los procesos evolutivos de software no establecen la velocidad máxima de la evolución. Si las evoluciones ocurren demasiado rápido, sin un periodo de relajamiento, es seguro que el proceso se volverá un caos. Por otro lado, si la velocidad es muy lenta, se verá perjudicada la productividad…
>
> En tercer lugar, los procesos de software deben centrarse en la flexibilidad y capacidad de extensión en lugar de en la alta calidad. Esto suena preocupante. Sin embargo, debe darse prioridad a la velocidad del desarrollo con el enfoque de cero defectos. Extender el desarrollo a fin de lograr alta calidad podría dar como resultado la entrega tardía del producto, cuando haya desaparecido el nicho de oportunidad. Este cambio de paradigma es impuesto por la competencia al borde del caos.

En realidad, sí parece preocupante un proceso del software que se centre en la flexibilidad, expansión y velocidad del desarrollo por encima de la calidad. No obstante, esta idea ha sido propuesta por varios expertos en ingeniería de software muy respetados ([You95], [Bac97]).

---

13 Sin embargo, es importante notar que ser el primero en llegar al mercado no es garantía de éxito. En realidad, muchos productos de software muy exitosos han llegado en segundo o hasta en tercer lugar al mercado (aprenden de los errores de sus antecesores).

El objetivo de los modelos evolutivos es desarrollar software de alta calidad[14] en forma iterativa o incremental. Sin embargo, es posible usar un proceso evolutivo para hacer énfasis en la flexibilidad, expansibilidad y velocidad del desarrollo. El reto para los equipos de software y sus administradores es establecer un balance apropiado entre estos parámetros críticos del proyecto y el producto, y la satisfacción del cliente (árbitro definitivo de la calidad del software).

2.4 MODELOS DE PROCESO ESPECIALIZADO

Los modelos de proceso especializado tienen muchas de las características de uno o más de los modelos tradicionales que se presentaron en las secciones anteriores. Sin embargo, dichos modelos tienden a aplicarse cuando se elige un enfoque de ingeniería de software especializado o definido muy específicamente.[15]

2.4.1 Desarrollo basado en componentes

WebRef

En la dirección **www.cbd-hq.com** hay información útil sobre el desarrollo basado en componentes.

Los componentes comerciales de software general (COTS, por sus siglas en inglés), desarrollados por vendedores que los ofrecen como productos, brindan una funcionalidad que se persigue con interfaces bien definidas que permiten que el componente se integre en el software que se va a construir. El *modelo de desarrollo basado en componentes* incorpora muchas de las características del modelo espiral. Es de naturaleza evolutiva [Nie92] y demanda un enfoque iterativo para la creación de software. Sin embargo, el modelo de desarrollo basado en componentes construye aplicaciones a partir de fragmentos de software prefabricados.

Las actividades de modelado y construcción comienzan con la identificación de candidatos de componentes. Éstos pueden diseñarse como módulos de software convencional o clases orientadas a objetos o paquetes[16] de clases. Sin importar la tecnología usada para crear los componentes, el modelo de desarrollo basado en componentes incorpora las etapas siguientes (se implementan con el uso de un enfoque evolutivo):

1. Se investigan y evalúan, para el tipo de aplicación de que se trate, productos disponibles basados en componentes.
2. Se consideran los aspectos de integración de los componentes.
3. Se diseña una arquitectura del software para que reciba los componentes.
4. Se integran los componentes en la arquitectura.
5. Se efectúan pruebas exhaustivas para asegurar la funcionalidad apropiada.

El modelo del desarrollo basado en componentes lleva a la reutilización del software, y eso da a los ingenieros de software varios beneficios en cuanto a la mensurabilidad. Si la reutilización de componentes se vuelve parte de la cultura, el equipo de ingeniería de software tiene la posibilidad tanto de reducir el ciclo de tiempo del desarrollo como el costo del proyecto. En el capítulo 10 se analiza con más detalle el desarrollo basado en componentes.

---

14 En este contexto, la calidad del software se define con mucha amplitud para que agrupe no sólo la satisfacción del cliente sino también varios criterios técnicos que se estudian en los capítulos 14 y 16.

15 En ciertos casos, los modelos de proceso especializado pueden caracterizarse mejor como un conjunto de técnicas o "metodología" para alcanzar una meta específica de desarrollo de software. No obstante, implican un proceso.

16 En el apéndice 2 se estudian los conceptos orientados a objetos, y se utilizan en toda la parte 2 del libro. En este contexto, una clase agrupa un conjunto de datos y los procedimientos para procesarlos. Un paquete de clases es un conjunto de clases relacionadas que funcionan juntas para alcanzar cierto resultado final.