**CAPÍTULO 2** MODELOS DEL PROCESO **33**



##### CLAVE

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

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 razonable- mente eficaz para los equipos de software. Sin embargo, el trabajo de ingeniería de software y el producto que genera siguen “al borde del caos”.

**2.3 MODELOS DE PR OCESO PR ESCRIPTIVO**

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:

Cita:

“Si el proceso está bien, los resultados cuidarán de sí mis- mos.”

**Takashi Osada**

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 rígi- da 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.

Las implicaciones filosóficas de este argumento son significativas para la ingeniería de software. Si los modelos de proceso prescriptivo5 buscan generar estructura y orden, ¿son inapropiados para el mundo del software, que se basa en el cambio? Pero si rechazamos los modelos de pro- ceso 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 prescrip- tivo 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 es- tructurales, 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 ge- nerales 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).

* + 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 defi- nidas 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 requerimien- tos están bien definidos y tienen una estabilidad razonable.

1. Los modelos de proceso prescriptivo en ocasiones son denominados modelos de proceso “tradicional”.

**34 PARTE UNO** EL PROCESO DEL SOFTWARE

FIGURA 2.3 **Modelo de la cascada**

**Comunicación**

inicio del proyecto recabar los requerimientos



##### 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.

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

**?**

El *modelo de la cascada*, a veces llamado *ciclo de vida clásico*, sugiere un enfoque sistemático y secuencial6 para el desarrollo del software, que comienza con la especificación de los reque- rimientos por parte del cliente y avanza a través de planeación, modelado, construcción y des- pliegue, para concluir con el apoyo del software terminado (véase la figura 2.3).

**Despliegue** entrega asistencia retroalimentación

**Construcción** código pruebas

**Modelado** análisis diseño

**Planeación** estimación programación seguimiento

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 construc- ción 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 em- bargo, 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 ocasio- nes surgen al aplicar el modelo de la cascada se encuentran los siguientes:

1. Es raro que los proyectos reales sigan el flujo secuencial propuesto por el modelo. Aun- que 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 requerimien- tos. El modelo de la cascada necesita que se haga y tiene dificultades para aceptar la in- certidumbre 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 pri- mera 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 reali- dad, ¡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

1. Aunque el modelo de la cascada propuesto originalmente por Winston Royce [Roy70] prevé los “bucles de retroa- limentación”, la gran mayoría de organizaciones que aplican este modelo de proceso lo tratan como si fuera estrictamente lineal.
2. En la parte 3 del libro se estudian con detalle las acciones de aseguramiento de la calidad.

**CAPÍTULO 2** MODELOS DEL PROCESO **35**

**El modelo en V**

FIGURA 2.4

Pruebas de integración

Pruebas del sistema

Pruebas de aceptación

Pruebas unitarias

Generación de código

Diseño de los componentes

Diseño de la arquitectura

Modelado de los requerimientos

Software ejecutable



##### 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.

***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.*

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

* + 1. **Modelos de proceso incremental**

Hay muchas situaciones en las que los requerimientos iniciales del software están razonable- mente 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 estudia- dos en la sección 2.1. En relación con la figura 2.5, el modelo incremental aplica secuencias li- neales en forma escalonada a medida que avanza el calendario de actividades. Cada secuencia lineal produce “incrementos” de software susceptibles de entregarse [McD93] de manera pare-

cida a los incrementos producidos en un flujo de proceso evolutivo (sección 2.3.3).

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 edi- ción 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 pági- nas. 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 *pro- ducto 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,

**36 PARTE UNO** EL PROCESO DEL SOFTWARE

FIGURA 2.5

**El modelo incremental**

**Calendario del proyecto**



Comunicación Planeación

Modelado (análisis, diseño)

Construcción (código, prueba) Despliegue (entrega, retroalimentación)

incremento # *n*

entrega del *n*-ésimo

incremento # 2 incremento

incremento # 1

entrega del segundo incremento

entrega del primer incremento

**Funcionalidad y características del software**



##### CLAVE

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

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 caracterís- ticas adicionales y más funcionalidad. Este proceso se repite después de entregar cada incre- mento, hasta terminar el producto final.

El modelo de proceso incremental se centra en que en cada incremento se entrega un pro- ducto 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 im- plementación completa del proyecto en el plazo establecido por el negocio. Los primeros incre- mentos 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.

* + 1. **Modelos de proceso evolutivo**

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 lan- zarse una versión limitada a fin de aliviar la presión de la competencia o del negocio; se com- prende 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 evo- luciona 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 pre- sentan 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.

**CAPÍTULO 2** MODELOS DEL PROCESO **37**

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**

***CONSEJO ***

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

**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 caracte- rísticas. 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.

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 cuales- quiera 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 retroalimenta- ción para mejorar los requerimientos. La iteración ocurre a medida de que el prototipo es afi- nado 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 existen- tes o aplicar herramientas (por ejemplo, generadores de reportes y administradores de venta- nas) 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**

Construcción del

prototipo

Despliegue Entrega y Retroalimentación

Modelado Diseño rápido

Comunicación

Plan rápido

**38 PARTE UNO** EL PROCESO DEL SOFTWARE

***CONSEJO ***

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

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 si- guientes razones:

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 facili- dad 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 funcio- nal. Con demasiada frecuencia, el gerente de desarrollo del software cede.
2. Como ingeniero de software, es frecuente que llegue a compromisos respecto de la im- plementación 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 parti- cipantes 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 pen- sar en cómo van a enfocar la parte del software de este proyecto.

**Doug:** Pareciera que en el pasado hemos estado muy desorganiza- dos 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 enfo- que *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 progra- mas 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.

**CAPÍTULO 2** MODELOS DEL PROCESO **39**

**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.



##### 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.

**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 soft- ware. Tiene dos características distintivas principales. La primera es el enfoque *cíclico* para el creci- miento incremental del grado de definición de un sistema y su implementación, mientras que dismi- nuye 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.

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 inge- niería 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 ana- lizadas.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

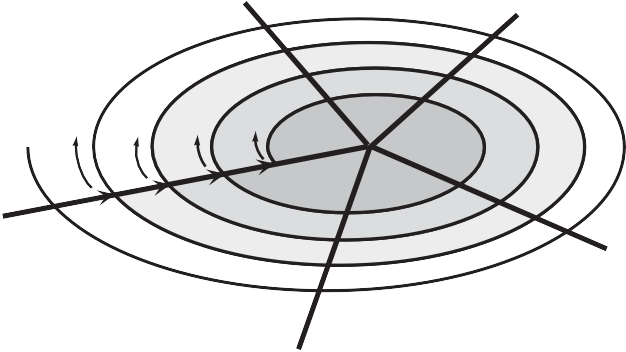
**Planeación**

**Modelo de espiral**

**común**

estimación

programación análisis de riesgo

**Comunicación**

*Inicio*

**Modelado** análisis diseño

**Despliegue** entrega retroalimentación

**Construcción** código prueba

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.

**40 PARTE UNO** EL PROCESO DEL SOFTWARE

***CONSEJO ***

*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.*

Cita:

“Sólo voy aquí y sólo el mañana me guía.”

**Dave Matthews Band**

***CONSEJO ***

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

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 especifica- ción 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 resul- tado 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. A diferencia de otros modelos del proceso que finalizan cuando se entrega el software,

**WebRef**

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

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 múltiples10 hasta que queda terminado el desarrollo del concepto. Si el concepto va a desarro- llarse 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).

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 de- manda 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.

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.

* + 1. **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 mo- delos 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 soft- ware: hacer prototipos, análisis y diseño.11

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

1. Las flechas que apuntan hacia dentro a lo largo del eje que separa la región del **despliegue** de la de **comunica- ción** indican un potencial para la iteración local a lo largo de la misma trayectoria espiral.
2. 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.

**CAPÍTULO 2** MODELOS DEL PROCESO **41**



**CASASEGURA**

***Selección de un modelo de proceso,*** otro incremento. También se ajusta a la naturaleza del producto.

***parte 2*** Podemos lanzar con rapidez algo al mercado y luego agregar fun- cionalidad con cada versión, digo… con cada incremento.

**La escena:** Sala de juntas del grupo de ingeniería de software de **Lee:** Un momento. Doug, ¿dijiste que volveríamos a hacer el plan a CPI Corporation, compañía que manufactura productos de consumo cada vuelta de la espiral? Eso no es nada bueno; necesitamos un para uso doméstico y comercial. plan, un programa de actividades y apegarnos a ellos.

**Participantes:** Lee Warren, gerente de ingeniería; Doug Miller, **Doug:** Ésa es la vieja escuela, Lee. Como dijeron los chicos, tene-

gerente de ingeniería de software; Vinod y Jamie, miembros del mos que hacerlo apegado a la realidad. Afirmo que es mejor afinar equipo de ingeniería de software. el plan a medida de que aprendamos más y conforme se soliciten **La conversación:** [Doug describe las opciones de proceso evoluti- cambios. Eso es más realista. ¿Qué sentido tiene un plan si no refleja vo.] la realidad?

**Jamie:** Ahora me doy cuenta de algo. El enfoque incremental tiene **Lee (con el ceño fruncido):** Supongo, pero… a la alta dirección

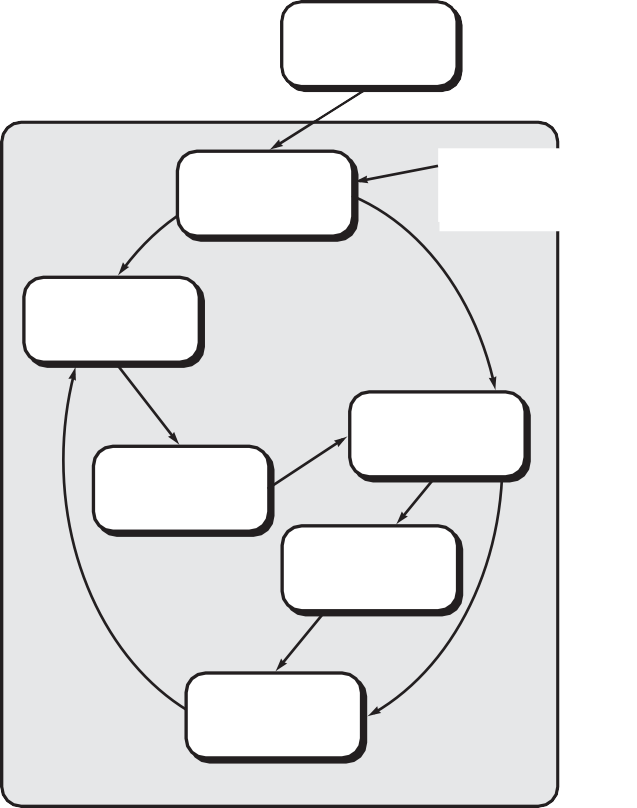
sentido, y en verdad me gusta el flujo del modelo en espiral. Es bas- no le va a gustar… quieren un plan fijo.

tante realista.

**Vinod:** De acuerdo. Entregamos un incremento, aprendemos de la **Doug (sonriente):** Entonces tendrás que reeducarlos, amigo.

retroalimentación del cliente, volvemos a planear y luego entregamos

actividad **—modelado—** puede estar en cualquiera de los estados12 mencionados en un mo- mento dado. En forma similar, es posible representar de manera análoga otras actividades, acciones o tareas (por ejemplo, **comunicación** o **construcción**). Todas las actividades de in- geniería de software existen de manera concurrente, pero se hallan en diferentes estados.



Inactivo

**Actividad de modelado**

En desarrollo

Representa el estado de una actividad o tarea de la ingeniería de software

Cambios en espera

En revisión

En evaluación

Alcance mínimo

Terminado

FIGURA 2.8

**Un elemento del modelo de proceso concurrente**

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

**42 PARTE UNO** EL PROCESO DEL SOFTWARE

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 soft- ware 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 organiza- ción tiene un cliente, y un proceso sin cliente no tiene pro- pósito.”

**V. Daniel Hunt**

El modelado concurrente es aplicable a todos los tipos de desarrollo de software y propor- ciona un panorama apropiado del estado actual del proyecto. En lugar de confinar las activida- des, 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 activida- des, acciones o tareas. Los eventos generados en cierto punto de la red del proceso desencade- nan transiciones entre los estados.

* + 1. Una última palabra acerca de los procesos evolutivos

Ya se dijo que el software de cómputo moderno se caracteriza por el cambio continuo, por tiem- pos 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 re- sumidas por Nogueira y sus colegas [Nog00]:

A pesar de los beneficios incuestionables de los procesos evolutivos de software, existen algunas pre- ocupaciones. 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 pro- yectos 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 pro- ductividad…

En tercer lugar, los procesos de software deben centrarse en la flexibilidad y capacidad de exten- sión 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 (apren- den de los errores de sus antecesores).

**CAPÍTULO 2** MODELOS DEL PROCESO **43**

El objetivo de los modelos evolutivos es desarrollar software de alta calidad14 en forma itera- tiva 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 pro- yecto y el producto, y la satisfacción del cliente (árbitro definitivo de la calidad del software).

**2.4 MODELOS DE PR OCESO 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 mo- delos tienden a aplicarse cuando se elige un enfoque de ingeniería de software especializado o definido muy específicamente.15

* + 1. Desarrollo basado en componentes

Los componentes comerciales de software general (COTS, por sus siglas en inglés), desarrolla- dos 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 caracte- rísticas 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.

**WebRef**

En la dirección [**www.cbd-hq.com**](http://www.cbd-hq.com/)hay información útil sobre el desarrollo basado en componentes.

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 paquetes16 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 disponi- bles 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 reutiliza- ción 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.

1. 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.
2. En ciertos casos, los modelos de proceso especializado pueden caracterizarse mejor como un conjunto de téc- nicas o “metodología” para alcanzar una meta específica de desarrollo de software. No obstante, implican un proceso.
3. 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.

**44 PARTE UNO** EL PROCESO DEL SOFTWARE

**Si con los métodos formales puede demostrarse lo correcto de un software, ¿por qué no son ampliamente utilizados?**

**?**

**WebRef**

Existen muchos recursos e información sobre SOA en la dirección: **aosd.net**



##### CLAVE

El DSOA define “aspectos” que expresan preocupaciones del cliente que afectan múltiples funciones, características e información del sistema.

* + 1. **El modelo de métodos formales**

El *modelo de métodos formales* agrupa actividades que llevan a la especificación matemática formal del software de cómputo. Los métodos formales permiten especificar, desarrollar y veri- ficar un sistema basado en computadora por medio del empleo de una notación matemática rigurosa. Ciertas organizaciones de desarrollo de software aplican una variante de este enfoque, que se denomina *ingeniería de software de quirófano* [Mil87, Dye92].

Cuando durante el desarrollo se usan métodos formales (capítulo 21), se obtiene un meca- nismo para eliminar muchos de los problemas difíciles de vencer con otros paradigmas de la ingeniería de software. Lo ambiguo, incompleto e inconsistente se descubre y corrige con más facilidad, no a través de una revisión *ad hoc* sino con la aplicación de análisis matemático. Si durante el diseño se emplean métodos formales, éstos sirven como base para la verificación del programa, y así permiten descubrir y corregir errores que de otro modo no serían detectados.

Aunque el modelo de los métodos formales no es el más seguido, promete un software libre de defectos. Sin embargo, se han expresado preocupaciones acerca de su aplicabilidad en un ambiente de negocios:

* El desarrollo de modelos formales consume mucho tiempo y es caro.
* Debido a que pocos desarrolladores de software tienen la formación necesaria para aplicar métodos formales, se requiere mucha capacitación.
* Es difícil utilizar los modelos como mecanismo de comunicación para clientes sin complejidad técnica.

A pesar de estas preocupaciones, el enfoque de los métodos formales ha ganado partidarios entre los desarrolladores que deben construir software de primera calidad en seguridad (por ejemplo, control electrónico de aeronaves y equipos médicos), y entre los desarrolladores que sufrirían graves pérdidas económicas si ocurrieran errores en su software.

* + 1. Desarrollo de software orientado a aspectos

Sin importar el proceso del software que se elija, los constructores de software complejo imple- mentan de manera invariable un conjunto de características, funciones y contenido de informa- ción localizados. Estas características localizadas del software se modelan como componentes (clases orientadas a objetos) y luego se construyen dentro del contexto de una arquitectura de sistemas. A medida que los sistemas modernos basados en computadora se hacen más sofisti- cados (y complejos), ciertas *preocupaciones* —propiedades que requiere el cliente o áreas de interés técnico— se extienden a toda la arquitectura. Algunas de ellas son las propiedades de alto nivel de un sistema (por ejemplo, seguridad y tolerancia a fallas). Otras afectan a funcio- nes (aplicación de las reglas de negocios), mientras que otras más son sistémicas (sincroniza- ción de la tarea o administración de la memoria).

Cuando las preocupaciones afectan múltiples funciones, características e información del sistema, es frecuente que se les llame *preocupaciones globales*. Los *requerimientos del aspecto* definen aquellas preocupaciones globales que tienen algún efecto a través de la arquitectura del software. El *desarrollo de software orientado a aspectos* (DSOA), conocido también como *progra- mación orientada a aspectos* (POA), es un paradigma de ingeniería de software relativamente nuevo que proporciona un proceso y enfoque metodológico para definir, especificar, diseñar y construir *aspectos*: “mecanismos más allá de subrutinas y herencia para localizar la expresión de una preocupación global” [Elr01].

Grundy [Gru02] analiza con más profundidad los aspectos en el contexto de lo que denomina

*ingeniería de componentes orientada a aspectos* (ICOA):

La ICOA usa el concepto de rebanadas horizontales a través de componentes de software descom- puestos verticalmente, llamados “aspectos”, para caracterizar las propiedades globales funcionales y

**CAPÍTULO 2** MODELOS DEL PROCESO **45**

no funcionales de los componentes. Los aspectos comunes y sistémicos incluyen interfaces de usua- rio, trabajo en colaboración, distribución, persistencia, administración de la memoria, procesamiento de las transacciones, seguridad, integridad, etc. Los componentes pueden proveer o requerir uno o más “detalles de aspectos” en relación con un aspecto particular, como un mecanismo de visión, al- cance extensible y clase de interfaz (aspectos de la interfaz de usuario); generación de eventos, trans- porte y recepción (aspectos de distribución); almacenamiento, recuperación e indización de datos (aspectos de persistencia); autenticación, encriptación y derechos de acceso (aspectos de seguridad); descomposición de las transacciones, control de concurrencia y estrategia de registro (aspectos de las transacciones), entre otros. Cada detalle del aspecto tiene cierto número de propiedades relacionadas con las características funcionales o no del detalle del aspecto.

Aún no madura un proceso distinto orientado a aspectos. Sin embargo, es probable que un proceso así adopte características tanto de los modelos de proceso evolutivo como concurrente. El modelo evolutivo es apropiado en tanto los aspectos se identifican y después se construyen. La naturaleza paralela del desarrollo concurrente es esencial porque la ingeniería de aspectos se hace en forma independiente de los componentes de software localizados; aun así, los as- pectos tienen un efecto directo sobre éstos. De esta forma, es esencial disponer de comunica- ción asincrónica entre las actividades de proceso del software aplicadas a la ingeniería, y la construcción de los aspectos y componentes.

El análisis detallado del desarrollo de software orientado al aspecto se deja a libros especia- lizados en el tema. Si el lector tiene interés en profundizar, se le invita a consultar [Saf08], [Cla05], [Jac04] y [Gra03].

Administración del proceso

**Objetivo:** Ayudar a la definición, ejecución y administra- ción de modelos de proceso prescriptivo.

**Mecánica:** Las herramientas de administración del proceso permiten que una organización o equipo de software defina un modelo com- pleto del proceso (actividades estructurales, acciones, tareas, asegura- miento de la calidad, puntos de revisión, referencias y productos del trabajo). Además, las herramientas proporcionan un mapa conforme los ingenieros de software realizan el trabajo técnico, y una plantilla para los gerentes que deben dar seguimiento y controlar el proceso del software.

**Herramientas representativas:**17

*GDPA, grupo de herramientas de investigación de definición del pro- ceso*, desarrollada por la Universidad de Bremen, en Alemania

**HERRAMIENTAS DE SOFTWARE**

([**www.informatik.uni-bremen.de/uniform/gdpa/**](http://www.informatik.uni-bremen.de/uniform/gdpa/) **home.htm**), proporciona una amplia variedad de funciones para modelar y administrar procesos.



*SpeeDev*, desarrollada por SpeeDev Corporation (**www.speedev. com**), incluye un conjunto de herramientas para la definición del proceso, administración de los requerimientos, resolución de pro- blemas, y planeación y seguimiento del proyecto.

*ProVision BPMx*, desarrollado por Proforma (**www.proforma- corp.com**), es representativo de muchas herramientas que ayu- dan a definir el proceso y que automatizan el flujo del trabajo.

En la dirección [**www.processwave.net/Links/tool\_links.**](http://www.processwave.net/Links/tool_links) **htm**, se encuentra una lista extensa de muchas herramientas dife- rentes asociadas con el proceso del software.

**2.5 EL PR OCESO UNIFICADO**

En su libro fundamental, *Unified Process*, Ivar Jacobson, Grady Booch y James Rumbaugh [Jac99] analizan la necesidad de un proceso del software “impulsado por el caso de uso, centrado en la arquitectura, iterativo e incremental”, con la afirmación siguiente:

17 Las herramientas mencionadas aquí no representan una obligación; sólo son una muestra de las de esta catego- ría. En la mayoría de casos, los nombres de las herramientas son marcas registradas por sus desarrolladores respectivos.

**46 PARTE UNO** EL PROCESO DEL SOFTWARE



##### CLAVE

Las fases del PU tienen un objetivo similar al de las actividades estructurales generales definidas en este libro.

En la actualidad, la tendencia en el software es hacia sistemas más grandes y complejos. Eso se debe en parte al hecho de que año tras año las computadoras son más poderosas, lo que hace que los usuarios esperen más de ellas. Esta tendencia también se ha visto influida por el uso creciente de in- ternet para intercambiar toda clase de información […] Nuestro apetito por software cada vez más sofisticado aumenta conforme aprendemos, entre un lanzamiento y otro de un producto, cómo mejo- rar éste. Queremos software que se adapte mejor a nuestras necesidades, pero eso a su vez lo hace más complejo. En pocas palabras, queremos más.

En cierto modo, el proceso unificado es un intento por obtener los mejores rasgos y caracterís- ticas de los modelos tradicionales del proceso del software, pero en forma que implemente muchos de los mejores principios del desarrollo ágil de software (véase el capítulo 3). El proceso unificado reconoce la importancia de la comunicación con el cliente y los métodos directos para describir su punto de vista respecto de un sistema (el caso de uso).18 Hace énfasis en la impor- tancia de la arquitectura del software y “ayuda a que el arquitecto se centre en las metas correc- tas, tales como que sea comprensible, permita cambios futuros y la reutilización” [Jac99]: Su- giere un flujo del proceso iterativo e incremental, lo que da la sensación evolutiva que resulta esencial en el desarrollo moderno del software.

* + 1. **Breve historia**

Al principio de la década de 1990, James Rumbaugh [Rum91], Grady Booch [Boo94] e Ivar Jacob- son [Jac92] comenzaron a trabajar en un “método unificado” que combinaría lo mejor de cada uno de sus métodos individuales de análisis y diseño orientado a objetos. El resultado fue un UML, *lenguaje de modelado unificado*, que contiene una notación robusta para el modelado y desarrollo de los sistemas orientados a objetos.

El UML se utiliza en toda la parte 2 del libro para representar tanto los modelos de requeri- mientos como el diseño. En el apéndice 1 se presenta un método introductorio a la enseñanza para quienes no están familiarizados con las reglas básicas de notación y modelado con el UML. El estudio exhaustivo del UML se deja a libros dedicados al tema. En el apéndice 1 se enlistan los textos recomendables.

El UML brinda la tecnología necesaria para apoyar la práctica de la ingeniería de software orientada a objetos, pero no da la estructura del proceso que guíe a los equipos del proyecto cuando aplican la tecnología. En los siguientes años, Jacobson, Rumbaugh y Booch desarrolla- ron el *proceso unificado*, estructura para la ingeniería de software orientado a objetos que utiliza UML. Actualmente, el proceso unificado (PU) y el UML se usan mucho en proyectos de toda clase orientados a objetos. El modelo iterativo e incremental propuesto por el PU puede y debe adap- tarse para que satisfaga necesidades específicas del proyecto.

* + 1. Fases del proceso unificado19

Al principio de este capítulo se estudiaron cinco actividades estructurales generales y se dijo que podían usarse para describir cualquier modelo de proceso del software. El proceso unificado no es la excepción. La figura 2.9 ilustra las “fases” del PU y las relaciona con las actividades gene- rales estudiadas en el capítulo 1 y al inicio de éste.

La *fase de concepción* del PU agrupa actividades tanto de comunicación con el cliente como de planeación. Al colaborar con los participantes, se identifican los requerimientos del negocio,

1. El *caso de uso* (véase el capítulo 5) es la narración o plantilla que describe una función o rasgo de un sistema desde el punto de vista del usuario. Éste escribe un caso en uso que sirve como base para la creación de un modelo de requerimientos más completos.
2. El proceso unificado en ocasiones recibe el nombre de *Proceso Racional Unificado* (PRU), acuñado por Rational Corporation (adquirida posteriormente por IBM), que contribuyó desde el principio al desarrollo y mejora del PU y a la elaboración de ambientes completos (herramientas y tecnología) que apoyan el proceso.

**CAPÍTULO 2** MODELOS DEL PROCESO **47**

**El proceso unificado**

FIGURA 2.9

***Elaboración***

***Concepción***



Lanzamiento

incremento del software

***Construcción Transición***

***Producción***

se propone una arquitectura aproximada para el sistema y se desarrolla un plan para la natura- leza iterativa e incremental del proyecto en cuestión. Los requerimientos fundamentales del negocio se describen por medio de un conjunto de casos de uso preliminares (véase el capítulo 5) que detallan las características y funciones que desea cada clase principal de usuarios. En este punto, la arquitectura no es más que un lineamiento tentativo de subsistemas principales y la función y rasgos que tienen. La arquitectura se mejorará después y se expandirá en un conjunto de modelos que representarán distintos puntos de vista del sistema. La planeación identifica los recursos, evalúa los riesgos principales, define un programa de actividades y establece una base para las fases que se van a aplicar a medida que avanza el incremento del software.

La *fase de elaboración* incluye las actividades de comunicación y modelado del modelo gene- ral del proceso (véase la figura 2.9). La elaboración mejora y amplía los casos de uso prelimina- res desarrollados como parte de la fase de concepción y aumenta la representación de la arqui- tectura para incluir cinco puntos de vista distintos del software: los modelos del caso de uso, de requerimientos, del diseño, de la implementación y del despliegue. En ciertos casos, la elabora- ción crea una “línea de base de la arquitectura ejecutable” [Arl02] que representa un sistema ejecutable de “primer corte”.20 La línea de base de la arquitectura demuestra la viabilidad de ésta, pero no proporciona todas las características y funciones que se requieren para usar el sistema. Además, al terminar la fase de elaboración se revisa con cuidado el plan a fin de asegurar que el alcance, riesgos y fechas de entrega siguen siendo razonables. Es frecuente que en este mo- mento se hagan modificaciones al plan.

La *fase de construcción* del PU es idéntica a la actividad de construcción definida para el pro- ceso general del software. Con el uso del modelo de arquitectura como entrada, la fase de construcción desarrolla o adquiere los componentes del software que harán que cada caso de uso sea operativo para los usuarios finales. Para lograrlo, se completan los modelos de re- querimientos y diseño que se comenzaron durante la fase de elaboración, a fin de que reflejen la versión final del incremento de software. Después se implementan en código fuente todas las características y funciones necesarias para el incremento de software (por ejemplo, el lanza- miento). A medida de que se implementan los componentes, se diseñan y efectúan pruebas unitarias21 para cada uno. Además, se realizan actividades de integración (ensamble de compo-

**WebRef**

En la dirección **www.ambysoft. com/unifiedprocess/agileUP. html**, se encuentra un análisis interesante del PU en el contexto del desarrollo ágil.

1. Es importante darse cuenta de que la línea de base de la arquitectura no es un prototipo y que no se desecha. Por el contrario, es revestida durante la fase siguiente del PU.
2. En los capítulos 17 a 20 se presenta el análisis exhaustivo de las pruebas del software (incluso las *pruebas unita- rias*).

**48 PARTE UNO** EL PROCESO DEL SOFTWARE

nentes y pruebas de integración). Se emplean casos de uso para obtener un grupo de pruebas de aceptación que se ejecutan antes de comenzar la siguiente fase del PU.

La *fase de transición* del PU incluye las últimas etapas de la actividad general de construcción y la primera parte de la actividad de despliegue general (entrega y retroalimentación). Se da el software a los usuarios finales para las pruebas beta, quienes reportan tanto los defectos como los cambios necesarios. Además, el equipo de software genera la información de apoyo nece- saria (por ejemplo, manuales de usuario, guías de solución de problemas, procedimientos de instalación, etc.) que se requiere para el lanzamiento. Al finalizar la fase de transición, el soft- ware incrementado se convierte en un producto utilizable que se lanza.

La *fase de producción* del PU coincide con la actividad de despliegue del proceso general. Durante esta fase, se vigila el uso que se da al software, se brinda apoyo para el ambiente de operación (infraestructura) y se reportan defectos y solicitudes de cambio para su evaluación.

Es probable que al mismo tiempo que se llevan a cabo las fases de construcción, transición y producción, comience el trabajo sobre el siguiente incremento del software. Esto significa que las cinco fases del PU no ocurren en secuencia sino que concurren en forma escalonada.

El flujo de trabajo de la ingeniería de software está distribuido a través de todas las fases del PU. En el contexto de éste, un *flujo de trabajo* es análogo al conjunto de tareas (que ya se descri- bió en este capítulo). Es decir, un flujo de trabajo identifica las tareas necesarias para completar una acción importante de la ingeniería de software y los productos de trabajo que se generan como consecuencia de la terminación exitosa de aquéllas. Debe notarse que no toda tarea iden- tificada para el flujo de trabajo del PU es realizada en todos los proyectos de software. El equipo adapta el proceso (acciones, tareas, subtareas y productos del trabajo) a fin de que cumpla sus necesidades.

**2.6 MODELOS DEL PR OCESO PERSONAL Y DEL EQUIPO**

El mejor proceso del software es el que está cerca de las personas que harán el trabajo. Si un modelo del proceso del software se ha desarrollado en un nivel corporativo u organizacional, será eficaz sólo si acepta una adaptación significativa para que cubra las necesidades del equipo de proyecto que en realidad hace el trabajo de ingeniería de software. En la situación ideal se crearía un proceso que se ajustara del mejor modo a los requerimientos, y al mismo tiempo cubriera las más amplias necesidades del equipo y de la organización. En forma alternativa, el equipo crearía un proceso propio que satisficiera las necesidades más estrechas de los indivi- duos y las más generales de la organización. Watts Humphrey ([Hum97] y [Hum00]) afirma que es posible crear un “proceso personal de software” y/o un “proceso del equipo de software”. Ambos requieren trabajo duro, capacitación y coordinación, pero los dos son asequibles.22

* + 1. Proceso personal del software (PPS)

Cita:

“La persona que es exitosa tan sólo se ha hecho el hábito de hacer las cosas que no hacen las personas que no tienen éxito.”

**Dexter Yager**

Todo desarrollador utiliza algún proceso para elaborar software de cómputo. El proceso puede ser caprichoso o *ad hoc*; quizá cambie a diario; tal vez no sea eficiente, eficaz o incluso no sirva; pero sí existe un “proceso”. Watts Humphrey [Hum97] sugiere que a fin de cambiar un proceso personal ineficaz, un individuo debe pasar por las cuatro fases, cada una de las cuales requiere capacitación e instrumentación cuidadosa. El *proceso personal del software* (PPS) pone el énfasis en la medición personal tanto del producto del trabajo que se genera como de su calidad. Ade- más, el PPS responsabiliza al profesional acerca de la planeación del proyecto (por ejemplo,

**WebRef**

En la dirección [**www.ipd.uka.de/**](http://www.ipd.uka.de/) **PSP**, se hallan muchos recursos para el PPS.

22 Es útil notar que quienes proponen un desarrollo ágil del software (véase el capítulo 3) también plantean que el proceso debe ser cercano al equipo. Para lograr esto sugieren un método alternativo.

**CAPÍTULO 2** MODELOS DEL PROCESO **49**

**¿Qué actividades estructurales se usan durante el PPS?**

**?**



##### CLAVE

El PPS pone el énfasis en la necesidad de registrar y analizar los tipos de errores que se cometen, de modo que se desarrollen estrategias para eliminarlos.

**WebRef**

En la dirección [**www.sei.cmu.**](http://www.sei.cmu/) **edu/tsp/**, hay información sobre la formación de equipos de alto rendimiento que usan PES y PPS.

estimación y programación de actividades) y delega en el practicante el poder de controlar la calidad de todos los productos del trabajo de software que se desarrollen. El modelo del PPS define cinco actividades estructurales:

**Planeación.** Esta actividad aísla los requerimientos y desarrolla las estimaciones tanto del tamaño como de los recursos. Además, realiza la estimación de los defectos (el número de defectos proyectados para el trabajo). Todas las mediciones se registran en hojas de tra- bajo o plantillas. Por último, se identifican las tareas de desarrollo y se crea un programa para el proyecto.

**Diseño de alto nivel.** Se desarrollan las especificaciones externas para cada compo- nente que se va a construir y se crea el diseño de componentes. Si hay incertidumbre, se elaboran prototipos. Se registran todos los aspectos relevantes y se les da seguimiento.

**Revisión del diseño de alto nivel.** Se aplican métodos de verificación formal (véase el capítulo 21) para descubrir errores en el diseño. Se mantienen las mediciones para todas las tareas y resultados del trabajo importantes.

**Desarrollo.** Se mejora y revisa el diseño del componente. El código se genera, revisa, compila y prueba. Las mediciones se mantienen para todas las tareas y resultados de tra- bajo de importancia.

**Post mórtem.** Se determina la eficacia del proceso por medio de medidas y mediciones obtenidas (ésta es una cantidad sustancial de datos que deben analizarse con métodos es- tadísticos). Las medidas y mediciones deben dar la guía para modificar el proceso a fin de mejorar su eficacia.

El PPS enfatiza la necesidad de detectar pronto los errores; de igual importancia es entender los tipos de ellos que es probable cometer. Esto se logra a través de una actividad de evaluación rigurosa ejecutada para todos los productos del trabajo que se generen.

El PPS representa un enfoque disciplinado basado en la medición para la ingeniería de soft- ware que quizá sea un choque cultural para muchos de sus practicantes. Sin embargo, cuando se introduce el PPS en forma apropiada en los ingenieros de software [Hum96], es significativa la mejora resultante en la productividad de la ingeniería respectiva y en la calidad del software [Fer97]. No obstante, el PPS no ha sido adoptado con amplitud por la industria. Es triste recono- cer que las razones de esto tienen que ver más con la naturaleza humana y la inercia organiza- cional que con las fortalezas y debilidades del enfoque del PPS. Dicho enfoque plantea desafíos intelectuales y demanda un nivel de compromiso (por parte de los practicantes y sus adminis- tradores) que no siempre es posible obtener. La capacitación es relativamente larga y sus costos elevados. El nivel requerido de las mediciones es culturalmente difícil para muchas personas de la comunidad del software.

¿Es posible usar el PPS como un proceso eficaz de software a nivel personal? La respuesta es un rotundo “sí”. Pero aun si no se adoptara por completo el PPS, muchos de los conceptos del proceso de mejora personal que introduce constituyen un aprendizaje provechoso.

* + 1. **Proceso del equipo de software (PES)**

Debido a que muchos proyectos de software industrial son elaborados por un equipo de profe- sionales, Watts Humphrey extendió las lecciones aprendidas de la introducción del PPS y pro- puso un *proceso del equipo de software* (PES). El objetivo de éste es construir un equipo “autodi- rigido” para el proyecto, que se organice para producir software de alta calidad. Humphrey [Hum98] define los objetivos siguientes para el PES:

* Formar equipos autodirigidos que planeen y den seguimiento a su trabajo, que esta- blezcan metas y que sean dueños de sus procesos y planes. Éstos pueden ser equipos de software puros o de productos integrados (EPI) constituidos por 3 a 20 ingenieros.

**50 PARTE UNO** EL PROCESO DEL SOFTWARE

***CONSEJO ***

*Para formar un equipo autodirigido, usted debe colaborar bien en lo interno y comunicarse bien en lo externo.*



##### CLAVE

Los scripts del PES definen elementos del proceso del equipo y de las actividades que ocurren dentro del proceso.

* Mostrar a los gerentes cómo dirigir y motivar a sus equipos y cómo ayudarlos a mantener un rendimiento máximo.
* Acelerar la mejora del proceso del software, haciendo del modelo de madurez de la capacidad, CMM,23 nivel 5, el comportamiento normal y esperado.
* Brindar a las organizaciones muy maduras una guía para la mejora.
* Facilitar la enseñanza universitaria de aptitudes de equipo con grado industrial.

Un equipo autodirigido tiene la comprensión consistente de sus metas y objetivos generales; define el papel y responsabilidad de cada miembro del equipo; da seguimiento cuantitativo a los datos del proyecto (sobre la productividad y calidad); identifica un proceso de equipo que sea apropiado para el proyecto y una estrategia para implementarlo; define estándares locales apli- cables al trabajo de ingeniería de software del equipo; evalúa en forma continua el riesgo y re- acciona en consecuencia; y da seguimiento, administra y reporta el estado del proyecto.

El PES define las siguientes actividades estructurales: **inicio del proyecto, diseño de alto nivel, implementación, integración y pruebas,** y **post mórtem.** Como sus contrapartes del PPS (observe que la terminología es algo diferente), estas actividades permiten que el equipo planee, diseñe y construya software en forma disciplinada, al mismo tiempo que mide cuantita- tivamente el proceso y el producto. La etapa post mórtem es el escenario de las mejoras del proceso.

El PES utiliza una variedad amplia de *scripts*, formatos y estándares que guían a los miembros del equipo en su trabajo. Los *scripts* definen actividades específicas del proceso (por ejemplo, inicio del proyecto, diseño, implementación, integración y pruebas del sistema, y post mórtem), así como otras funciones más detalladas del trabajo (planeación del desarrollo, desarrollo de requerimientos, administración de la configuración del software y prueba unitaria) que forman parte del proceso de equipo.

El PES reconoce que los mejores equipos de software son los autodirigidos.24 Los miembros del equipo establecen los objetivos del proyecto, adaptan el proceso para que cubra las necesi- dades, controlan la programación de actividades del proyecto y, con la medida y análisis de las mediciones efectuadas, trabajan de manera continua en la mejora del enfoque de ingeniería de software que tiene el equipo.

Igual que el PPS, el PES es un enfoque riguroso para la ingeniería de software y proporciona beneficios distintivos y cuantificables en productividad y calidad. El equipo debe tener un com- promiso total con el proceso y recibir capacitación completa para asegurar que el enfoque se aplique en forma apropiada.

**2.7 TECNOLOGÍA DEL PR OCESO**

El equipo del software debe adaptar uno o más de los modelos del proceso estudiados en las secciones precedentes. Para ello, se han desarrollado *herramientas de tecnología del proceso* que ayudan a las organizaciones de software a analizar su proceso actual, organizar las tareas de trabajo, controlar y vigilar el avance, y administrar la calidad técnica.

Las herramientas de tecnología del proceso permiten que una organización de software construya un modelo automatizado de la estructura del proceso, conjuntos de tareas y activida- des sombrilla, estudiados en la sección 2.1. El modelo, que normalmente se representa como

1. El modelo de madurez de la capacidad (CMM), que es una medida de la eficacia de un proceso del software, se estudia en el capítulo 30.
2. En el capítulo 31 se analiza la importancia de los equipos “autoorganizados” como elemento clave del desarrollo ágil del software.

**CAPÍTULO 2** MODELOS DEL PROCESO **51**

una red, se analiza para determinar el flujo de trabajo normal y se examinan estructuras alter- nativas del proceso que podrían llevar a disminuir el tiempo o costo del desarrollo.

Una vez creado un proceso aceptable, se emplean otras herramientas de tecnología para asignar, vigilar e incluso controlar todas las actividades, acciones y tareas de la ingeniería de software definidas como parte del modelo del proceso. Cada miembro de un equipo de software utiliza dichas herramientas para desarrollar una lista de verificación de las tareas de trabajo que deben realizarse. La herramienta de tecnología del proceso también se usa para coordinar el empleo de otras herramientas de la ingeniería de software que sean apropiadas para una tarea particular del trabajo.

***Herramientas de modelado del proceso***

**Objetivo:** Si una organización trabaja para mejorar un proceso (o software) de negocios, primero debe entender-

lo. Las herramientas de modelado del proceso (también llamadas herramientas de *tecnología del proceso* o *de administración del pro- ceso*) se usan para representar los elementos clave de un proceso, de modo que se entienda mejor. Dichas herramientas también se relacio- nan con descripciones del proceso que ayudan a los involucrados a entender las acciones y tareas del trabajo que se requieren para lle- varlo a cabo. Las herramientas de modelado del proceso tienen víncu- los con otras que dan apoyo a las actividades del proceso definido.

**Mecánica:** Las herramientas en esta categoría permiten que un equipo defina los elementos de un modelo de proceso único (accio- nes, tareas, productos del trabajo, puntos de aseguramiento de la

**HERRAMIENTAS DE SOFTWARE**

calidad, etc.), dan una guía detallada acerca del contenido o descrip- ción de cada elemento del proceso, y después administran el proceso conforme se realiza. En ciertos casos, las herramientas de tecnología del proceso incorporan tareas estándar de administración de proyec- tos, tales como estimación, programación, seguimiento y control.



**Herramientas representativas:**25

*Igrafx Process Tools:* herramientas que permiten que un equipo mapee, mida y modele el proceso del software (**www.micro- grafx.com**)

*Adeptia BPM Server:* diseñado para administrar, automatizar y opti- mizar procesos de negocios (**www.adptia.com**)

*SpeedDev Suite:* conjunto de seis herramientas con mucho énfasis en las actividades de administración de la comunicación y modelado (**www.speedev.com**)

**2.8 PR ODUCTO Y PR OCESO**

Si el proceso es deficiente, no cabe duda de que el producto final sufrirá. Pero también es peli- grosa la dependencia excesiva del proceso. En un ensayo corto escrito hace muchos años, Margaret Davis [Dav95a] hace comentarios atemporales sobre la dualidad del producto y del proceso:

Cada diez años, más o menos, la comunidad del software redefine “el problema” por medio de cambiar su atención de aspectos del producto a aspectos del proceso. Así, hemos adoptado lenguajes de pro- gramación estructurada (producto) seguidos de métodos de análisis estructurados (proceso) que van seguidos por el encapsulamiento de datos (producto) a los que siguieron el énfasis actual en el modelo de madurez de la capacidad, del Instituto de Ingeniería de Software para el Desarrollo de Software (proceso) (seguido por métodos orientados a objetos, a los que sigue el desarrollo ágil de software).

En tanto que la tendencia natural de un péndulo es alcanzar el estado de reposo en el punto medio entre dos extremos, la atención de la comunidad del software cambia constantemente porque se aplica una nueva fuerza al fallar la última oscilación. Estos vaivenes son dañinos en sí mismos porque confunden al profesional promedio del software al cambiar en forma radical lo que significa hacer el trabajo bien. Los cambios periódicos no resuelven “el problema” porque están predestinados a fallar toda vez que el producto y el proceso son tratados como si fueran una dicotomía en lugar de una dualidad.

1. Las herramientas mencionadas aquí no son obligatorias, sino una muestra de las que hay en esta categoría. En la mayoría de casos, los nombres de las herramientas son marcas registradas por sus respectivos desarrolladores.

**52 PARTE UNO** EL PROCESO DEL SOFTWARE

En la comunidad científica existe el precedente de adoptar nociones de dualidad cuando las con- tradicciones en las observaciones no pueden ser explicadas por alguna teoría alternativa. La natura- leza dual de la luz, que parece ser al mismo tiempo onda y partícula, ha sido aceptada desde la década de 1920, cuando la propuso Louis de Broglie. Pienso que las observaciones que podemos hacer sobre el conjunto del software y su desarrollo demuestran una dualidad fundamental entre el producto y el proceso. Nunca es posible derivar u obtener todo el conjunto, su contexto, uso, significado y beneficios si se le ve sólo como proceso o sólo como producto…

Toda la actividad humana es un proceso, pero cada uno de nosotros obtiene un sentido de benefi- cio propio gracias a aquellas actividades que dan como resultado una representación o instancia que puede usar o apreciar más de una persona, utilizarla una y otra vez, o emplearla en algún otro con- texto no considerado. Es decir, obtenemos sentimientos de satisfacción por la reutilización de nues- tros productos, ya sea que lo hagamos nosotros u otras personas.

Entonces, si bien la rápida asimilación de las metas de reutilización en el desarrollo del software incrementa potencialmente la satisfacción que obtienen los profesionales del software en su trabajo, también aumenta la urgencia de la aceptación de la dualidad de producto y proceso. Pensar en un artefacto reutilizable como si fuera sólo un producto o sólo un proceso oscurece el contexto y las formas de emplearlo, o bien oculta el hecho de que cada uso da como resultado un producto que a su vez será utilizado como entrada para alguna otra actividad de desarrollo de software. Privilegiar un punto de vista sobre el otro reduce mucho las oportunidades para la reutilización y, por tanto, se pierde la oportunidad de aumentar la satisfacción por el trabajo.

La gente obtiene tanta (o más) satisfacción del proceso creativo como del producto final. Un artista disfruta las pinceladas tanto como el resultado que enmarca. Un escritor goza de la bús- queda de la metáfora apropiada tanto como del libro terminado. Como profesional creativo del software, usted también debe obtener tanta satisfacción del proceso como del producto final. La dualidad de producto y proceso es un elemento importante para hacer que personas creativas se involucren conforme la ingeniería de software evoluciona.

**2.9 RESUMEN**

Un modelo general del proceso para la ingeniería de software incluye un conjunto de actividades estructurales y sombrilla, acciones y tareas de trabajo. Cada uno de los modelos de proceso puede describirse por un flujo distinto del proceso: descripción de cómo se organizan secuencial y cronológicamente las actividades estructurales, acciones y tareas. Los patrones del proceso pueden utilizarse para resolver los problemas comunes que surgen como parte del proceso del software.

Los modelos de proceso prescriptivo se han aplicado durante muchos años en un esfuerzo por introducir orden y estructura al desarrollo de software. Cada uno de dichos modelos sugiere un flujo de proceso algo distinto, pero todos llevan a cabo el mismo conjunto de actividades estructurales generales: comunicación, planeación, modelado, construcción y desarrollo.

Los modelos de proceso secuencial, como el de la cascada y en V, son los paradigmas más antiguos del software. Sugieren un flujo lineal del proceso que con frecuencia no es congruente con las realidades modernas (cambio continuo, sistemas en evolución, plazos ajustados, etc.) del mundo del software. Sin embargo, tienen aplicación en situaciones en las que los requeri- mientos están bien definidos y son estables.

Los modelos de proceso incremental son de naturaleza iterativa y producen con mucha ra- pidez versiones funcionales del software. Los modelos de proceso evolutivo reconocen la natu- raleza iterativa e incremental de la mayoría de proyectos de ingeniería de software y están di- señados para aceptar los cambios. Los modelos evolutivos, tales como el de hacer prototipos y el espiral, generan rápido productos de trabajo incremental (o versiones funcionales del soft- ware). Estos modelos se adoptan para aplicarse a lo largo de todas las actividades de la inge-

**CAPÍTULO 2** MODELOS DEL PROCESO **53**

niería de software, desde el desarrollo del concepto hasta el mantenimiento del sistema a largo plazo.

El modelo de proceso concurrente permite que un equipo de software represente los elemen- tos iterativos y concurrentes de cualquier modelo de proceso. Los modelos especializados in- cluyen el basado en componentes, que pone el énfasis en la reutilización y ensamble de los componentes; el modelo de métodos formales consiste en un enfoque basado en matemáticas para desarrollar y verificar el software; y el modelo orientado a aspectos implica preocupaciones globales que afectan toda la arquitectura del sistema. El proceso unificado es un proceso del software diseñado como estructura para los métodos y herramientas del UML, y está “impulsado por el caso de uso, centrado en la arquitectura, y es iterativo e incremental”.

Se han propuesto modelos personal y del equipo para el proceso del software. Ambos enfa- tizan la medición, planeación y autodirección como los ingredientes clave para un proceso exitoso del software.

**PR OBLEMAS Y PUNTOS POR EVALUAR**

* 1. En la introducción de este capítulo, Baetjer afirma que: “El proceso genera interacción entre usuarios y diseñadores, entre usuarios y herramientas cambiantes [tecnología].” Enliste cinco preguntas que *a*) los di- señadores deben responder a los usuarios, *b*) los usuarios deben plantear a los diseñadores, *c*) los usuarios deben hacerse a sí mismos sobre el producto de software que ha de elaborarse, *d*) los diseñadores deben plantearse acerca del producto de software que va a construirse y del proceso que se usará para ello.
  2. Trate de desarrollar un conjunto de acciones para la actividad de comunicación. Seleccione una acción y defina un conjunto de tareas para ella.
  3. Un problema común durante la **comunicación** ocurre cuando se encuentra a dos participantes que tienen ideas en conflicto sobre lo que debe ser el software, es decir, que tienen requerimientos mutuamente conflictivos. Desarrolle un patrón del proceso (esto sería un patrón de la etapa) con el empleo de la plantilla presentada en la sección 2.1.3 que aborda este problema y sugiera un enfoque eficaz para él.
  4. Investigue un poco sobre el PPS y haga una breve presentación que describa los tipos de mediciones que se pide hacer a un ingeniero individual de software y la forma en la que pueden usarse para mejorar la eficacia personal.
  5. El uso de scripts (mecanismo requerido en el PES) no es apreciado de manera universal en la comuni- dad del software. Haga una lista de pros y contras en relación con los scripts y sugiera al menos dos situa- ciones en las que serían útiles, y otras dos en las que generarían menos beneficios.
  6. Lea a [Nog00] y escriba un ensayo de dos o tres páginas donde analice el efecto que tiene el “caos” en la ingeniería de software.
  7. Dé tres ejemplos de proyectos de software que podrían efectuarse con el modelo de cascada. Sea espe- cífico.
  8. Proporcione tres ejemplos de proyectos de software que podrían abordarse con el modelo de hacer prototipos. Sea específico.
  9. ¿Qué adaptaciones del proceso se requerirían si el proyecto evolucionara en un sistema o producto que se entregase?
  10. Diga tres ejemplos de proyectos de software que podrían realizarse con el modelo incremental. Sea específico.
  11. Conforme avanza hacia fuera por el flujo de proceso en espiral, ¿qué puede decirse sobre el software que se está desarrollando o que está en mantenimiento?
  12. ¿Es posible combinar modelos de proceso? Si es así, diga un ejemplo.
  13. El modelo de proceso concurrente define un conjunto de “estados”. Describa con sus propias palabras qué es lo que representan, y después indique cómo entran en juego dentro del modelo de proceso concu- rrente.

**54 PARTE UNO** EL PROCESO DEL SOFTWARE

* 1. ¿Cuáles son las ventajas y desventajas de desarrollar software en el que la calidad no es “suficiente- mente buena”? Es decir, ¿qué pasa cuando se pone el énfasis en la velocidad de desarrollo sobre la calidad del producto?
  2. Dé tres ejemplos de proyectos de software que serían abordables con el modelo basado en compo- nentes. Sea específico.
  3. ¿Es posible demostrar que un componente de software, o incluso un programa completo, es correcto? Entonces, ¿por qué no todos lo hacen?
  4. ¿Son lo mismo el proceso unificado y el UML? Explique su respuesta.

**LECTURAS ADICIONALES Y FUENTES DE INFOR MACIÓN**

La mayor parte de los libros de ingeniería de software consideran en detalle los modelos de proceso tradi- cionales. Libros como el de Sommerville (*Software Engineering*, 8a. ed., Addison-Wesley, 2006), Pfleeger y Atlee (*Software Engineering*, 3a. ed., Prentice-Hall, 2005), y Schach (*Object-Oriented and Classical Software Engineering*, 7a. ed., McGraw-Hill, 2006) consideran los paradigmas tradicionales y estudian sus fortalezas y debilidades. Glass (*Facts and Fallacies of Software Engineering*, Prentice-Hall, 2002) da un punto de vista prag- mático y crudo del proceso de ingeniería de software. Aunque no se dedica específicamente al proceso, Brooks (*The Mythical Man-Month*, 2a. ed., Addison-Wesley, 1995) presenta la sabiduría antigua sobre los proyectos y plantea que todo tiene que ver con el proceso.

Firesmith y Henderson-Sellers (*The OPEN Process Framework: An Introduction*, Addison-Wesley, 2001) presenta una plantilla general para crear “procesos de software flexibles pero con disciplina” y analiza los atributos y objetivos del proceso. Madachy (*Software Process Dynamics*, Wiley-IEEE, 2008) estudia técnicas de modelado que permiten analizar los elementos técnicos y sociales interrelacionados del proceso del soft- ware. Sharpe y McDermott (*Workflow Modeling: Tools for Process Improvement and Application Development*, Artech House, 2001) presentan herramientas para modelar procesos tanto de software como de negocios.

Lim (*Managing Software Reuse*, Prentice-Hall, 2004) estudia la reutilización desde la perspectiva del ge- rente. Ezran, Morisio y Tully (*Practical Software Reuse*, Springer, 2002) y Jacobson, Griss y Jonsson (*Software Reuse*, Addison-Wesley, 1997) presentan mucha información útil sobre el desarrollo basado en componentes. Heineman y Council (*Component-Based Software Engineering*, Addison-Wesley, 2001) describen el proceso requerido para implementar sistemas basados en componentes. Kenett y Baker (*Software Process Quality: Management and Control*, Marcel Dekker, 1999) analizan la manera en la que se conectan íntimamente la administración de la calidad y el diseño del proceso.

Nygard (*Release It!: Design and Deploy Production-Ready Software,* Pragmatic Bookshelf, 2007) y Richard- son y Gwaltney (*Ship it! A Practical Guide to Successful Software Projects*, Pragmatic Bookshelf, 2005) presentan una amplia colección de lineamientos útiles aplicables a la actividad de despliegue.

Además del libro fundamental de Jacobson, Rumbaugh y Booch acerca del proceso unificado [Jac99], los libros de Arlow y Neustadt (*UML 2 and the Unified Process,* Addison-Wesley, 2005), Kroll y Kruchten (*The Rational Unified Process Made Easy*, Addison-Wesley, 2003) y Farve (*UML and the Unified Process*, IRM Press, 2003) proveen información complementaria excelente. Gibbs (*Project Management with the IBM Rational Unified Process*, IBM Press, 2006) analiza la administración de proyectos dentro del contexto del PU.

En internet existe una amplia variedad de fuentes de información sobre la ingeniería de software y el proceso del software. En el sitio web del libro, [**www.mhhe.com/engcs/compsci/pressman/professio-**](http://www.mhhe.com/engcs/compsci/pressman/professio-) **nal/olc/ser.htm**, hay una lista actualizada de referencias en la Red Mundial que son relevantes para el proceso del software.