de software. La organización propia tiene cierto número de beneficios técnicos, pero, lo que es más importante, sirve para mejorar la colaboración y elevar la moral del equipo. En esencia, el equipo sirve como su propio gerente. Ken Schwaber [Sch02] aborda estos aspectos cuando escribe: "El equipo selecciona cuánto trabajo cree que puede realizar en cada iteración, y se compromete con la labor. Nada desmotiva tanto a un equipo como que alguien establezca compromisos por él. Nada motiva más a un equipo como aceptar la responsabilidad de cumplir los compromisos que haya hecho él mismo."

3.4 Programación extrema (XP)

A fin de ilustrar un proceso ágil con más detalle, daremos un panorama de la *programación extrema* (XP), el enfoque más utilizado del desarrollo de software ágil. Aunque las primeras actividades con las ideas y los métodos asociados a XP ocurrieron al final de la década de 1980, el trabajo fundamental sobre la materia había sido escrito por Kent Beck [Bec04a]. Una variante de XP llamada *XP industrial* [IXP] se propuso en una época más reciente [Ker05]. IXP mejora la XP y tiene como objetivo el proceso ágil para ser usado específicamente en organizaciones grandes.

3.4.1 Valores XP

Beck [Bec04a] define un conjunto de cinco *valores* que establecen el fundamento para todo trabajo realizado como parte de XP: comunicación, simplicidad, retroalimentación, valentía y respeto. Cada uno de estos valores se usa como un motor para actividades, acciones y tareas específicas de XP.

A fin de lograr la *comunicación* eficaz entre los ingenieros de software y otros participantes (por ejemplo, para establecer las características y funciones requeridas para el software), XP pone el énfasis en la colaboración estrecha pero informal (verbal) entre los clientes y los desarrolladores, en el establecimiento de metáforas[3] para comunicar conceptos importantes, en la retroalimentación continua y en evitar la documentación voluminosa como medio de comunicación.

CONSEJO

Mantenlo sencillo siempre que se pueda, pero reconoce que el "rediseño" continuo consume mucho tiempo y recursos.

Para alcanzar la *simplicidad*, XP restringe a los desarrolladores para que diseñen sólo para las necesidades inmediatas, en lugar de considerar las del futuro. El objetivo es crear un diseño sencillo que se implemente con facilidad en forma de código. Si hay que mejorar el diseño, se rediseñará[4] en un momento posterior.

La *retroalimentación* se obtiene de tres fuentes: el software implementado, el cliente y otros miembros del equipo de software. Al diseñar e implementar una estrategia de pruebas eficaz (capítulos 17 a 20), el software (por medio de los resultados de las pruebas) da retroalimentación al equipo ágil. XP usa la *prueba unitaria* como su táctica principal de pruebas. A medida que se desarrolla cada clase, el equipo implementa una prueba unitaria para ejecutar cada operación de acuerdo con su funcionalidad especificada. Cuando se entrega un incremento a un cliente, las *historias del usuario* o *casos de uso* (véase el capítulo 5) que se implementan con el incremento se utilizan como base para las pruebas de aceptación. El grado en el que el software implementa la salida, función y comportamiento del caso de uso es una forma de retroalimentación. Por último, conforme se obtienen nuevos requerimientos como parte de la planeación iterativa, el equipo da al cliente una retroalimentación rápida con miras al costo y al efecto en la programación de actividades.

3 En el contexto de XP, una *metáfora* es "una historia que cada quien —clientes, programadores y gerentes— narra, acerca de cómo funciona el sistema" [Bec04a].

4 El rediseño permite que un ingeniero mejore la estructura interna de un diseño (o código fuente) sin cambiar su funcionalidad o comportamiento externos. En esencia, el rediseño puede utilizarse para mejorar la eficiencia, disponibilidad o rendimiento de un diseño o del código que lo implementa.

Cita:

"XP es la respuesta a la pregunta: '¿Cuán pequeño podemos hacer un gran software?'."

Anónimo

Beck [Bec04a] afirma que la adhesión estricta a ciertas prácticas de XP requiere *valentía*. Un término más apropiado sería *disciplina*. Por ejemplo, es frecuente que haya mucha presión para diseñar requerimientos futuros. La mayor parte de equipos de software sucumben a ella y se justifican porque "diseñar para el mañana" ahorrará tiempo y esfuerzo en el largo plazo. Un equipo XP ágil debe tener la disciplina (valentía) para diseñar para hoy y reconocer que los requerimientos futuros tal vez cambien mucho, por lo que demandarán repeticiones sustanciales del diseño y del código implementado.

Al apegarse a cada uno de estos valores, el equipo ágil inculca *respeto* entre sus miembros, entre otros participantes y los integrantes del equipo, e indirectamente para el software en sí mismo. Conforme logra la entrega exitosa de incrementos de software, el equipo desarrolla más respeto para el proceso XP.

3.4.2 El proceso XP

WebRef

En la dirección **www.extremeprogramming.org/rules.html**, se encuentra un panorama excelente de las "reglas" de XP.

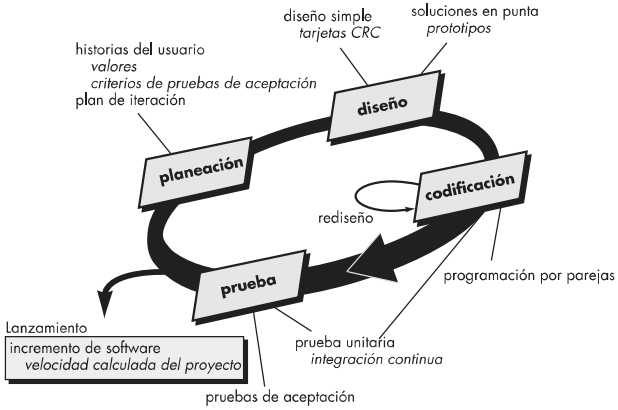
La programación extrema usa un enfoque orientado a objetos (véase el apéndice 2) como paradigma preferido de desarrollo, y engloba un conjunto de reglas y prácticas que ocurren en el contexto de cuatro actividades estructurales: planeación, diseño, codificación y pruebas. La figura 3.2 ilustra el proceso XP y resalta algunas de las ideas y tareas clave que se asocian con cada actividad estructural. En los párrafos que siguen se resumen las actividades de XP clave.

¿Qué es una "historia" XP?

Planeación. La actividad de planeación (también llamada *juego de planeación*) comienza *escuchando* —actividad para recabar requerimientos que permite que los miembros técnicos del equipo XP entiendan el contexto del negocio para el software y adquieran la sensibilidad de la salida y características principales y funcionalidad que se requieren—. Escuchar lleva a la creación de algunas "historias" (también llamadas *historias del usuario*) que describen la salida necesaria, características y funcionalidad del software que se va a elaborar. Cada *historia* (similar a los casos de uso descritos en el capítulo 5) es escrita por el cliente y colocada en una tarjeta indizada. El cliente asigna un *valor* (es decir, una prioridad) a la historia con base en el valor general de la característica o función para el negocio.[5] Después, los miembros del equipo XP

FIGURA 3.2

El proceso de la programación extrema



5 El valor de una historia también puede depender de la presencia de otra historia.

evalúan cada historia y le asignan un costo, medido en semanas de desarrollo. Si se estima que la historia requiere más de tres semanas de desarrollo, se pide al cliente que la descomponga en historias más chicas y de nuevo se asigna un valor y costo. Es importante observar que en cualquier momento es posible escribir nuevas historias.

Los clientes y desarrolladores trabajan juntos para decidir cómo agrupar las historias en la siguiente entrega (el siguiente incremento de software) que desarrollará el equipo XP. Una vez que se llega a un *compromiso* sobre la entrega (acuerdo sobre las historias por incluir, la fecha de entrega y otros aspectos del proyecto), el equipo XP ordena las historias que serán desarrolladas en una de tres formas: 1) todas las historias se implementarán de inmediato (en pocas semanas), 2) las historias con más valor entrarán a la programación de actividades y se implementarán en primer lugar o 3) las historias más riesgosas formarán parte de la programación de actividades y se implementarán primero.

WebRef
En la dirección **c2.com/cgi/wiki?planningGame**, se halla un "juego de planeación" XP provechoso.

Después de la primera entrega del proyecto (también llamada incremento de software), el equipo XP calcula la velocidad de éste. En pocas palabras, la *velocidad del proyecto* es el número de historias de los clientes implementadas durante la primera entrega. La velocidad del proyecto se usa para: 1) ayudar a estimar las fechas de entrega y programar las actividades para las entregas posteriores, y 2) determinar si se ha hecho un gran compromiso para todas las historias durante todo el desarrollo del proyecto. Si esto ocurre, se modifica el contenido de las entregas o se cambian las fechas de entrega final.

PUNTO CLAVE
La velocidad del proyecto es una medición sutil de la productividad del equipo.

A medida que avanza el trabajo, el cliente puede agregar historias, cambiar el valor de una ya existente, descomponerlas o eliminarlas. Entonces, el equipo XP reconsidera todas las entregas faltantes y modifica sus planes en consecuencia.

Diseño. El diseño XP sigue rigurosamente el principio MS (mantenlo sencillo). Un diseño sencillo siempre se prefiere sobre una representación más compleja. Además, el diseño guía la implementación de una historia conforme se escribe: nada más y nada menos. Se desalienta el diseño de funcionalidad adicional porque el desarrollador supone que se requerirá después.[6]

XP estimula el uso de las tarjetas CRC (véase el capítulo 7) como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos. Las tarjetas CRC (clase-responsabilidad-colaborador) identifican y organizan las clases orientadas a objetos[7] que son relevantes para el incremento actual de software. El equipo XP dirige el ejercicio de diseño con el uso de un proceso similar al que se describe en el capítulo 8. Las tarjetas CRC son el único producto del trabajo de diseño que se genera como parte del proceso XP.

CONSEJO
XP desalienta la importancia del diseño, opinión con la que no todos están de acuerdo. En realidad, hay veces en las que debe hacerse énfasis en el diseño.

Si en el diseño de una historia se encuentra un problema de diseño difícil, XP recomienda la creación inmediata de un prototipo operativo de esa porción del diseño. Entonces, se implementa y evalúa el prototipo del diseño, llamado *solución en punta*. El objetivo es disminuir el riesgo cuando comience la implementación verdadera y validar las estimaciones originales para la historia que contiene el problema de diseño.

En la sección anterior se dijo que XP estimula el *rediseño*, técnica de construcción que también es un método para la optimización del diseño. Fowler [Fow00] describe el rediseño del modo siguiente:

> *Rediseño* es el proceso mediante el cual se cambia un sistema de software en forma tal que no altere el comportamiento externo del código, pero sí mejore la estructura interna. Es una manera disciplinada de limpiar el código [y modificar o simplificar el diseño interno] que minimiza la probabilidad de introducir errores. En esencia, cuando se rediseña, se mejora el diseño del código después de haber sido escrito.

WebRef
En la dirección **www.refactoring.com** se encuentran técnicos y herramientas de rediseño.

6 Estos lineamientos de diseño deben seguirse en todo método de ingeniería de software, aunque hay ocasiones en los que la notación y terminología sofisticadas del diseño son un camino hacia la simplicidad.

7 Las clases orientadas a objetos se estudian en el apéndice 2, en el capítulo 8 y en toda la parte 2 de este libro.

PUNTO CLAVE
El rediseño mejora la estructura interna de un diseño (o código fuente) sin cambiar su funcionalidad o comportamiento externo.

Como el diseño XP virtualmente no utiliza notación y genera pocos, si alguno, productos del trabajo que no sean tarjetas CRC y soluciones en punta, el diseño es visto como un artefacto en transición que puede y debe modificarse continuamente a medida que avanza la construcción. El objetivo del rediseño es controlar dichas modificaciones, sugiriendo pequeños cambios en el diseño que "son capaces de mejorarlo en forma radical" [Fow00]. Sin embargo, debe notarse que el esfuerzo que requiere el rediseño aumenta en forma notable con el tamaño de la aplicación.

Un concepto central en XP es que el diseño ocurre tanto antes *como después* de que comienza la codificación. Rediseñar significa que el diseño se hace de manera continua conforme se construye el sistema. En realidad, la actividad de construcción en sí misma dará al equipo XP una guía para mejorar el diseño.

Codificación. Después de que las historias han sido desarrolladas y de que se ha hecho el trabajo de diseño preliminar, el equipo *no* inicia la codificación, sino que desarrolla una serie de pruebas unitarias a cada una de las historias que se van a incluir en la entrega en curso (incremento de software).[8] Una vez creada la prueba unitaria,[9] el desarrollador está mejor capacitado para centrarse en lo que debe implementarse para pasar la prueba. No se agrega nada extraño (MS). Una vez que el código está terminado, se le aplica de inmediato una prueba unitaria, con lo que se obtiene retroalimentación instantánea para los desarrolladores.

WebRef
Hay información útil acerca de XP en la dirección **www.xprogramming.com.**

¿Qué es la programación por parejas?

Un concepto clave durante la actividad de codificación (y uno de los aspectos del que más se habla en la XP) es la *programación por parejas*. XP recomienda que dos personas trabajen juntas en una estación de trabajo con el objeto de crear código para una historia. Esto da un mecanismo para la solución de problemas en tiempo real (es frecuente que dos cabezas piensen más que una) y para el aseguramiento de la calidad también en tiempo real (el código se revisa conforme se crea). También mantiene a los desarrolladores centrados en el problema de que se trate. En la práctica, cada persona adopta un papel un poco diferente. Por ejemplo, una de ellas tal vez piense en los detalles del código de una porción particular del diseño, mientras la otra se asegura de que se siguen los estándares de codificación (parte necesaria de XP) o de que el código para la historia satisfará la prueba unitaria desarrollada a fin de validar el código confrontándolo con la historia.

CONSEJO
Muchos equipos de software están llenos de individualistas. Si la programación por parejas ha de funcionar con eficacia, tendrá que trabajar para cambiar esa cultura.

A medida que las parejas de programadores terminan su trabajo, el código que desarrollan se integra con el trabajo de los demás. En ciertos casos, esto lo lleva a cabo a diario un equipo de integración. En otros, las parejas de programadores tienen la responsabilidad de la integración. Esta estrategia de "integración continua" ayuda a evitar los problemas de compatibilidad e interfaces y brinda un ambiente de "prueba de humo" (véase el capítulo 17) que ayuda a descubrir a tiempo los errores.

¿Cómo se usan las pruebas unitarias en XP?

Pruebas. Ya se dijo que la creación de pruebas unitarias antes de que comience la codificación es un elemento clave del enfoque de XP. Las pruebas unitarias que se crean deben implementarse con el uso de una estructura que permita automatizarlas (de modo que puedan ejecutarse en repetidas veces y con facilidad). Esto estimula una estrategia de pruebas de regresión (véase el capítulo 17) siempre que se modifique el código (lo que ocurre con frecuencia, dada la filosofía del rediseño en XP).

A medida que se organizan las pruebas unitarias individuales en un "grupo de prueba universal" [Wel99], las pruebas de la integración y validación del sistema pueden efectuarse a diario. Esto da al equipo XP una indicación continua del avance y también lanza señales de alerta si las

8 Este enfoque es equivalente a saber las preguntas del examen antes de comenzar a estudiar. Vuelve mucho más fácil el estudio porque centra la atención sólo en las preguntas que se van a responder.

9 La prueba unitaria, que se estudia en detalle en el capítulo 17, se centra en un componente de software individual sobre interfaz, estructuras de datos y funcionalidad del componente, en un esfuerzo por descubrir errores locales del componente.

cosas marchan mal. Wells [Wel99] dice: "Corregir pequeños problemas cada cierto número de horas toma menos tiempo que resolver problemas enormes justo antes del plazo final."

PUNTO CLAVE
Las pruebas de aceptación se derivan de las historias de los usuarios.

Las *pruebas de aceptación* XP, también llamadas *pruebas del cliente*, son especificadas por el cliente y se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. Las pruebas de aceptación se derivan de las historias de los usuarios que se han implementado como parte de la liberación del software.

3.4.3 XP industrial

Joshua Kerievsky [Ker05] describe la *programación extrema industrial* [IXP, por sus siglas en inglés] en la forma siguiente: "IXP es la evolución orgánica de XP. Está imbuida del espíritu minimalista, centrado en el cliente y orientado a las pruebas que tiene XP. IXP difiere sobre todo de la XP original en su mayor inclusión de la gerencia, el papel más amplio de los clientes y en sus prácticas técnicas actualizadas". IXP incorpora seis prácticas nuevas diseñadas para ayudar a garantizar que un proyecto XP funciona con éxito para proyectos significativos dentro de una organización grande.

? ¿Qué nuevas prácticas se agregan a XP para crear IXP?

Evaluación de la factibilidad. Antes de iniciar un proyecto IXP, la organización debe efectuar una *evaluación de la factibilidad*. Ésta deja en claro si: 1) existe un ambiente apropiado de desarrollo que acepte IXP, 2) el equipo estará constituido por los participantes adecuados, 3) la organización tiene un programa de calidad distintivo y apoya la mejora continua, 4) la cultura organizacional apoyará los nuevos valores de un equipo ágil, y 5) la comunidad extendida del proyecto estará constituida de modo apropiado.

Comunidad del proyecto. La XP clásica sugiere que se utilice personal apropiado para formar el equipo ágil a fin de asegurar el éxito. La implicación es que las personas en el equipo deben estar bien capacitadas, ser adaptables y hábiles, y tener el temperamento apropiado para contribuir al equipo con organización propia. Cuando se aplica XP a un proyecto significativo en una organización grande, el concepto de "equipo" debe adoptar la forma de *comunidad*. Una comunidad puede tener un tecnólogo y clientes que son fundamentales para el éxito del proyecto, así como muchos otros participantes (equipo jurídico; auditores de calidad, de tipos de manufactura o de ventas, etc.) que "con frecuencia se encuentran en la periferia en un proyecto IXP, pero que desempeñan en éste papeles importantes" [Ker05]. En IXP, los miembros de la comunidad y sus papeles deben definirse de modo explícito, así como establecer los mecanismos para la comunicación y coordinación entre los integrantes de la comunidad.

Cita:

"Habilidad es aquello que eres capaz de hacer. La motivación determina lo que haces. La actitud determina cuán bien lo haces."

Lou Holtz

Calificación del proyecto. El equipo de IXP evalúa el proyecto para determinar si existe una justificación apropiada de negocios y si el proyecto cumplirá las metas y objetivos generales de la organización. La calificación también analiza el contexto del proyecto a fin de determinar cómo complementa, extiende o reemplaza sistemas o procesos existentes.

Administración orientada a pruebas. Un proyecto IXP requiere criterios medibles para evaluar el estado del proyecto y el avance realizado. La administración orientada a pruebas establece una serie de "destinos" medibles [Ker05] y luego define los mecanismos para determinar si se han alcanzado o no éstos.

Retrospectivas. Después de entregar un incremento de software, el equipo XP realiza una revisión técnica especializada que se llama *retrospectiva* y que examina "los temas, eventos y lecciones aprendidas" [Ker05] a lo largo del incremento de software y/o de la liberación de todo el software. El objetivo es mejorar el proceso IXP.

Aprendizaje continuo. Como el aprendizaje es una parte vital del proceso de mejora continua, los miembros del equipo XP son invitados (y tal vez incentivados) a aprender nuevos métodos y técnicas que conduzcan a una calidad más alta del producto.

Además de las seis nuevas prácticas analizadas, IXP modifica algunas de las existentes en XP. El *desarrollo impulsado por la historia* (DIH) insiste en que las historias de las pruebas de aceptación se escriban antes de generar una sola línea de código. El *diseño impulsado por el dominio* (DID) es una mejora sobre el concepto de la "metáfora del sistema" usado en XP. El DID [Eva03] sugiere la creación evolutiva de un modelo de dominio que "represente con exactitud cómo piensan los expertos del dominio en su materia" [Ker05]. La *formación de parejas* amplía el concepto de programación en pareja para que incluya a los gerentes y a otros participantes. El objetivo es mejorar la manera de compartir conocimientos entre los integrantes del equipo XP que no estén directamente involucrados en el desarrollo técnico. La *usabilidad iterativa* desalienta el diseño de una interfaz cargada al frente y estimula un diseño que evoluciona a medida que se liberan los incrementos de software y que se estudia la interacción de los usuarios con el software.

La IXP hace modificaciones más pequeñas a otras prácticas XP y redefine ciertos roles y responsabilidades para hacerlos más asequibles a proyectos significativos de las organizaciones grandes. Para mayores detalles de IXP, visite el sitio **http://industrialxp.org**.

3.4.4 El debate XP

Los nuevos modelos y métodos de proceso han motivado análisis provechosos y en ciertas instancias debates acalorados. La programación extrema desencadena ambos. En un libro interesante que examina la eficacia de XP, Stephens y Rosenberg [Ste03] afirman que muchas prácticas de XP son benéficas, pero que otras están sobreestimadas y unas más son problemáticas. Los autores sugieren que la naturaleza codependiente de las prácticas de XP constituye tanto su fortaleza como su debilidad. Debido a que muchas organizaciones adoptan sólo un subconjunto de prácticas XP, debilitan la eficacia de todo el proceso. Los defensores contradicen esto al afirmar que la XP está en evolución continua y que muchas de las críticas que se le hacen han llevado a correcciones conforme maduran sus prácticas. Entre los aspectos que destacan algunos críticos de la XP están los siguientes:[10]

? ¿Cuáles son algunos de los aspectos que llevan al debate de XP?

- *Volatilidad de los requerimientos.* Como el cliente es un miembro activo del equipo XP, los cambios a los requerimientos se solicitan de manera informal. En consecuencia, el alcance del proyecto cambia y el trabajo inicial tiene que modificarse para dar acomodo a las nuevas necesidades. Los defensores afirman que esto pasa sin importar el proceso que se aplique y que la XP proporciona mecanismos para controlar los vaivenes del alcance.
- *Necesidades conflictivas del cliente.* Muchos proyectos tienen clientes múltiples, cada uno con sus propias necesidades. En XP, el equipo mismo tiene la tarea de asimilar las necesidades de distintos clientes, trabajo que tal vez esté más allá del alcance de su autoridad.
- *Los requerimientos se expresan informalmente.* Las historias de usuario y las pruebas de aceptación son la única manifestación explícita de los requerimientos en XP. Los críticos afirman que es frecuente que se necesite un modelo o especificación más formal para garantizar que se descubran las omisiones, inconsistencias y errores antes de que se construya el sistema. Los defensores contraatacan diciendo que la naturaleza cambiante de los requerimientos vuelve obsoletos esos modelos y especificaciones casi tan pronto como se desarrollan.
- *Falta de un diseño formal.* XP desalienta la necesidad del diseño de la arquitectura y, en muchas instancias, sugiere que el diseño de todas las clases debe ser relativamente informal. Los críticos argumentan que cuando se construyen sistemas complejos, debe ponerse el énfasis en el diseño con el objeto de garantizar que la estructura general del software tendrá calidad y que será susceptible de recibir mantenimiento. Los defensores

10 Para un estudio más detallado de ciertas críticas profundas hechas a XP, visite www.softwarereality.com/ExtremeProgramming.jsp.

de XP sugieren que la naturaleza incremental del proceso XP limita la complejidad (la sencillez es un valor fundamental), lo que reduce la necesidad de un diseño extenso.

El lector debe observar que todo proceso del software tiene sus desventajas, y que muchas organizaciones de software han utilizado con éxito la XP. La clave es identificar dónde tiene sus debilidades un proceso y adaptarlo a las necesidades de la organización.

3.5 Otros modelos ágiles de proceso

Cita:

"Nuestra profesión pasa por las metodologías como un chico de 14 años pasa por la ropa."

Stephen Hawrysh y Jim Ruprecht

La historia de la ingeniería de software está salpicada de decenas de descripciones y metodologías de proceso, métodos de modelado y notaciones, herramientas y tecnología, todos ellos obsoletos. Cada uno tuvo notoriedad y luego fue eclipsado por algo nuevo y (supuestamente) mejor. Con la introducción de una amplia variedad de modelos ágiles del proceso —cada uno en lucha por la aceptación de la comunidad de desarrollo de software— el movimiento ágil está siguiendo la misma ruta histórica.[11]

CasaSegura

Consideración del desarrollo ágil de software

La escena: Oficina de Doug Miller.

Participantes: Doug Miller, gerente de ingeniería de software; Jamie Lazar, miembro del equipo de software; Vinod Raman, integrante del equipo de software.

La conversación:

(Tocan a la puerta; Jamie y Vinod entran a la oficina de Doug.)

Jamie: Doug, ¿tienes un minuto?

Doug: Seguro, Jamie, ¿qué pasa?

Jamie: Hemos estado pensando en nuestra conversación de ayer sobre el proceso… ya sabes, el que vamos a elegir para este nuevo proyecto de *CasaSegura*.

Doug: ¿Y?

Vinod: Hablé con un amigo de otra compañía, y me contó sobre la programación extrema. Es un modelo de proceso ágil… ¿has oído de él?

Doug: Sí, algunas cosas buenas y otras malas.

Jamie: Bueno, a nosotros nos pareció bien. Permite el desarrollo de software realmente rápido, usa algo llamado programación en parejas para revisar la calidad en tiempo real… Pienso que es muy bueno.

Doug: Tiene muchas ideas realmente buenas. Por ejemplo, me gusta el concepto de programación en parejas y la idea de que los participantes deben formar parte del equipo.

Jamie: ¿Qué? ¿Quieren decir que mercadotecnia trabajará con nosotros en el proyecto?

Doug (afirmando con la cabeza): Ellos son uno de los participantes, ¿o no?

Jamie: Sí… Pedirán cambios cada cinco minutos.

Vinod: No necesariamente. Mi amigo dijo que hay formas de "adoptar" los cambios durante un proyecto de XP.

Doug: Entonces, chicos, ¿piensan que debemos usar XP?

Jamie: Definitivamente, sería bueno considerarlo.

Doug: Estoy de acuerdo. E incluso si elegimos un modelo incremental como nuestro enfoque, no hay razón para no incorporar mucho de lo que XP tiene que ofrecer.

Vinod: Doug, dijiste hace un rato "cosas buenas y malas". ¿Cuáles son las malas?

Doug: Lo que no me gusta es la forma en la que XP desprecia el análisis y el diseño… algo así como decir que la escritura del código está donde hay acción…

(Los miembros del equipo se miran entre sí y sonríen.)

Doug: Entonces, ¿están de acuerdo con el enfoque XP?

Jamie (habla por ambos): ¡Escribir código es lo que hacemos, jefe!

Doug (ríe): Es cierto, pero me gustaría ver que dediquen un poco menos de tiempo a escribir código y luego a repetirlo, y que pasen algo más de tiempo en el análisis de lo que debe hacerse para diseñar una solución que funcione.

Vinod: Tal vez tengamos las dos cosas, agilidad con un poco de disciplina.

Doug: Creo que podemos, Vinod. En realidad, estoy seguro de que se puede.

11 Esto no es algo malo. Antes de que uno o varios modelos se acepten como el estándar *de facto*, todos deben luchar por conquistar las mentes y corazones de los ingenieros de software. Los "ganadores" evolucionan hacia las mejores prácticas, mientras que los "perdedores" desaparecen o se funden con los modelos ganadores.

Como se dijo en la sección anterior, el más usado de todos los modelos ágiles de proceso es la programación extrema (XP). Pero se han propuesto muchos otros y están en uso en toda la industria. Entre ellos se encuentran los siguientes:

- Desarrollo adaptativo de software (DAS)
- Scrum
- Método de desarrollo de sistemas dinámicos (MDSD)
- Cristal
- Desarrollo impulsado por las características (DIC)
- Desarrollo esbelto de software (DES)
- Modelado ágil (MA)
- Proceso unificado ágil (PUA)

En las secciones que siguen se presenta un panorama muy breve de cada uno de estos modelos ágiles del proceso. Es importante notar que *todos* los modelos de proceso ágil se apegan (en mayor o menor grado) al *Manifiesto para el desarrollo ágil de software* y a los principios descritos en la sección 3.3.1. Para mayores detalles, consulte las referencias mencionadas en cada subsección o ingrese en la entrada "desarrollo de software ágil" de Wikipedia.[12]

3.5.1 Desarrollo adaptativo de software (DAS)

WebRef

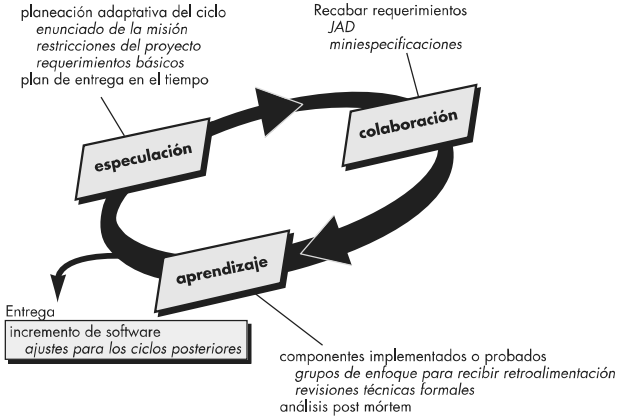
En la dirección **www.adaptivesd.com** hay referencias útiles sobre el DAS.

El *desarrollo adaptativo de software* (DAS) fue propuesto por Jim Highsmith [Hig00] como una técnica para elaborar software y sistemas complejos. Los fundamentos filosóficos del DAS se centran en la colaboración humana y en la organización propia del equipo.

Highsmith argumenta que un enfoque de desarrollo adaptativo basado en la colaboración es "tanto una fuente de *orden* en nuestras complejas interacciones, como de disciplina e ingeniería". Él define un "ciclo de vida" del DAS (véase la figura 3.3) que incorpora tres fases: especulación, colaboración y aprendizaje.

Figura 3.3

Desarrollo adaptativo de software



12 Consulte http://en.wikipedia.org/wiki/Agile_software_development#Agile_methods.

Durante la *especulación*, se inicia el proyecto y se lleva a cabo la *planeación adaptativa del ciclo*. La especulación emplea la información de inicio del proyecto —enunciado de misión de los clientes, restricciones del proyecto (por ejemplo, fechas de entrega o descripciones de usuario) y requerimientos básicos— para definir el conjunto de ciclos de entrega (incrementos de software) que se requerirán para el proyecto.

CONSEJO

La colaboración eficaz con el cliente sólo ocurrirá si evita cualquier actitud del tipo "nosotros y ellos".

No importa lo completo y previsor que sea el plan del ciclo, será inevitable que cambie. Con base en la información obtenida al terminar el primer ciclo, el plan se revisa y se ajusta, de modo que el trabajo planeado se acomode mejor a la realidad en la que trabaja el equipo DAS.

Las personas motivadas usan la *colaboración* de manera que multiplica su talento y producción creativa más allá de sus números absolutos. Este enfoque es un tema recurrente en todos los métodos ágiles. Sin embargo, la colaboración no es fácil. Incluye la comunicación y el trabajo en equipo, pero también resalta el individualismo porque la creatividad individual desempeña un papel importante en el pensamiento colaborativo. Es cuestión, sobre todo, de confianza. Las personas que trabajan juntas deben confiar una en otra a fin de: 1) criticarse sin enojo, 2) ayudarse sin resentimiento, 3) trabajar tan duro, o más, que como de costumbre, 4) tener el conjunto de aptitudes para contribuir al trabajo, y 5) comunicar los problemas o preocupaciones de manera que conduzcan a la acción efectiva.

PUNTO CLAVE

El DAS pone el énfasis en el aprendizaje como elemento clave para lograr un equipo con "organización propia".

Conforme los miembros de un equipo DAS comienzan a desarrollar los componentes que forman parte de un ciclo adaptativo, el énfasis se traslada al "aprendizaje" de todo lo que hay en el avance hacia la terminación del ciclo. En realidad, Highsmith [Hig00] afirma que los desarrolladores de software sobreestiman con frecuencia su propia comprensión (de la tecnología, del proceso y del proyecto) y que el aprendizaje los ayudará a mejorar su nivel de entendimiento real. Los equipos DAS aprenden de tres maneras: grupos de enfoque (véase el capítulo 5), revisiones técnicas (véase el capítulo 14) y análisis post mórtem del proyecto.

La filosofía DAS tiene un mérito, sin importar el modelo de proceso que se use. El énfasis general que hace el DAS en la dinámica de los equipos con organización propia, la colaboración interpersonal y el aprendizaje individual y del equipo generan equipos para proyectos de software que tienen una probabilidad de éxito mucho mayor.

3.5.2 Scrum

Scrum (nombre que proviene de cierta jugada que tiene lugar durante un partido de rugby)[13] es un método de desarrollo ágil de software concebido por Jeff Sutherland y su equipo de desarrollo a principios de la década de 1990. En años recientes, Schwaber y Beedle [Sch01a] han desarrollado más los métodos Scrum.

WebRef

En la dirección **www.controlchaos.com** hay información útil sobre Scrum.

Los principios Scrum son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis que incorpora las siguientes actividades estructurales: requerimientos, análisis, diseño, evolución y entrega. Dentro de cada actividad estructural, las tareas del trabajo ocurren con un patrón del proceso (que se estudia en el párrafo siguiente) llamado *sprint*. El trabajo realizado dentro de un sprint (el número de éstos que requiere cada actividad estructural variará en función de la complejidad y tamaño del producto) se adapta al problema en cuestión y se define —y con frecuencia se modifica— en tiempo real por parte del equipo Scrum. El flujo general del proceso Scrum se ilustra en la figura 3.4.

PUNTO CLAVE

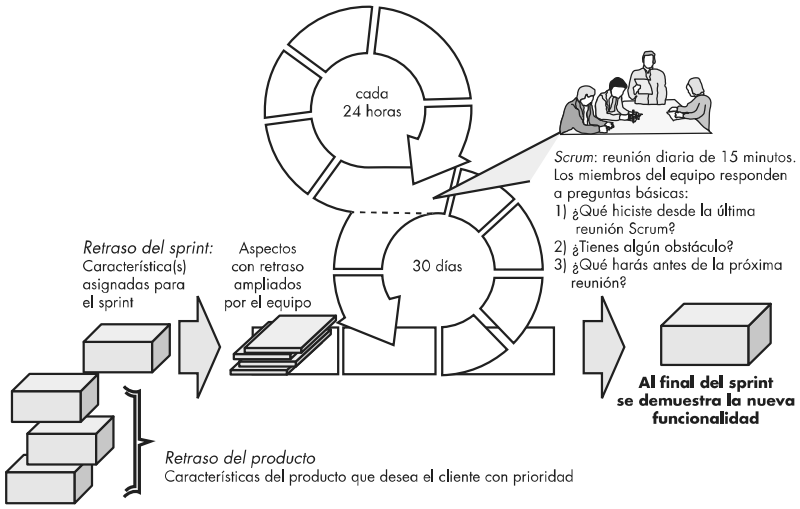
Scrum incorpora un conjunto de patrones del proceso que ponen el énfasis en las prioridades del proyecto, las unidades de trabajo agrupadas, la comunicación y la retroalimentación frecuente con el cliente.

Scrum acentúa el uso de un conjunto de patrones de proceso del software [Noy02] que han demostrado ser eficaces para proyectos con plazos de entrega muy apretados, requerimientos cambiantes y negocios críticos. Cada uno de estos patrones de proceso define un grupo de acciones de desarrollo:

13 Se forma un grupo de jugadores alrededor del balón y todos trabajan juntos (a veces con violencia) para moverlo a través del campo.

FIGURA 3.4

Flujo del proceso Scrum



Retraso: lista de prioridades de los requerimientos o características del proyecto que dan al cliente un valor del negocio. Es posible agregar en cualquier momento otros aspectos al retraso (ésta es la forma en la que se introducen los cambios). El gerente del proyecto evalúa el retraso y actualiza las prioridades según se requiera.

Sprints: consiste en unidades de trabajo que se necesitan para alcanzar un requerimiento definido en el retraso que debe ajustarse en una caja de tiempo[14] predefinida (lo común son 30 días). Durante el sprint no se introducen cambios (por ejemplo, aspectos del trabajo retrasado). Así, el sprint permite a los miembros del equipo trabajar en un ambiente de corto plazo pero estable.

Reuniones Scrum: son reuniones breves (de 15 minutos, por lo general) que el equipo Scrum efectúa a diario. Hay tres preguntas clave que se pide que respondan todos los miembros del equipo [Noy02]:

- ¿Qué hiciste desde la última reunión del equipo?
- ¿Qué obstáculos estás encontrando?
- ¿Qué planeas hacer mientras llega la siguiente reunión del equipo?

Un líder del equipo, llamado *maestro Scrum*, dirige la junta y evalúa las respuestas de cada persona. La junta Scrum ayuda al equipo a descubrir los problemas potenciales tan pronto como sea posible. Asimismo, estas juntas diarias llevan a la "socialización del conocimiento" [Bee99], con lo que se promueve una estructura de equipo con organización propia.

Demostraciones preliminares: entregar el incremento de software al cliente de modo que la funcionalidad que se haya implementado pueda demostrarse al cliente y éste pueda evaluarla.

14 Una *caja de tiempo* es un término de la administración de proyectos (véase la parte 4 de este libro) que indica el tiempo que se ha asignado para cumplir alguna tarea.

Es importante notar que las demostraciones preliminares no contienen toda la funcionalidad planeada, sino que éstas se entregarán dentro de la caja de tiempo establecida.

Beedle y sus colegas [Bee99] presentan un análisis exhaustivo de estos patrones en el que dicen: "Scrum supone de entrada la existencia de caos..." Los patrones de proceso Scrum permiten que un equipo de software trabaje con éxito en un mundo en el que es imposible eliminar la incertidumbre.

3.5.3 Método de desarrollo de sistemas dinámicos (MDSD)

WebRef
En la dirección **www.dsdm.org** hay recursos útiles para el MDSD.

El *método de desarrollo de sistemas dinámicos* (MDSD) [Sta97] es un enfoque de desarrollo ágil de software que "proporciona una estructura para construir y dar mantenimiento a sistemas que cumplan restricciones apretadas de tiempo mediante la realización de prototipos incrementales en un ambiente controlado de proyectos" [CCS02]. La filosofía MDSD está tomada de una versión modificada de la regla de Pareto: 80 por ciento de una aplicación puede entregarse en 20 por ciento del tiempo que tomaría entregarla completa (100 por ciento).

El MDSD es un proceso iterativo de software en el que cada iteración sigue la regla de 80 por ciento. Es decir, se requiere sólo suficiente trabajo para cada incremento con objeto de facilitar el paso al siguiente. Los detalles restantes se terminan más tarde, cuando se conocen los requerimientos del negocio y se han pedido y efectuado cambios.

El grupo DSDM Consortium (**www.dsdm.org**) es un conglomerado mundial de compañías que adoptan colectivamente el papel de "custodios" del método. El consorcio ha definido un modelo de proceso ágil, llamado *ciclo de vida MDSD*, que define tres ciclos iterativos distintos, precedidos de dos actividades adicionales al ciclo de vida:

Estudio de factibilidad: establece los requerimientos y restricciones básicas del negocio, asociados con la aplicación que se va a construir, para luego evaluar si la aplicación es un candidato viable para aplicarle el proceso MDSD.

PUNTO CLAVE
El MDSD es una estructura de proceso que adopta las tácticas de otro enfoque ágil, como XP.

Estudio del negocio: establece los requerimientos e información funcionales que permitirán la aplicación para dar valor al negocio; asimismo, define la arquitectura básica de la aplicación e identifica los requerimientos para darle mantenimiento.

Iteración del modelo funcional: produce un conjunto de prototipos incrementales que demuestran al cliente la funcionalidad. (Nota: todos los prototipos de MDSD están pensados para que evolucionen hacia la aplicación que se entrega.) El objetivo de este ciclo iterativo es recabar requerimientos adicionales por medio de la obtención de retroalimentación de los usuarios cuando practican con el prototipo.

Diseño e iteración de la construcción: revisita los prototipos construidos durante la *iteración del modelo funcional* a fin de garantizar que en cada iteración se ha hecho ingeniería en forma que permita dar valor operativo del negocio a los usuarios finales; la *iteración del modelo funcional* y el *diseño e iteración de la construcción* ocurren de manera concurrente.

Implementación: coloca el incremento más reciente del software (un prototipo "operacional") en el ambiente de operación. Debe notarse que: 1) el incremento tal vez no sea el de 100% final, o 2) quizá se pidan cambios cuando el incremento se ponga en su lugar. En cualquier caso, el trabajo de desarrollo MDSD continúa y vuelve a la actividad de iteración del modelo funcional.

El MDSD se combina con XP (véase la sección 3.4) para dar un enfoque de combinación que define un modelo sólido del proceso (ciclo de vida MDSD) con las prácticas detalladas (XP) que se requieren para elaborar incrementos de software. Además, los conceptos DAS se adaptan a un modelo combinado del proceso.

3.5.4 Cristal

PUNTO CLAVE
Cristal es una familia de modelos de proceso con el mismo "código genético" pero diferentes métodos para adaptarse a las características del proyecto.

Alistar Cockburn [Coc05] creó la *familia Cristal de métodos ágiles*[15] a fin de obtener un enfoque de desarrollo de software que premia la "maniobrabilidad" durante lo que Cockburn caracteriza como "un juego cooperativo con recursos limitados, de invención y comunicación, con el objetivo primario de entregar software útil que funcione y con la meta secundaria de plantear el siguiente juego" [Coc02].

Para lograr la maniobrabilidad, Cockburn y Highsmith definieron un conjunto de metodologías, cada una con elementos fundamentales comunes a todos, y roles, patrones de proceso, producto del trabajo y prácticas que son únicas para cada uno. La familia Cristal en realidad es un conjunto de ejemplos de procesos ágiles que han demostrado ser efectivos para diferentes tipos de proyectos. El objetivo es permitir que equipos ágiles seleccionen al miembro de la familia Cristal más apropiado para su proyecto y ambiente.

3.5.5 Desarrollo impulsado por las características (DIC)

WebRef
En la dirección **www.featuredrivendevelopment.com/** se encuentra una amplia variedad de artículos y presentaciones sobre el DIC.

El *desarrollo impulsado por las características* (DIC) lo concibió originalmente Peter Coad y sus colegas [Coa99] como modelo práctico de proceso para la ingeniería de software orientada a objetos. Stephen Palmer y John Felsing [Pal02] ampliaron y mejoraron el trabajo de Coad con la descripción de un proceso adaptativo y ágil aplicable a proyectos de software de tamaño moderado y grande.

Igual que otros proyectos ágiles, DIC adopta una filosofía que: 1) pone el énfasis en la colaboración entre los integrantes de un equipo DIC; 2) administra la complejidad de los problemas y del proyecto con el uso de la descomposición basada en las características, seguida de la integración de incrementos de software, y 3) comunica los detalles técnicos en forma verbal, gráfica y con medios basados en texto. El DIC pone el énfasis en las actividades de aseguramiento de la calidad del software mediante el estímulo de la estrategia de desarrollo incremental, el uso de inspecciones del diseño y del código, la aplicación de auditorías de aseguramiento de la calidad del software (véase el capítulo 16), el conjunto de mediciones y el uso de patrones (para el análisis, diseño y construcción).

En el contexto del DIC, una *característica* "es una función valiosa para el cliente que puede implementarse en dos semanas o menos" [Coa99]. El énfasis en la definición de características proporciona los beneficios siguientes:

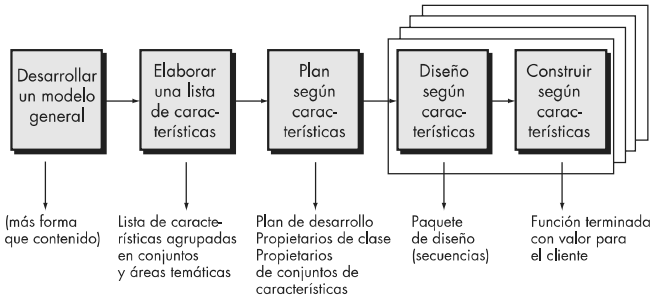
- Debido a que las características son bloques pequeños de funcionalidad que se entrega, los usuarios las describen con más facilidad, entienden cómo se relacionan entre sí y las revisan mejor en busca de ambigüedades, errores u omisiones.
- Las características se organizan por jerarquía de grupos relacionados con el negocio.
- Como una característica es el incremento de software DIC que se entrega, el equipo desarrolla características operativas cada dos semanas.
- El diseño y representación en código de las características son más fáciles de inspeccionar con eficacia porque éstas son pequeñas.
- La planeación, programación de actividades y seguimiento son determinadas por la jerarquía de características, y no por un conjunto de tareas de ingeniería de software adoptadas en forma arbitraria.

Coad y sus colegas [Coa99] sugieren el esquema siguiente para definir una característica:

<acción> el **<resultado>** **<a|por|de|para>** un **<objeto>**

15 El nombre "cristal" se deriva de los cristales de minerales, cada uno de los cuales tiene propiedades específicas de color, forma y dureza.

FIGURA 3.5
Desarrollo impulsado por las características [Coa99] (con permiso)



donde **<objeto>** es "una persona, lugar o cosa (incluso roles, momentos del tiempo o intervalos temporales, o descripciones parecidas a las entradas de un catálogo)". Algunos ejemplos de características para una aplicación de comercio electrónico son los siguientes:

Agregar el producto al carrito de compras

Mostrar las especificaciones técnicas del producto

Guardar la información de envío para el cliente

Un conjunto de características agrupa las que son similares en categorías relacionadas con el negocio y se define así:

<acción><ndo> un **<objeto>**

Por ejemplo: *Haciendo una venta del producto* es un conjunto de características que agruparía las que ya se mencionaron y otras más.

El enfoque DIC define cinco actividades estructurales "colaborativas" [Coa99] (en el enfoque DIC se llaman "procesos"), como se muestra en la figura 3.5.

El DIC pone más énfasis que otros métodos ágiles en los lineamientos y técnicas para la administración de proyectos. A medida que éstos aumentan su tamaño y complejidad, no es raro que la administración de proyectos *ad hoc* sea inadecuada. Para los desarrolladores, sus gerentes y otros participantes, es esencial entender el estado del proyecto, es decir, los avances realizados y los problemas que han surgido. Si la presión por cumplir el plazo de entrega es mucha, tiene importancia crítica determinar si la entrega de los incrementos del software está programada en forma adecuada. Para lograr esto, el DIC define seis puntos de referencia durante el diseño e implementación de una característica: "recorrido por el diseño, diseño, inspección del diseño, código, inspección del código, decisión de construir" [Coa99].

3.5.6 Desarrollo esbelto de software (DES)

El *desarrollo esbelto de software* (DES) adapta los principios de la manufactura esbelta al mundo de la ingeniería de software. Los principios de esbeltez que inspiran al proceso DES se resumen como sigue ([Pop03], [Pop06a]): *eliminar el desperdicio, generar calidad, crear conocimiento, aplazar el compromiso, entregar rápido, respetar a las personas* y *optimizar al todo*.

Es posible adaptar cada uno de estos principios al proceso del software. Por ejemplo, *eliminar el desperdicio* en el contexto de un proyecto de software ágil significa [Das05]: 1) no agregar características o funciones extrañas, 2) evaluar el costo y el efecto que tendrá en la programación de actividades cualquier nuevo requerimiento solicitado, 3) eliminar cualesquiera etapas superfluas del proceso, 4) establecer mecanismos para mejorar la forma en la que los miembros

del equipo obtienen información, 5) asegurar que las pruebas detecten tantos errores como sea posible, 6) reducir el tiempo requerido para pedir y obtener una decisión que afecta al software o al proceso que se aplica para crearlo, y 7) simplificar la manera en la que se transmite la información a todos los participantes involucrados en el proceso.

Para un análisis detallado del DES y para conocer lineamientos prácticos a fin de implementar el proceso, debe consultarse [Pop06a] y [Pop06b].

3.5.7 Modelado ágil (MA)

WebRef
En la dirección **www.agilemodeling.com** hay información amplia sobre el modelado ágil.

Hay muchas situaciones en las que los ingenieros de software deben construir sistemas grandes de importancia crítica para el negocio. El alcance y complejidad de tales sistemas debe modelarse de modo que: 1) todos los actores entiendan mejor cuáles son las necesidades que deben satisfacerse, 2) el problema pueda dividirse con eficacia entre las personas que deben resolverlo, y 3) se asegure la calidad a medida que se hace la ingeniería y se construye el sistema.

En los últimos 30 años se ha propuesto una gran variedad de métodos de modelado y notación para la ingeniería de software con objeto de hacer el análisis y el diseño (tanto en la arquitectura como en los componentes). Estos métodos tienen su mérito, pero se ha demostrado que son difíciles de aplicar y sostener (en muchos proyectos). Parte del problema es el "peso" de dichos métodos de modelación. Con esto se hace referencia al volumen de la notación que se requiere, al grado de formalismo sugerido, al tamaño absoluto de los modelos para proyectos grandes y a la dificultad de mantener el(los) modelo(s) conforme suceden los cambios. Sin embargo, el análisis y el modelado del diseño tienen muchos beneficios para los proyectos grandes, aunque no fuera más que porque hacen a esos proyectos intelectualmente más manejables. ¿Hay algún enfoque ágil para el modelado de la ingeniería de software que brinde una alternativa?

Cita:
"El otro día fui a la farmacia por una medicina para el resfriado… no fue fácil. Había toda una pared cubierta de productos. Al recorrerla vi uno que era de acción rápida, pero otro que era de larga duración… ¿Qué es más importante, el presente o el futuro?"
Jerry Seinfeld

En el "sitio oficial de modelado ágil", Scott Ambler [Amb02a] describe el *modelado ágil* (MA) del modo siguiente:

> El modelado ágil (MA) es una metodología basada en la práctica para modelar y documentar con eficacia los sistemas basados en software. En pocas palabras, es un conjunto de valores, principios y prácticas para hacer modelos de software aplicables de manera eficaz y ligera a un proyecto de desarrollo de software. Los modelos ágiles son más eficaces que los tradicionales porque son sólo buenos, sin pretender ser perfectos.

El modelado ágil adopta todos los valores del manifiesto ágil. La filosofía de modelado ágil afirma que un equipo ágil debe tener la valentía para tomar decisiones que impliquen rechazar un diseño y reconstruirlo. El equipo también debe tener la humildad de reconocer que los tecnólogos no tienen todas las respuestas y que los expertos en el negocio y otros participantes deben ser respetados e incluidos.

Aunque el MA sugiere una amplia variedad de principios de modelado "fundamentales" y "suplementarios", aquellos que son exclusivos del MA son los siguientes [Amb02a]:

Modelo con un propósito. Un desarrollador que use el MA debe tener en mente una meta específica (por ejemplo, comunicar información al cliente o ayudarlo a entender mejor algún aspecto del software) antes de crear el modelo. Una vez identificada la meta para el modelo, el tipo y nivel de detalle de la notación por usar serán más obvios.

Uso de modelos múltiples. Hay muchos modelos y notaciones diferentes que pueden usarse para describir el software. Para la mayoría de proyectos sólo es esencial un pequeño subconjunto. El MA sugiere que para dar la perspectiva necesaria, cada modelo debe presentar un diferente aspecto del sistema y que sólo deben utilizarse aquellos modelos que den valor al público al que se dirigen.