### **El ciclo de vida del desarrollo de sistemas**

El SDLC (Software Development Life Cycle) es una metodología en fases para el análisis y diseño, de acuerdo con la cual los sistemas se desarrollan mejor al utilizar un ciclo específico de actividades del analista y los usuarios.

#### ***Etapas del ciclo de vida del software según Kendall***

1. Identificación de los problemas, oportunidades y objetivos: En esta primera fase del ciclo de vida del desarrollo de sistemas, el analista se encarga de identificar correctamente los problemas, las oportunidades y los objetivos. En la primera fase el analista debe analizar con honestidad lo que está ocurriendo en la empresa. Después, junto con otros miembros de la organización, debe comenzar a señalar los problemas.  
   La identificación de los objetivos también es un componente importante de la primera fase. El analista debe descubrir primero qué trata de hacer la empresa; después debe ser capaz de determinar si alguno de los aspectos de las aplicaciones de los sistemas de información puede ayudar a que la empresa logre sus objetivos al enfrentar problemas u oportunidades específicos.  
   Las personas involucradas en la primera fase son los usuarios, los analistas y los administradores de sistemas que coordinan el proyecto. En esta fase las actividades consisten en entrevistar a los encargados de la administración de los usuarios, sintetizar el conocimiento obtenido, estimar el alcance del proyecto y documentar los resultados. El resultado de esta fase es un informe de viabilidad, el cual contiene la definición de un problema y sintetiza los objetivos. Después, la administración de la empresa debe tomar una decisión en cuanto a proceder o no con el proyecto propuesto. Si el grupo de usuarios no tiene suficientes fondos en su presupuesto o desea hacer frente a problemas que no están relacionados, o si los problemas no requieren un sistema computacional, tal vez se pueda recomendar una solución distinta y el proyecto de sistemas no continúe.
2. Determinación de los requerimientos de información del factor humano: La siguiente fase a la que entra el analista es determinar las necesidades de los usuarios involucrados, mediante el uso de varias herramientas, para comprender la forma en que interactúan en el contexto laboral con sus sistemas de información actuales. El analista utilizará métodos interactivos como entrevistas, muestreos e investigación de datos duros, además de los cuestionarios y los métodos discretos, como observar el comportamiento de los encargados al tomar las decisiones y sus entornos de oficina, y los métodos integrales como la creación de prototipos. En la fase de requerimientos del SDLC, el analista se esfuerza por comprender qué información requieren los usuarios para realizar sus trabajos. En este punto el analista examina cómo hacer que el sistema sea útil para las personas involucradas. Las personas involucradas en esta fase son los analistas y los usuarios, por lo general los gerentes y los trabajadores de operaciones. El analista de sistema debe conocer los detalles sobre las funciones del sistema actual: el quién (las personas involucradas), el qué (la actividad de la empresa), el dónde (el entorno en el que se lleva a cabo el trabajo), el cuándo (la coordinación) y el cómo (de qué manera particular se realizan los procedimientos actuales) de la empresa a la que está estudiando. Después, el analista debe preguntar por qué la empresa utiliza el sistema actual. Puede haber buenas razones por las cuales la empresa trabaje con los métodos actuales, razón por la que se deben tener en cuenta al diseñar un nuevo sistema.
3. Análisis de las necesidades del sistema: Aquí también hay herramientas y técnicas especiales que ayudan al analista a realizar las determinaciones de los requerimientos. Las herramientas como los diagramas de flujo de datos (DFD) para graficar la entrada, los procesos y la salida de las funciones de la empresa, o los diagramas de actividad o de secuencia para mostrar la secuencia de los eventos, sirven para ilustrar a los sistemas de una manera estructurada y gráfica. En este punto del SDLC, el analista de sistemas prepara una propuesta de sistemas en la que sintetiza todo lo que ha averiguado sobre los usuarios, la capacidad de uso y la utilidad de los sistemas actuales; incluye un análisis de costo-beneficio de las alternativas y, si se requiere, hace recomendaciones. Si la administración acepta una de las recomendaciones, el análisis continúa por esa vía. Cada problema de sistemas es único, por lo que nunca hay sólo una solución correcta. La manera en que se formule una recomendación o solución depende de las cualidades individuales y la capacitación profesional de cada analista, y de su interacción con los usuarios en el contexto de su entorno laboral.
4. Diseño del sistema recomendado: En la fase de diseño del SDLC, el analista de sistemas utiliza la información recolectada antes para realizar el diseño lógico del sistema de información. El analista diseña los procedimientos para ayudar a que los usuarios introduzcan los datos con precisión, de manera que los datos que entren al sistema de información sean los correctos. Además, el analista debe ayudar a que los usuarios completen la entrada de datos efectiva al sistema de información mediante el uso de las técnicas del buen diseño de formularios y páginas Web o pantallas. La interfaz conecta al usuario con el sistema, por lo que es extremadamente importante. La interfaz del usuario se diseña con ayuda de los usuarios para asegurar que el sistema sea perceptible, legible y seguro, así como atractivo y divertido de usar. La fase de diseño también incluye el diseño de bases de datos que almacenarán gran parte de los datos nece- sarios para los encargados de tomar las decisiones en la organización. Los usuarios se benefician de una base de datos bien organizada que sea lógica para ellos y se corresponda con la forma en que ven su trabajo. Por último, el analista debe diseñar controles y procedimientos de respaldo para proteger el sistema y los datos, y para producir paquetes de especificación de programas para los programadores. Cada paquete debe contener los diseños de las entradas y las salidas, las especificaciones de los archivos y los detalles sobre el procesamiento; también puede incluir árboles o tablas de decisión, UML o diagramas de flujo de datos, junto con los nombres y las funciones de cualquier código previamente escrito dentro de la empresa o que utilice código u otras bibliotecas de clases.
5. Desarrollo y documentación del software: En la quinta fase del SDLC, el analista trabaja con los programadores para desarrollar el software original requerido. Durante ella, el analista desarrolla junto con los usuarios una documentación efectiva para el software, incluyendo manuales de procedimientos, ayuda en línea, sitios Web con preguntas frecuentes (FAQ) y archivos Léame (Readme) para incluir con el nuevo software. Como los usuarios están involucrados desde el principio, la fase de documentación debe lidiar con las preguntas que hicieron y resolvieron junto con el analista. La documentación indica a los usuarios cómo deben usar el software y qué deben hacer en caso de que ocurran problemas. Los programadores desempeñan un rol clave en esta fase, ya que diseñan, codifican y eliminan los errores sintácticos de los programas de computadora. Para asegurar la calidad, un programador puede llevar a cabo un recorrido por el diseño o por el código para explicar las porciones complejas del programa a un equipo formado por otros programadores.
6. Prueba y mantenimiento del sistema: Antes de utilizar el sistema de información, se debe probar. Es mucho menos costoso detectar los problemas antes de entregar el sistema a los usuarios. Una parte del procedimiento de prueba es llevado a cabo por los programadores solos; la otra la realizan junto con los analistas de sistemas. Primero se completa una serie de pruebas para señalar los problemas con datos de muestra y después se utilizan datos reales del sistema actual. A menudo, los planes de prueba se crean en las primeras etapas del SDLC y se refinan a medida que el proyecto progresa. El mantenimiento del sistema y la documentación de este mantenimiento empieza en esta fase y se lleva a cabo de manera rutinaria durante toda la vida del sistema de información. Gran parte del trabajo rutinario del programador consiste en el mantenimiento, por lo cual las empresas invierten una gran cantidad de dinero en este proceso. Ciertos procedimientos de mantenimiento, como las actualizaciones de los programas, se pueden llevar a cabo a través del sitio Web del distribuidor. Muchos de los procedimientos sistemáticos que emplea el analista durante el SDLC pueden ayudar a asegurar que el mantenimiento siempre se mantenga en el nivel mínimo necesario.
7. Implementación y evaluación del sistema: En esta última fase del desarrollo de sistemas, el analista ayuda a implementar el sistema de información. En esta fase hay que capacitar a los usuarios para operar el sistema. Los distribuidores se encargan de una parte de la capacitación, pero la supervisión de la capacitación es responsabilidad del analista de sistemas. Además, el analista necesita planear una conversión sin problemas del sistema antiguo al nuevo. Este proceso incluye convertir los archivos de los formatos anteriores a los nuevos, o crear una base de datos, instalar equipo y llevar el nuevo sistema a producción.

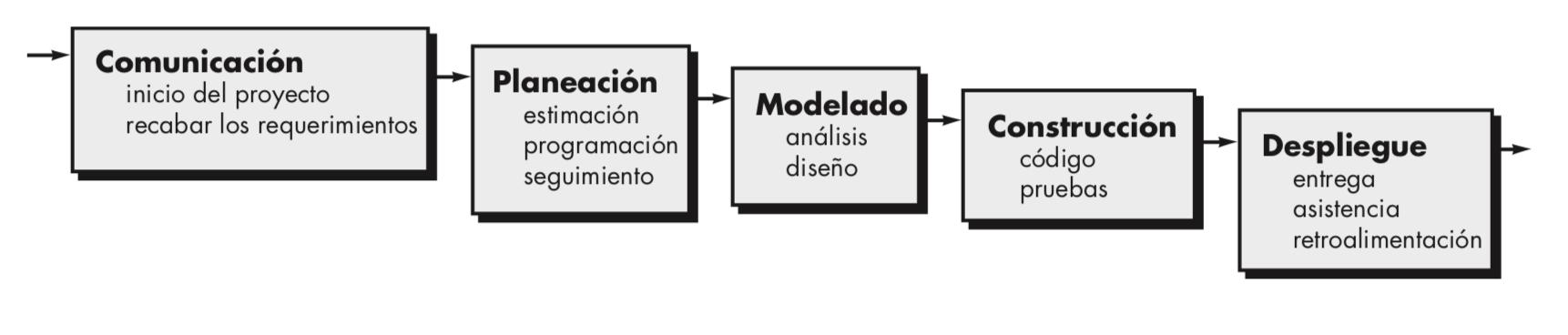
#### ***Etapas del ciclo de vida del software según Pressman***

1. Comunicación: Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes). Se busca entender los objetivos de los participantes respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software.
2. Planeación: Cualquier viaje complicado se simplifica si existe un mapa. Un proyecto de software es un viaje difícil, y la actividad de planeación crea un “mapa” que guía al equipo mientras viaja. El mapa —llamado plan del proyecto de software— define el trabajo de ingeniería de software al describir las tareas técnicas por realizar, los riesgos probables, los recursos que se requieren, los productos del trabajo que se obtendrán y una programación de las actividades.
3. Modelado: Ya sea usted diseñador de paisaje, constructor de puentes, ingeniero aeronáutico, carpintero o arquitecto, a diario trabaja con modelos. Crea un “bosquejo” del objeto por hacer a fin de entender el panorama general —cómo se verá arquitectónicamente, cómo ajustan entre sí las partes constituyentes y muchas características más—. Si se requiere, refina el bosquejo con más y más detalles en un esfuerzo por comprender mejor el problema y cómo resolverlo. Un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.
4. Construcción: Esta actividad combina la generación de código (ya sea manual o automatizada) y las pruebas que se requieren para descubrir errores en éste.
5. Despliegue: El software (como entidad completa o como un incremento parcialmente terminado) se entrega al consumidor que lo evalúa y que le da retroalimentación, misma que se basa en dicha evaluación.

### **Tipos de ciclo de vida**

#### **Modelo de la cascada**

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

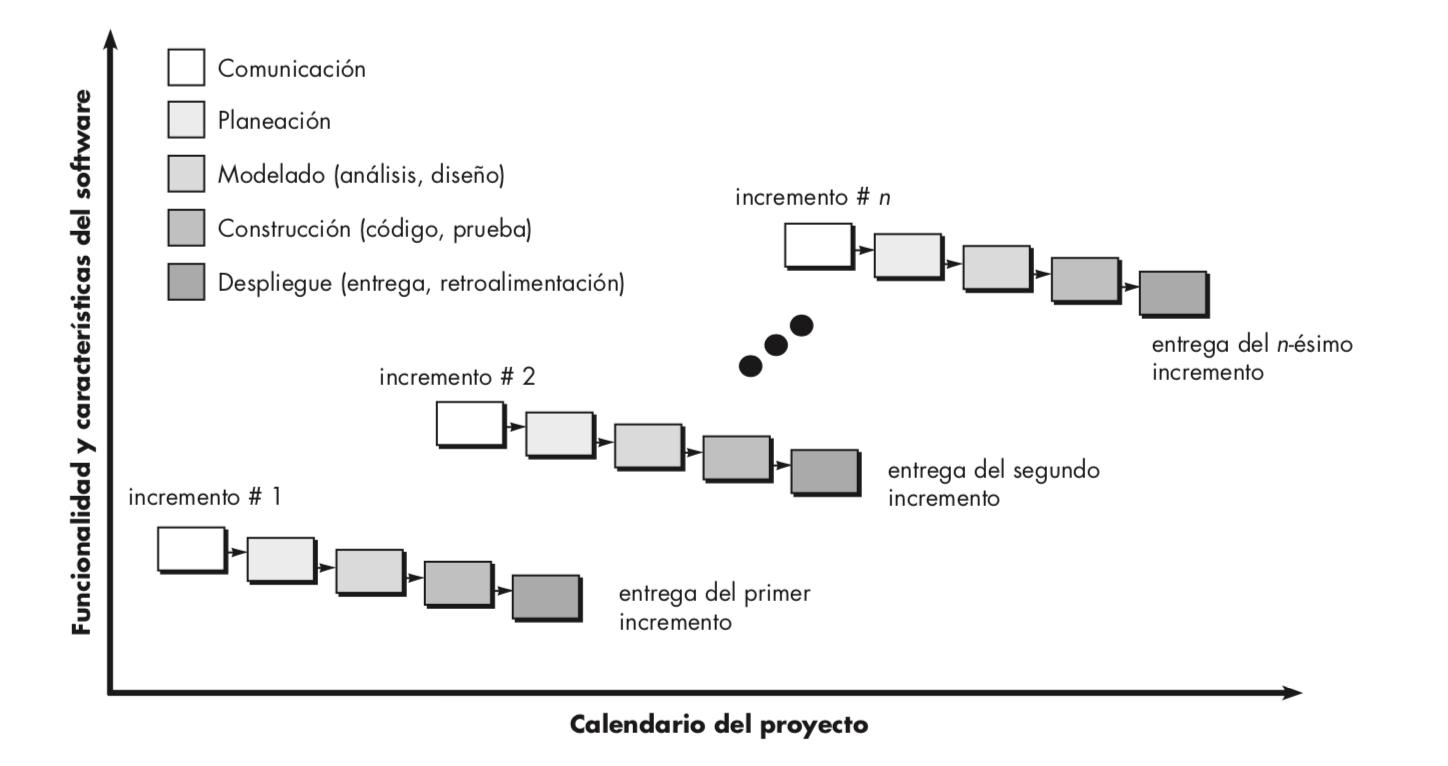


Entre los problemas que en ocasiones surgen al aplicar el modelo de la cascada se encuentran los siguientes:

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

#### **Modelos de proceso incremental**

Hay muchas situaciones en las que los requerimientos iniciales del software están razonablemente bien definidos, pero el alcance general del esfuerzo de desarrollo imposibilita un proceso lineal. Además, tal vez haya una necesidad imperiosa de dar rápidamente cierta funcionalidad limitada de software a los usuarios y aumentarla en las entregas posteriores de software. En tales casos, se elige un modelo de proceso diseñado para producir el software en incrementos. Cuando se utiliza un modelo incremental, es frecuente que el primer incremento sea el producto fundamental. Es decir, se abordan los requerimientos básicos, pero no se proporcionan muchas características suplementarias (algunas conocidas y otras no). El cliente usa el producto fundamental (o lo somete a una evaluación detallada). Como resultado del uso y/o evaluación, se desarrolla un plan para el incremento que sigue. El plan incluye la modificación del producto fundamental para cumplir mejor las necesidades del cliente, así como la entrega de características adicionales y más funcionalidad. Este proceso se repite después de entregar cada incremento, hasta terminar el producto final.  
El modelo de proceso incremental se centra en que en cada incremento se entrega un producto que ya opera. Los primeros incrementos son versiones desnudas del producto final, pero proporcionan capacidad que sirve al usuario y también le dan una plataforma de evaluación.  
El desarrollo incremental es útil en particular cuando no se dispone de personal para la implementación completa del proyecto en el plazo establecido por el negocio. Los primeros incrementos se desarrollan con pocos trabajadores. Si el producto básico es bien recibido, entonces se agrega más personal (si se requiere) para que labore en el siguiente incremento. Además, los incrementos se planean para administrar riesgos técnicos. Por ejemplo, un sistema grande tal vez requiera que se disponga de hardware nuevo que se encuentre en desarrollo y cuya fecha de entrega sea incierta. En este caso, tal vez sea posible planear los primeros incrementos de forma que eviten el uso de dicho hardware, y así proporcionar una funcionalidad parcial a los usuarios finales sin un retraso importante.



Ventajas:

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

Desventajas:

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

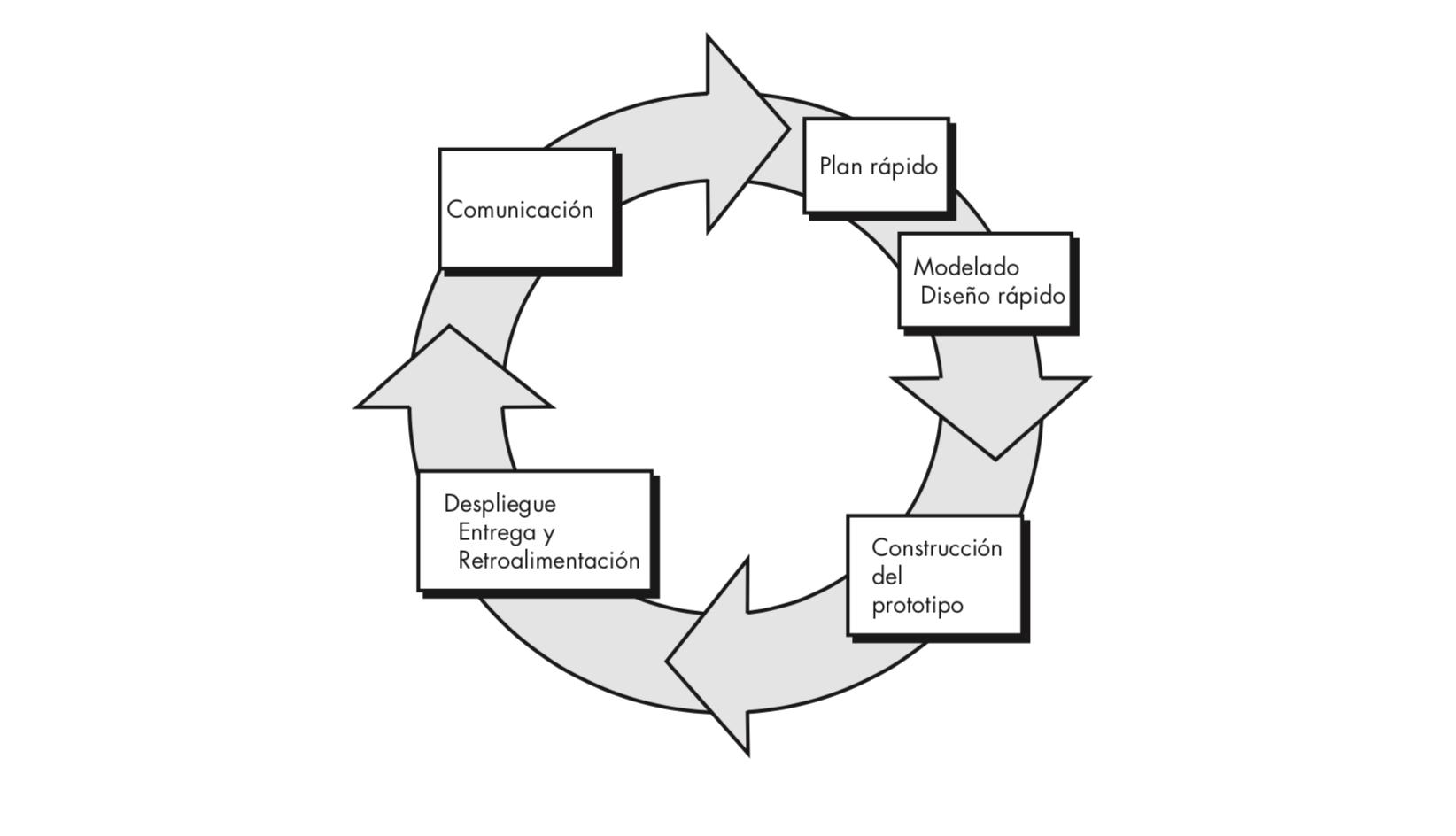
#### **Modelos de proceso evolutivo**

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

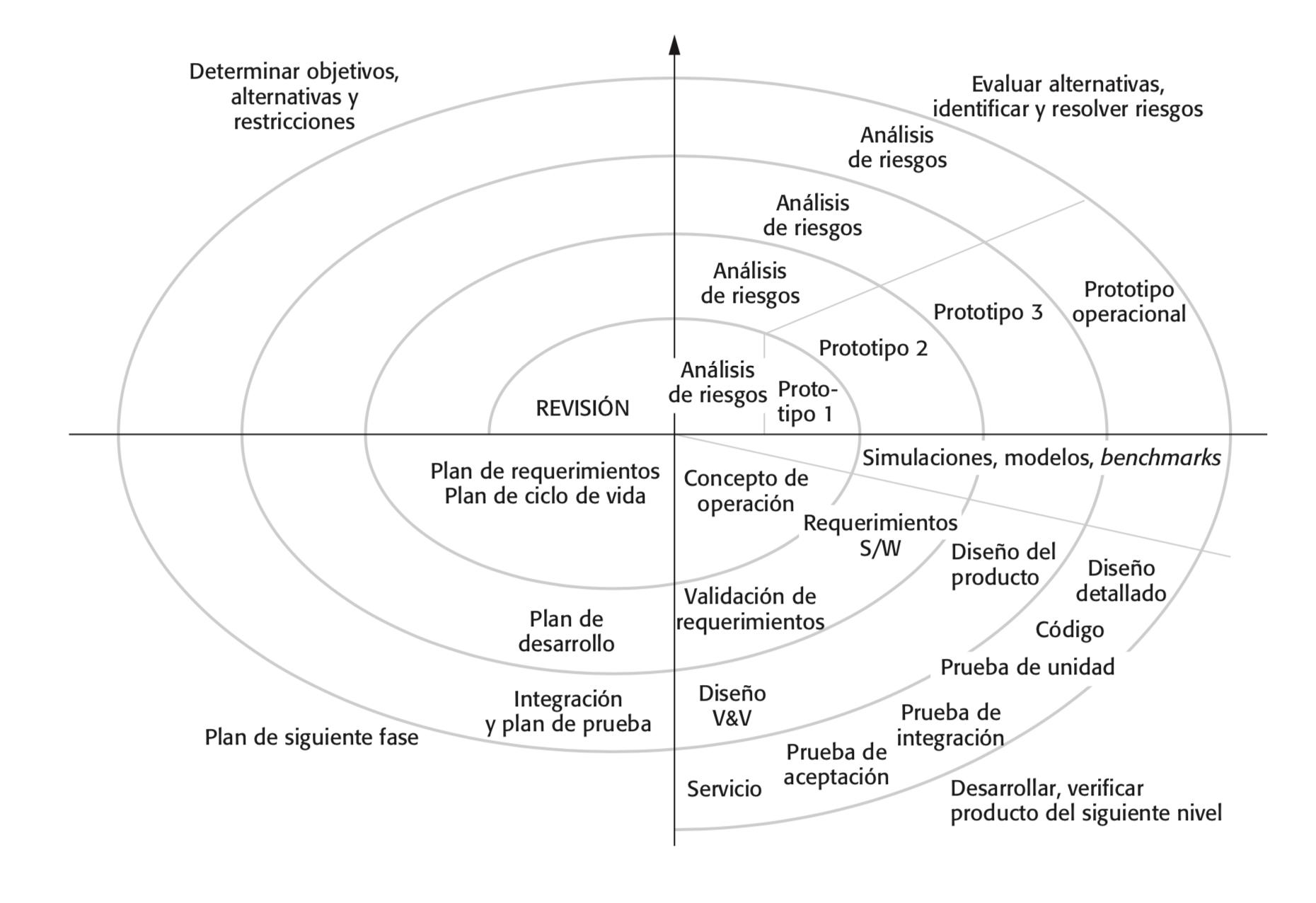
Prototipado: Es frecuente que un cliente defina un conjunto de objetivos generales para el software, pero que no identifique los requerimientos detallados para las funciones y características. En otros casos, el desarrollador tal vez no esté seguro de la eficiencia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debe adoptar la interacción entre el humano y la máquina. En estas situaciones, y muchas otras, el paradigma de hacer prototipos tal vez ofrezca el mejor enfoque.  
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 comienza con la comunicación. El analista se reúne con otros participantes para definir los objetivos generales del software, identifica cualesquiera requerimientos que conozca y detecta las áreas en las que es imprescindible una mayor definición. Se planea rápidamente una iteración para hacer el prototipo, y se lleva a cabo el modelado (en forma de un “diseño rápido”). Éste se centra en la representación de aquellos aspectos del software que serán visibles para los usuarios finales (por ejemplo, disposición de la interfaz humana o formatos de la pantalla de salida). El diseño rápido lleva a la construcción de un prototipo. Éste se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. La iteración ocurre a medida de que el prototipo es afinado para satisfacer las necesidades de distintos participantes, y al mismo tiempo le permite a usted entender mejor lo que se necesita hacer.  
El ideal es que el prototipo sirva como mecanismo para identificar los requerimientos del software. Si va a construirse un prototipo, pueden utilizarse fragmentos de programas existentes o aplicar herramientas (por ejemplo, generadores de reportes y administradores de ventanas) que permitan generar rápidamente programas que funcionen.  
Tanto a los participantes como a los ingenieros de software les gusta el paradigma de hacer prototipos. Los usuarios adquieren la sensación del sistema real, y los desarrolladores logran construir algo de inmediato. No obstante, hacer prototipos llega a ser problemático por las siguientes razones:

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

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



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



Cada ciclo en la espiral se divide en cuatro sectores:

1. Establecimiento de objetivos: Se definen objetivos específicos para dicha fase del proyecto. Se identifican restricciones en el proceso y el producto, y se traza un plan de gestión detallado. Se identifican los riesgos del proyecto. Pueden planearse estrategias alternativas, según sean los riesgos.
2. Valoración y reducción del riesgo: En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso. Se dan acciones para reducir el riesgo. Por ejemplo, si existe un riesgo de que los requerimientos sean inadecuados, puede desarrollarse un sistema prototipo.
3. Desarrollo y validación: Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema. Por ejemplo, la creación de prototipos desechables sería el mejor enfoque de desarrollo, si predominan los riesgos en la interfaz del usuario. Si la principal consideración son los riesgos de seguridad, el desarrollo con base en transformaciones formales sería el proceso más adecuado, entre otros. Si el principal riesgo identificado es la integración de subsistemas, el modelo en cascada sería el mejor modelo de desarrollo a utilizar.
4. Planeación: El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral. Si se opta por continuar, se trazan los planes para la siguiente fase del proyecto

La diferencia principal entre el modelo en espiral con otros modelos de proceso de software es su reconocimiento explícito del riesgo. Un ciclo de la espiral comienza por elaborar objetivos como rendimiento y funcionalidad. Luego, se numeran formas alternativas de alcanzar dichos objetivos y de lidiar con las restricciones en cada uno de ellos. Cada alternativa se valora contra cada objetivo y se identifican las fuentes de riesgo del proyecto. El siguiente paso es resolver dichos riesgos, mediante actividades de recopilación de información, como análisis más detallado, creación de prototipos y simulación. Una vez valorados los riesgos se realiza cierto desarrollo, seguido por una actividad de planeación para la siguiente fase del proceso. De manera informal, el riesgo significa simplemente algo que podría salir mal. Por ejemplo, si la intención es usar un nuevo lenguaje de programación, un riesgo sería que los compiladores disponibles no sean confiables o no produzcan un código-objeto suficientemente eficaz. Los riesgos conducen a propuestas de cambios de software y a problemas de proyecto como exceso en las fechas y el costo, de manera que la minimización del riesgo es una actividad muy importante de administración del proyecto.

Al comenzar el proceso evolutivo, el equipo de software realiza actividades implícitas en un circuito alrededor de la espiral en el sentido horario, partiendo del centro. El riesgo se considera conforme se desarrolla cada revolución. En cada paso evolutivo se marcan puntos de referencia puntuales: combinación de productos del trabajo y condiciones que se encuentran a lo largo de la trayectoria de la espiral.  
El primer circuito alrededor de la espiral da como resultado el desarrollo de una especificación del producto; las vueltas sucesivas se usan para desarrollar un prototipo y, luego, versiones cada vez más sofisticadas del software. Cada paso por la región de planeación da como resultado ajustes en el plan del proyecto. El costo y la programación de actividades se ajustan con base en la retroalimentación obtenida del cliente después de la entrega. Además, el gerente del proyecto ajusta el número planeado de iteraciones que se requieren para terminar el software.  
El modelo espiral es un enfoque realista para el desarrollo de sistemas y de software a gran escala. Como el software evoluciona a medida que el proceso avanza, el desarrollador y cliente comprenden y reaccionan mejor ante los riesgos en cada nivel de evolución. El modelo espiral usa los prototipos como mecanismo de reducción de riesgos, pero, más importante, permite aplicar el enfoque de hacer prototipos en cualquier etapa de la evolución del producto. Mantiene el enfoque de escalón sistemático sugerido por el ciclo de vida clásico, pero lo incorpora en una estructura iterativa que refleja al mundo real en una forma más realista. El modelo espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto y, si se aplica de manera apropiada, debe reducir los riesgos antes de que se vuelvan un problema.  
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.

En realidad el modelo en espiral no significa una visión radicalmente distinta de los modelos tradicionales, incremental o prototipado. Cualquiera de los tres modelos pueden verse con una representación espiral. Este modelo de Boehm es más bien una formalización o representación de los modelos de ciclo de vida más acertada que la representación en forma de cascada, pues permite observar mejor todos los elementos del proceso (incluido riesgos, objetivos, etc.). No obstante, no pasa de ser eso: una mejor representación de los modelos de ciclo de vida, no un ciclo de vida en sí mismo.

Una última palabra acerca de los procesos evolutivos  
Ya se dijo que el software de cómputo moderno se caracteriza por el cambio continuo, por tiempos de entrega muy apretados y por una necesidad apremiante de la satisfacción del cliente o usuario. En muchos casos, el tiempo para llegar al mercado es el requerimiento administrativo más importante. Si se pierde un nicho de mercado, todo el proyecto de software podría carecer de sentido.  
A pesar de los beneficios incuestionables de los procesos evolutivos de software, existen algunas preocupaciones.  
La primera es que hacer prototipos (y otros procesos evolutivos más sofisticados) plantea un problema para la planeación del proyecto debido a la incertidumbre en el número de ciclos que se requieren para elaborar el producto. La mayor parte de técnicas de administración y estimación de proyectos se basa en un planteamiento lineal de las actividades, por lo que no se ajustan por completo.  
En segundo lugar, los procesos evolutivos de software no establecen la velocidad máxima de la evolución. Si las evoluciones ocurren demasiado rápido, sin un periodo de relajamiento, es seguro que el proceso se volverá un caos. Por otro lado, si la velocidad es muy lenta, se verá perjudicada la productividad.  
En tercer lugar, los procesos de software deben centrarse en la flexibilidad y capacidad de extensión en lugar de en la alta calidad. Esto suena preocupante. Sin embargo, debe darse prioridad a la velocidad del desarrollo con el enfoque de cero defectos. Extender el desarrollo a fin de lograr alta calidad podría dar como resultado la entrega tardía del producto, cuando haya desaparecido el nicho de oportunidad. Este cambio de paradigma es impuesto por la competencia al borde del caos.

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

#### **Modelos concurrentes**

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

#### **Desarrollo basado en componentes**

El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo espiral. Es de naturaleza evolutiva y demanda un enfoque iterativo para la creación de software. Sin embargo, el modelo de desarrollo basado en componentes construye aplicaciones a partir de fragmentos de software prefabricados.  
Las actividades de modelado y construcción comienzan con la identificación de candidatos de componentes. Éstos pueden diseñarse como módulos de software convencional o clases orientadas a objetos o paquetes de clases. Sin importar la tecnología usada para crear los componentes, el modelo de desarrollo basado en componentes incorpora las etapas siguientes (se implementan con el uso de un enfoque evolutivo):

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

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

#### **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 verificar un sistema basado en computadora por medio del empleo de una notación matemática rigurosa.

Cuando durante el desarrollo se usan métodos formales, se obtiene un mecanismo 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.