Capítulo 1

Ciclo de vida del software

Los temas tratados en este libro se refieren a dos etapas del ciclo de vida del software. Corresponde, entonces, que le dediquemos el primer capítulo. Veremos las etapas que componen este ciclo, la necesidad de adoptarlo y su definición. Las metodologías que podemos adoptar. Además, haremos un repaso a los modelos de ciclo de vida y a algunas de sus ventajas como así también a los tipos de proyectos más beneficiosos de utilizar.

SERVICIO DE ATENCIÓN AL LECTOR: lectores@mpediciones.com

NECESIDAD DE UNA METODOLOGÍA

Cuando surgió la necesidad de adaptar los sistemas informáticos a las exigencias del mercado, el programador realizaba un relevamiento de las solicitudes de quien necesitaba cierto programa o producto software, y con aquellos requerimientos bajo el brazo comenzaba la dura tarea de codificar. Esta tarea no estaba administrada, supervisada o gestionada de ningún modo, por lo que se iba corrigiendo a medida que surgían los errores, tantos los lógicos provenientes de la codificación, como los de requerimientos solicitados por el cliente o usuario final.

En la década de 1970 los programas fueron creciendo en complejidad, por lo que la antigua técnica de **code & fix** (codificar y corregir) terminó quedando obsoleta. Esta técnica se basaba en requerimientos ambiguos y sin especificaciones puntuales. Al no seguir normas para el proyecto, el cliente o usuario sólo impartían especificaciones muy generales del producto final. Se programaba, se corregía, y se volvía a programar sobre la misma marcha del proyecto. El ciclo de vida de este tipo de proyectos finalizaba cuando se satisfacían las especificaciones, no sólo las primeras por las cuales nació la necesidad del programa, sino también todas aquellas que fueron surgiendo sobre la marcha.

Esta técnica tiene las ventajas de no gastar recursos en análisis, planificación, gestión de recursos, documentación, etc., y bien sabemos que es muy cómoda y muchas veces recomendable cuando el proyecto es **muy pequeño** y es llevado adelante por uno o dos programadores. Por otro lado, cuando el sistema no es pequeño o es más complejo de lo creído (tengamos en cuenta que no hubo análisis) nos trae desventajas en lo que se refiere a costo de recursos, que siempre será mayor del previsto; aumentará el tiempo de desarrollo y la calidad del código será bastante dudosa.

DEFINICIÓN DE METODOLOGÍA

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con **altas posibilidades de éxito**. Esta sistematización nos indica cómo dividiremos un gran proyecto en módulos más pequeños llamados etapas, y las acciones que corresponden en cada una de ellas, nos ayuda a definir entradas y salidas para cada una de las etapas y, sobre todo, normaliza el modo en que administraremos el proyecto. Entonces, una metodología para el desarrollo de software son los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado.

Desde un punto de vista general puede considerarse que el ciclo de vida de un software tiene tres etapas claramente diferenciadas, las cuales se detallan a continuacion:

• **Planificación:** idearemos un planeamiento detallado que guíe la gestión del proyecto, temporal y económicamente.
• **Implementación:** acordaremos el conjunto de actividades que componen la realización del producto.
• **Puesta en producción:** nuestro proyecto entra en la etapa de definición, allí donde se lo presentamos al cliente o usuario final, sabiendo que funciona correctamente y responde a los requerimientos solicitados en su momento. Esta etapa es muy importante no sólo por representar la aceptación o no del proyecto por parte del cliente o usuario final sino por las múltiples dificultades que suele presentar en la práctica, alargándose excesivamente y provocando costos no previstos.

A estas tres grandes etapas es conveniente añadir otras dos que, si bien pudieron enunciarse junto a las otras, es conveniente hacer una diferenciación ya que se tiende a menospreciarlas o a no darles la importancia que requieren.

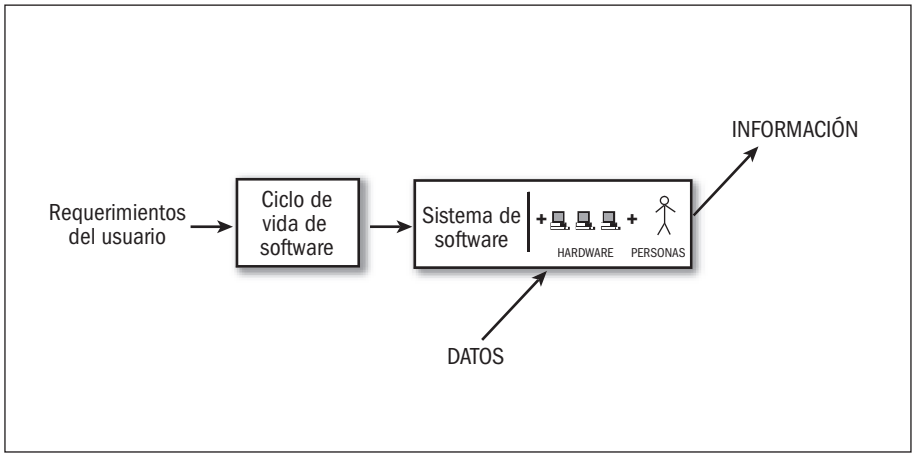

***Figura 1.** El ciclo de vida de un producto software se desarrolla fuera del ámbito productivo, aunque debemos conocer el entorno* (environment) *en el que será ejecutado.*

ISO12207

La **ISO**, International Organization for Standardization, en su norma **12207** define al **ciclo de vida de un software** como un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando desde la definición hasta la finalización de su uso.

- **Inicio:** éste es el nacimiento de la idea. Aquí definimos los objetivos del proyecto y los recursos necesarios para su ejecución. Hacia dónde queremos ir, y no cómo queremos ir. Las características implícitas o explícitas de cada proyecto hacen necesaria una etapa previa destinada a obtener el objetivo por el cual se escribirán miles o cientos de miles de líneas de código. Un alto porcentaje del éxito de nuestro proyecto se definirá en estas etapas que, al igual que la etapa de debugging, muchos líderes de proyecto subestiman.
- **Control en producción:** control del producto, analizando cómo el proceso difiere o no de los requerimientos originales e iniciando las acciones correctivas si fuesen necesarias. Cuando decimos que hay que corregir el producto, hacemos referencia a pequeñas desviaciones de los requerimientos originales que puedan llegar a surgir en el ambiente productivo. Si nuestro programa no realiza la tarea para lo cual fue creada, esta etapa no es la adecuada para el **rediseño**. Incluimos también en esta etapa el liderazgo, documentación y capacitación, proporcionando directivas a los recursos humanos, para que hagan su trabajo en forma correcta y efectiva.

OBJETIVOS DE CADA ETAPA

En cada una de las etapas de un modelo de ciclo de vida, se pueden establecer una serie de objetivos, tareas y actividades que lo caracterizan.
Haremos un repaso y una pequeña descripción de cada una de las etapas del ciclo de vida del software; una vez conocidas las etapas, tendremos que analizar cómo abordarlas en su conjunto. Existen distintos modelos de ciclo de vida, y la elección de un modelo para un determinado tipo de proyecto es realmente importante; el orden de las etapas es uno de estos puntos importantes, Si elegimos el modelo de cascada puro en el cual la validación se realiza al final del proyecto, y luego debemos retomar etapas previas, puede resultarnos no sólo incómodo, sino costoso.

- **Expresión de necesidades:** esta etapa tiene como objetivo el armado de un documento en el cual se reflejan los requerimientos y funcionalidades que ofrecerá al usuario el sistema a implementar (qué, y no cómo, se va a implementar).

{ } REINGENIERÍA

La reingeniería utiliza los resultados obtenidos por la ingeniería inversa para corregir o prevenir errores en un software. En los umbrales del año 2000, se reflejó la importancia de la reingeniería para evitar el **efecto Y2K**.

- **Especificaciones:** formalizamos los requerimientos; el documento obtenido en la etapa anterior se tomará como punto de partida para esta etapa.
- **Análisis:** determinamos los elementos que intervienen en el sistema a desarrollar, su estructura, relaciones, evolución temporal, funcionalidades, tendremos una descripción clara de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.
- **Diseño:** ya sabemos qué hacer, ahora tenemos que determinar cómo debemos hacerlo (¿cómo debe ser construido el sistema en cuestion?; definimos en detalle entidades y relaciones de las bases de datos, seleccionamos el lenguaje que vamos a utilizar, el Sistema Gestor de Bases de Datos, etc.).
- **Implementación:** empezamos a codificar algoritmos y estructuras de datos, definidos en las etapas anteriores, en el correspondiente lenguaje de programación o para un determinado sistema gestor de bases de datos. En muchos proyectos se pasa directamente a esta etapa; son proyectos muy arriesgados que adoptan un modelo de ciclo de vida de code & fix (codificar y corregir) donde se eliminan las etapas de especificaciones, análisis y diseño con la consiguiente pérdida de control sobre la gestión del proyecto.
- **Debugging:** el objetivo de esta etapa es garantizar que nuestro programa no contiene errores de diseño o codificación. En esta etapa no deseamos saber si nuestro programa realiza lo que solicitó el usuario, esa tarea le corresponde a la etapa de implementación. En ésta deseamos encontrar la mayor cantidad de errores. Todas los programas contienen errores: encontrarlos es cuestión de tiempo. Lo ideal es encontrar la mayoría, si no todos, en esta etapa. También se pueden agregar testeos de performance.
- **Validación:** esta etapa tiene como objetivo la verificación de que el sistema desarrollado cumple con los requerimientos expresados inicialmente por el cliente y que han dado lugar al presente proyecto. En muchos proyectos las etapas de validación y debugging se realizan en paralelo por la estrecha relación que llevan. Sin embargo, tenemos que evitar la confusión: podemos realizarlos en paralelo, pero no como una única etapa.
- **Evolución:** en la mayoría de los proyectos se considera esta etapa como **Mantenimiento y evolución**, y se le asigna, no sólo el agregado de nuevas funcionalida-

III PARADIGMAS DE PROGRAMACIÓN

Los paradigmas de programación son las estrategias para crear la estructura de un programa. Existen dos grupos: la **programación imperativa** y la **declarativa**. En la primera codificamos qué hacer y cómo, en la segunda sólo qué hacer, el lenguaje que utilizado hará el resto. Un ejemplo de la última es el lenguaje **SQL**, mediante el cual pedimos datos de una base pero no cómo hacerlo.

des (evolución); sino la corrección de errores que surgen (mantenimiento). En la práctica esta denominación no es del todo errónea, ya que es posible que aun luego de una etapa de debugging y validación exhaustiva, se filtren errores.

FINALIDAD DE UNA METODOLOGÍA

Lo que buscamos guiándonos con una metodología es prolijidad, corrección y control en cada etapa del desarrollo de un programa. Lo que nos permitirá una forma sistemática para poder obtener un producto correcto y libre de errores.

Clasificación de las metodologías

Existen dos metodologías que tienen analogía en la práctica con los **paradigmas de programación**. Metodología estructurada y metodología orientada a objetos.

- **Metodología estructurada:** la orientación de esta metodología se dirige hacia los procesos que intervienen en el sistema a desarrollar, es decir, cada función a realizar por el sistema se descompone en pequeños módulos individuales. Es más fácil resolver problemas pequeños, y luego unir cada una de las soluciones, que abordar un problema grande.
- **Metodología orientada a objetos:** a diferencia de la metodología mencionada anteriormente, ésta no comprende los procesos como funciones sino que arma módulos basados en componentes, es decir, cada componente es independiente del otro. Esto nos permite que el código sea reutilizable. Es más fácil de mantener porque los cambios están localizados en cada uno de estos componentes.

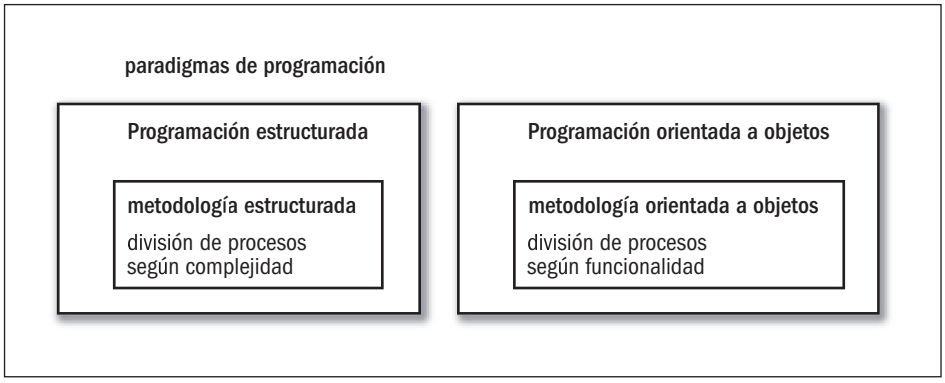

***Figura 2.** Las distintas metodologías nos ayudarán a abordar el proceso de desarrollo. Un consejo: la mejor metodología, es la que mejor conocemos.*

MODELOS DE CICLO DE VIDA

Las principales diferencias entre distintos modelos de ciclo de vida están divididas en tres grandes visiones:

- **El alcance del ciclo de vida**, que depende de hasta dónde deseamos llegar con el proyecto: sólo saber si es viable el desarrollo de un producto, el desarrollo completo o el desarrollo completo más las actualizaciones y el mantenimiento.
- **La cualidad y cantidad de las etapas** en que dividiremos el ciclo de vida: según el ciclo de vida que adoptemos, y el proyecto para el cual lo adoptemos.
- **La estructura y la sucesión de las etapas**, si hay realimentación entre ellas, y si tenemos libertad de repetirlas (iterar).

En los distintos modelos de ciclo de vida mencionaremos el **riesgo** que suponemos aceptar al elegirlo. Cuando hablamos de riesgo, nos referimos a la probabilidad que tendremos de volver a retomar una de las etapas anteriores, perdiendo tiempo, dinero y esfuerzo.

Ciclo de vida lineal

Es el más sencillo de todos los modelos. Consiste en descomponer la actividad global del proyecto en etapas separadas que son realizadas de manera lineal, es decir, cada etapa se realiza una sola vez, a continuación de la etapa anterior y antes de la etapa siguiente. Con un ciclo de vida lineal es muy fácil dividir las tareas, y prever los tiempos (sumando linealmente los de cada etapa).
Las actividades de cada una de las etapas mencionadas deben ser independientes entre sí, es decir, que es condición primordial que no haya retroalimentación entre ellas, aunque sí pueden admitirse ciertos supuestos de realimentación correctiva. Desde el punto de vista de la gestión, requiere también que se conozca desde el primer momento, con excesiva rigidez, lo que va a ocurrir en cada una de las distintas etapas antes de comenzarla. Esto ultimo minimiza, también, las posiblidades de errores durante la codificacion y reduce al mínimo la necesidad de requerir informacion del cliente o del usuario.

RIESGOS

Ninguno de los modelos de ciclo de vida evitan los riesgos que pueden aparecer en el desarrollo de un proyecto. Si evitaran los riesgos, entonces, eliminarían la **incertidumbre** que supone el cambio, agregado de requerimientos o errores cuando el proyecto se encuentra avanzado y ninguno lo hace. Intentan en mayor medida **prepararse para estos cambios o problemas.**

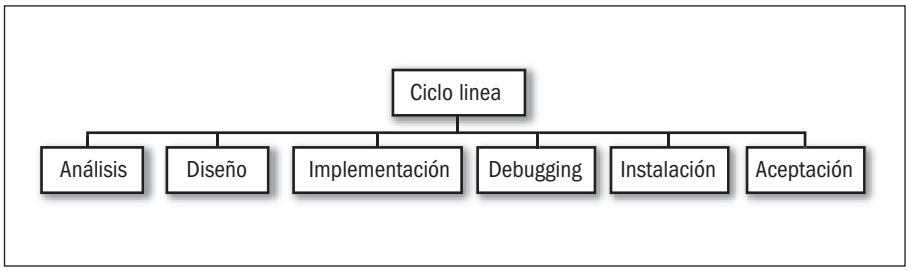

***Figura 3.** La sencillez del ciclo de vida lineal es la razón por la cual es el más elegido en el desarrollo de programas pequeños.*

Se destaca como ventaja la sencillez de su gestión y administración tanto económica como temporal, ya que se acomoda perfectamente a proyectos internos de una empresa para programas muy pequeños de **ABM** (sistemas que realizan Altas, Bajas y Modificaciones sobre un conjunto de datos). Tiene como desventaja que no es apto para Desarrollos que superen mínimamente requerimientos de retroalimentación entre etapas, es decir, es muy costoso retomar una etapa anterior al detectar alguna falla.

Es válido tomar este ciclo de vida cuando algún sector pequeño de una empresa necesita llevar un registro de datos acumulativos, sin necesidad de realizar procesos sobre ellos más que una consulta simple. Es decir, una aplicación que se dedique exclusivamente a almacenar datos, sea una base de datos o un archivo plano. Debido a que la realización de las etapas es muy simple y el código muy sencillo.

Ciclo de vida en cascada puro

Este modelo de ciclo de vida fue propuesto por Winston Royce en el año 1970. Es un ciclo de vida que admite iteraciones, contrariamente a la creencia de que es un ciclo de vida secuencial como el lineal. Después de cada etapa se realiza una o varias revisiones para comprobar si se puede pasar a la siguiente. Es un modelo rígido, poco flexible, y con muchas restricciones. Aunque fue uno de los primeros, y sirvió de base para el resto de los modelos de ciclo de vida.

INCERTIDUMBRE

Los modelos de ciclo de vida del software detallados en este libro desean tener cierta medida de la incertidumbre. Y muchos de estos ciclos de vida consideran que se puede medir, en un sentido de deseo, como la cantidad de información necesaria para evitar los riesgos posibles cuando se emprende la tarea del desarrollo de software.

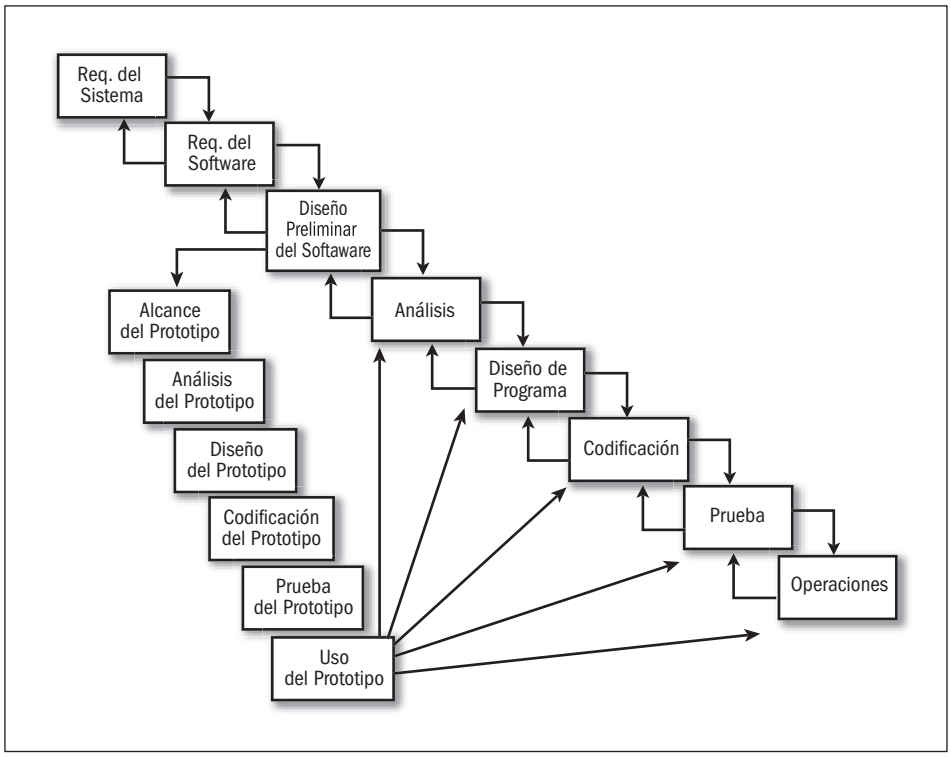

***Figura 4.** La necesidad de conocer los requerimientos al principio del proyecto es primordial al elegir este modelo de ciclo de vida a pesar de permitir iteraciones.*

Una de sus ventajas, además de su planificación sencilla, es la de proveer un producto con un elevado grado de calidad sin necesidad de un personal altamente calificado. Se pueden considerar como inconvenientes: la necesidad de contar con todos los requerimientos (o la mayoría) al comienzo del proyecto, y, si se han cometido errores y no se detectan en la etapa inmediata siguiente, es costoso y difícil volver atrás para realizar la correccion posterior.
Además, los resultados no los veremos hasta que no estemos en las etapas finales del ciclo, por lo que, cualquier error detectado nos trae retraso y aumenta el costo del desarrollo en funcion del tiempo que insume la correccion de éstos.

CICLO DE VIDA CON COMPONENTES

Muchas veces se necesita un programa en tiempo récord, o se desea administrar el proyecto despreocupándonos por una o varias etapas (principalmente la etapa de implementación). En los últimos años se empezó a considerar ciclo de vida con componentes al ensamblaje de software desarrollado por terceros en programas propios.

Es un ciclo adecuado para los proyectos en los que se dispone de todos los requerimientos al comienzo, para el desarrollo de un producto con funcionalidades conocidas o para proyectos, que aun siendo muy complejos, se entienden perfectamente desde el principio.

Se evidencia que es un modelo puramente teórico, ya que el usuario rara vez mantiene los requerimientos iniciales y existen muchas posibilidades de que debamos retomar alguna etapa anterior.
Pero es mejor que no seguir ningún ciclo de vida.

Fue utilizado en medianos y grandes proyectos hasta principios de la década de 1990, y a finales de esta década las críticas a este modelo aumentaron notablemente. Por lo que hoy en día sólo se lo cita como mero ejemplo bibliográfico. No podemos evitar decir que hay aspectos a cuestionar. Se le criticó, principalmente, el retardo en entregar partes del producto, su metodología para la corrección de errores, su obstinación por exigir requerimientos previos completos, y su alta rigidez.
A pesar de todo no es erróneo adaptarlo para alguna aplicación en la que el modelo de ciclo lineal no sea del todo adecuado, y el uso de un modelo de gestión más elaborado no lo justifique.

Ciclo de vida en V

Este ciclo fue diseñado por Alan Davis, y contiene las mismas etapas que el ciclo de vida en cascada puro. A diferencia de aquél, a éste se le agregaron dos subetapas de retroalimentación entre las etapas de análisis y mantenimiento, y entre las de diseño y debugging.

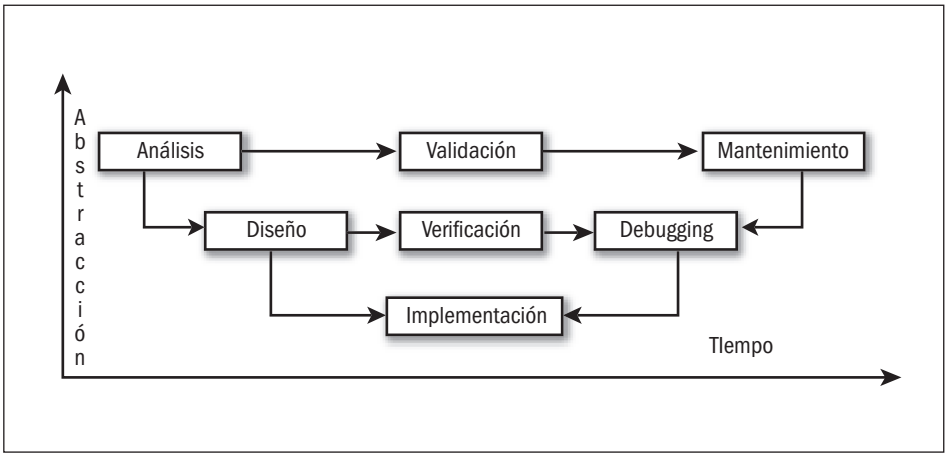

***Figura 5.** Este modelo nos ofrece mayor garantía de corrección al terminar el proyecto.*

Las ventajas y desventajas de este modelo son las mismas del ciclo anterior, con el agregado de los controles cruzados entre etapas para lograr una mayor corrección.

Podemos utilizar este modelo de ciclo de vida en aplicaciones, que si bien son simples (pequeñas transacciones sobre bases de datos por ejemplo), necesitan una confiabilidad muy alta. Un ejemplo claro en el que no nos podemos permitir el lujo de cometer errores es una aplicación de facturación, en la que si bien los procedimientos vistos individualmente son de codificación e interpretación sencilla, la aplicación en su conjunto puede tener matices complicados.

Ciclo de vida tipo Sashimi

Este ciclo de vida es parecido al ciclo de vida en cascada puro, con la diferencia de que en el ciclo de vida en cascada no se pueden solapar las etapas, y en éste sí. Esto suele, en muchos casos, aumentar su eficiencia ya que la retroalimentación entre etapas se encuentra implícitamente en el modelo

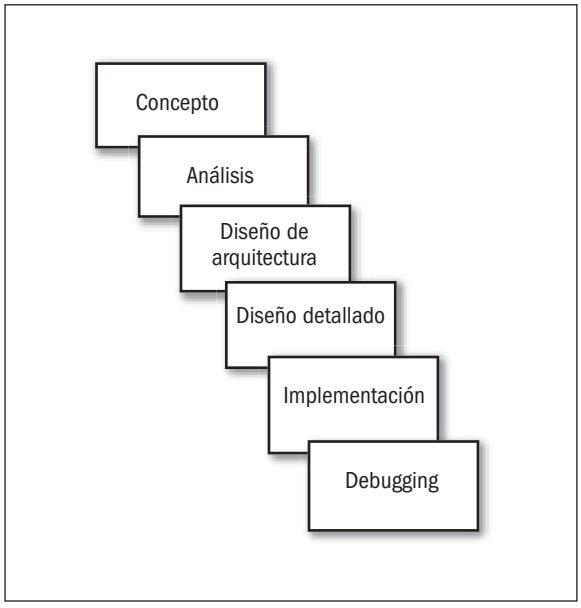

Figura 6. *El nombre procede del modelo del estilo japonés de presentar el pescado crudo cortado, en el que los cortes se solapan entre sí.*

Se hace notar como ventajas la ganancia de calidad en lo que respecta al producto final, la falta de necesidad de una documentación detallada (el ahorro proviene por el solapado de las etapas). Sus desventajas también se refieren al solapamiento de las etapas: es muy difícil gestionar el comienzo y fin de cada etapa y los problemas de comunicación, si aparecen, generan inconsistencias en el proyecto.

Cuando necesitemos realizar una aplicación que compartirá los recursos (CPU, memoria o espacio de almacenamiento) con otras aplicaciones en un ambiente productivo, este modelo de ciclo de vida es una opción muy válida. El solapamiento de sus etapas nos permite en la práctica jugar un poco con el **modelo de tres capas** ahorrando recursos.

Ciclo de vida en cascada con subproyectos

Sigue el modelo de ciclo de vida en cascada. Cada una de las cascadas se dividen en subetapas independientes que se pueden desarrollar en paralelo.

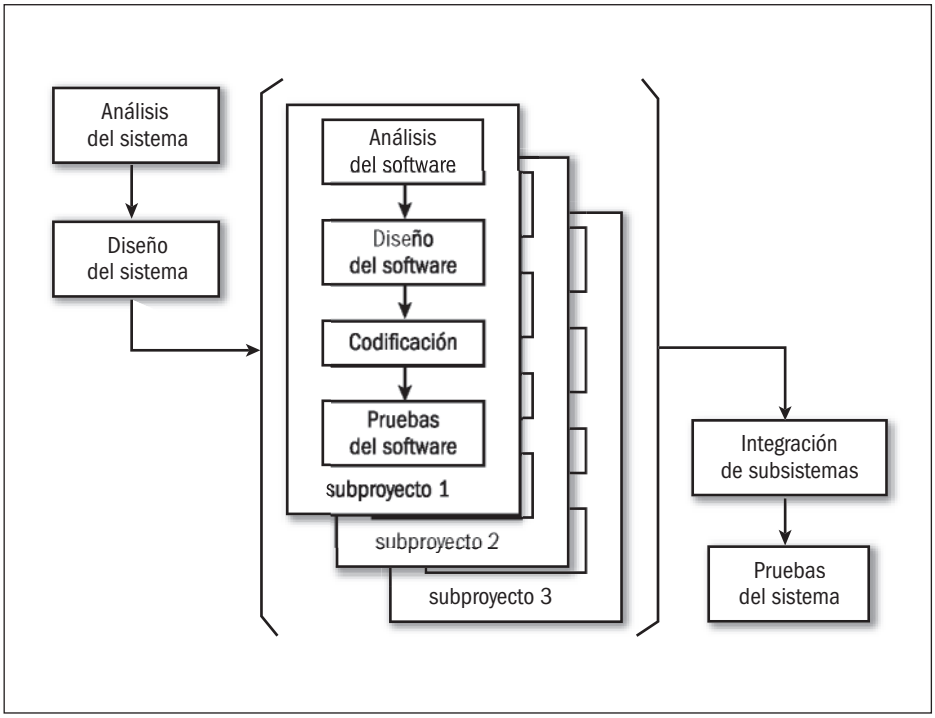

***Figura 7.** El modelo de ciclo de vida ideal cuando se cuenta con un plantel de programadores numeroso.*

III MODELO DE TRES CAPAS

Es un modelo de programación para aplicaciones de acceso a datos en que se busca separar la arquitectura del programa en tres capas. En la capa de datos solo nos preocupamos del almacenamiento de éstos, en la capa de negocios situamos todas las transacciones y validaciones y en la capa de presentación sólo encontraremos las rutinas de visualización e interacción con el usuario.

La ventaja es que se puede tener más gente trabajando al mismo tiempo, pero la desventaja es que pueden surgir dependencias entre las distintas subetapas que detengan el proyecto temporalmente si no es gestionado de manera correcta.
Podemos utilizar este modelo para administrar cualquier proyecto mencionado en los modelos anteriores. Pero cuidando de administrar muy bien los tiempos.

Ciclo de vida iterativo

También derivado del ciclo de vida en cascada puro, este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de solicitud de requerimientos.
Es la iteración de varios ciclos de vida en cascada. Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto. El cliente es quien luego de cada iteración, evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga al cliente.

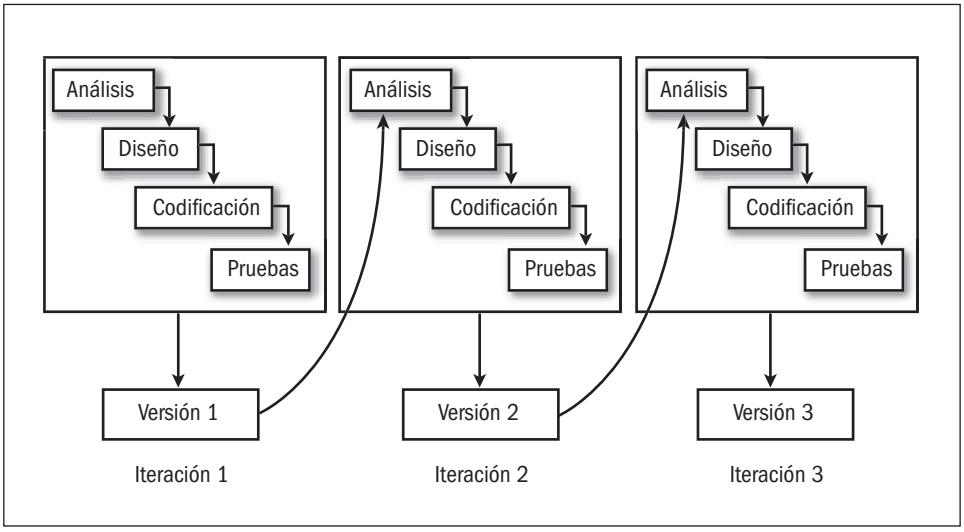

***Figura8.** Es un modelo ideal a seguir cuando el usuario necesita entregas rápidas aunque el proyecto no esté terminado.*

INGENIERÍA INVERSA

En casos en que ya se tiene un sistema y se necesita agregarle funcionalidades o modificarlo, pero no se dispone de la documentación. La ingeniería inversa se encarga de obtener y analizar el resultado de una etapa del ciclo de vida para obtener el resultado de la etapa anterior. lo que se busca es analizar el código para obtener el diseño con el cual fue implementado.

Se suele utilizar en proyectos en los que los requerimientos no están claros de parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente.

Podemos adoptar el modelo mencionado en aplicaciones medianas a grandes, en las que el usuario o cliente final no necesita todas las funcionalidades desde el principio del proyecto. Quizás una empresa que debe migrar sus aplicaciones hacia otra arquitectura, y desea hacerlo paulatinamente, es un candidato ideal para este tipo de modelo de ciclo de vida.

Ciclo de vida por prototipos

El uso de programas prototipo no es exclusivo del ciclo de vida iterativo. En la práctica los prototipos se utilizan para validar los requerimientos de los usuarios en cualquier ciclo de vida.

Si no se conoce exactamente cómo desarrollar un determinado producto o cuáles son las especificaciones de forma precisa, suele recurrirse a definir especificaciones iniciales para hacer un prototipo, o sea, un producto parcial y provisional. En este modelo, el objetivo es lograr un producto intermedio, antes de realizar el producto final, para conocer mediante el prototipo cómo responderán las funcionalidades previstas para el producto final.
Antes de adoptar este modelo de ciclo debemos evaluar si el esfuerzo por crear un prototipo vale realmente la pena adoptarlo.

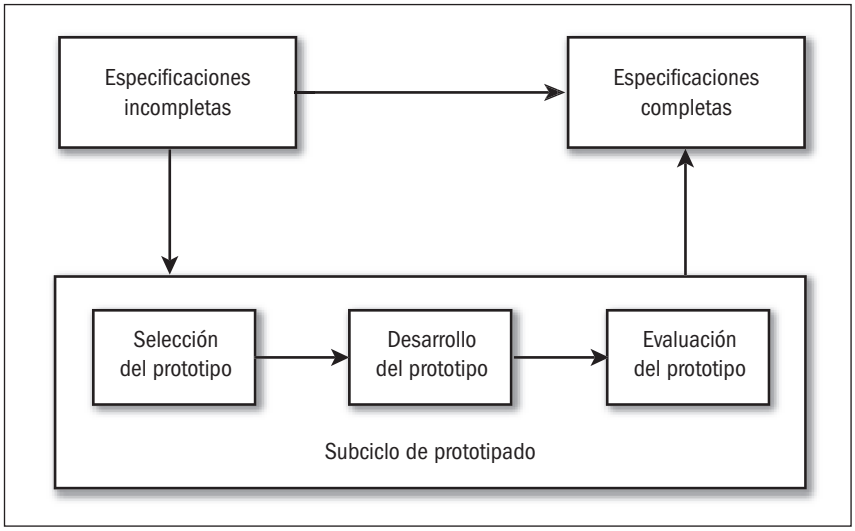

***Figura 9.** Este modelo nos permite suavizar la transición entre los requerimientos iniciales y finales que surgen en la creación de un proyecto con grandes innovaciones.*

Se utiliza mayoritariamente en desarrollos de productos con innovaciones importantes, o en el uso de tecnologías nuevas o poco probadas, en las que la incertidumbre sobre los resultados a obtener, o la ignorancia sobre el comportamiento, impiden iniciar un proyecto secuencial.
La ventaja de este ciclo se basa en que es el único apto para desarrollos en los que no se conoce a priori sus especificaciones o la tecnología a utilizar. Como contrapartida, por este desconocimiento, tiene la desventaja de ser altamente costoso y difícil para la administración temporal.

Si deseamos migrar aplicaciones de tecnología para adoptar sus nuevas funcionalidades o simplemente para estar en la **cresta de la ola**, este modelo es ideal. Un claro ejemplo son las llegadas de Java y la tecnología .NET que si bien contaban con respaldo y material de ayuda, implantaron nuevas tendencias.

Ciclo de vida evolutivo

Este modelo acepta que los requerimientos del usuario pueden cambiar en cualquier momento.
La práctica nos demuestra que obtener todos los requerimientos al comienzo del proyecto es extremadamente difícil, no sólo por la dificultad del usuario de transmitir su idea, sino porque estos requerimientos evolucionan durante el desarrollo y de esta manera, surgen nuevos requerimientos a cumplir. El modelo de ciclo de vida evolutivo afronta este problema mediante una iteración de ciclos **requerimientos–desarrollo–evaluación.**

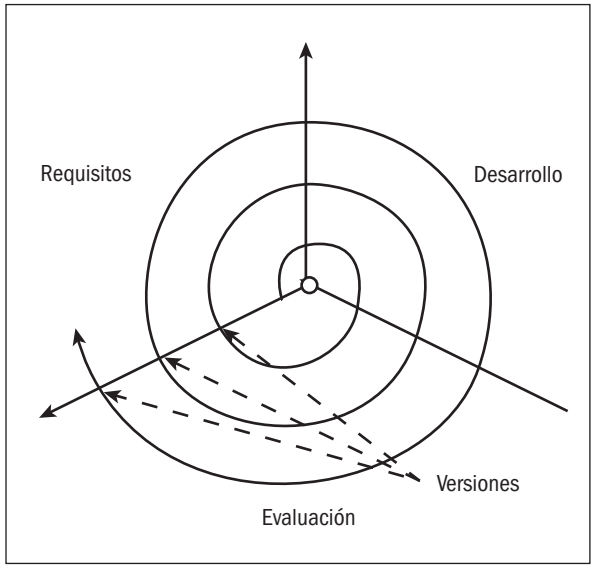

Figura 10. *Luego de cada desarrollo obtenemos una nueva versión del producto.*

Resulta ser un modelo muy útil cuando desconocemos la mayoría de los requerimientos iniciales, o estos requerimientos no están completos.
Tomemos como ejemplo un sistema centralizado de stock–ventas–facturación, en el cual hay muchas áreas que utilizarán la aplicación. Tenemos dos complicaciones: la primera, los usuarios no conocen de informática, la segunda, no es uno, sino varios los sectores que nos pueden pedir modificaciones o hacer nuevas solicitudes. Además, el pedido de un sector puede influir en los requerimientos del otro. Se hace necesario, entonces, lograr que la aplicación evolucione hasta lograr las satisfacciones de los todos los sectores involucrados.

Ciclo de vida incremental

Este modelo de ciclo de vida se basa en la filosofía de construir incrementando las funcionalidades del programa.
Se realiza construyendo por modulos que cumplen las diferentes funciones del sistema. Esto permite ir aumentando gradualmente las capacidades del software.
Este ciclo de vida facilita la tarea del desarrollo permitiendo a cada miembro del equipo desarrollar un modulo particular en el caso de que el proyecto sea realizado por un equipo de programadores.
Es una repetición del ciclo de vida en cascada, aplicándose este ciclo en cada funcionalidad del programa a construir. Al final de cada ciclo le entregamos una versión al cliente que contiene una nueva funcionalidad. Este ciclo de vida nos permite realizar una entrega al cliente antes de terminar el proyecto.

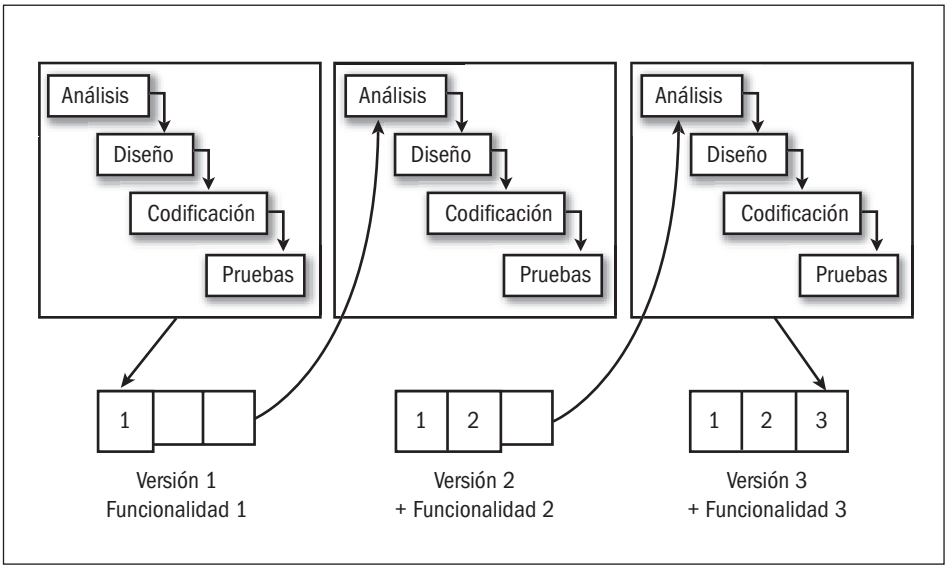

Figura 11. *Una forma de reducir los riesgos es ir construyendo partes del sistema adoptando este modelo.*

El modelo de ciclo de vida incremental nos genera algunos beneficios tales como los que se describen a continuacion:

- Construir un sistema pequeño siempre es menos riesgoso que construir un sistema grande.
- Como desarrollamos independientemente las funcionalidades, es más fácil relevar los requerimientos del usuario.
- Si se detecta un error grave, sólo desechamos la última iteración.
- No es necesario disponer de los requerimientos de todas las funcionalidades en el comienzo del proyecto y además facilita la labor del desarrollo con la conocida filosofía de **divide & conqueror**.

Este modelo de ciclo de vida no está pensado para cierto tipo de aplicaciones, sino que está orientado a cierto tipo de usuario o cliente. Podremos utilizar este modelo de ciclo de vida para casi cualquier proyecto, pero será verdaderamente útil cuando el usuario necesite entregas rápidas, aunque sean parciales.

Ciclo de vida en espiral

Este ciclo puede considerarse una variación del modelo con prototipado, fue diseñado por Boehm en el año 1988. El modelo se basa en una serie de ciclos repetitivos para ir ganando madurez en el producto final. Toma los beneficios de los ciclos de vida incremental y por prototipos, pero se tiene más en cuenta el concepto de riesgo que aparece debido a las incertidumbres e ignorancias de los requerimientos proporcionados al principio del proyecto o que surgirán durante el desarrollo. A medida que el ciclo se cumple (el avance del espiral), se van obteniendo prototipos sucesivos que van ganando la satisfacción del cliente o usuario. A menudo, la fuente de incertidumbres es el propio cliente o usuario, que en la mayoría de las oportunidades no sabe con perfección todas las funcionalidades que debe tener el producto.
En este modelo hay cuatro actividades que envuelven a las etapas.

DIFERENCIAS

El modelo de ciclo incremental **no** es parecido al modelo de ciclo de vida evolutivo. En el incremental partimos de que no hay incertidumbre en los requerimientos iniciales, en el evolutivo somos conscientes de que comenzamos con un alto grado de incertidumbre. En el incremental **suponemos** que conocemos el problema, y lo dividimos. El evolutivo gestiona la incertidumbre.

- **Planificación:** Relevamiento de requerimientos iniciales o luego de una iteración.
- **Análisis de riesgo:** De acuerdo con el relevamiento de requerimientos decidimos si continuamos con el desarrollo.
- **Implementación:** desarrollamos un prototipo basado en los requerimientos.
- **Evaluación:** El cliente evalúa el prototipo, si da su conformidad, termina el proyecto. En caso contrario, incluimos los nuevos requerimientos solicitados por el cliente en la siguiente iteración.

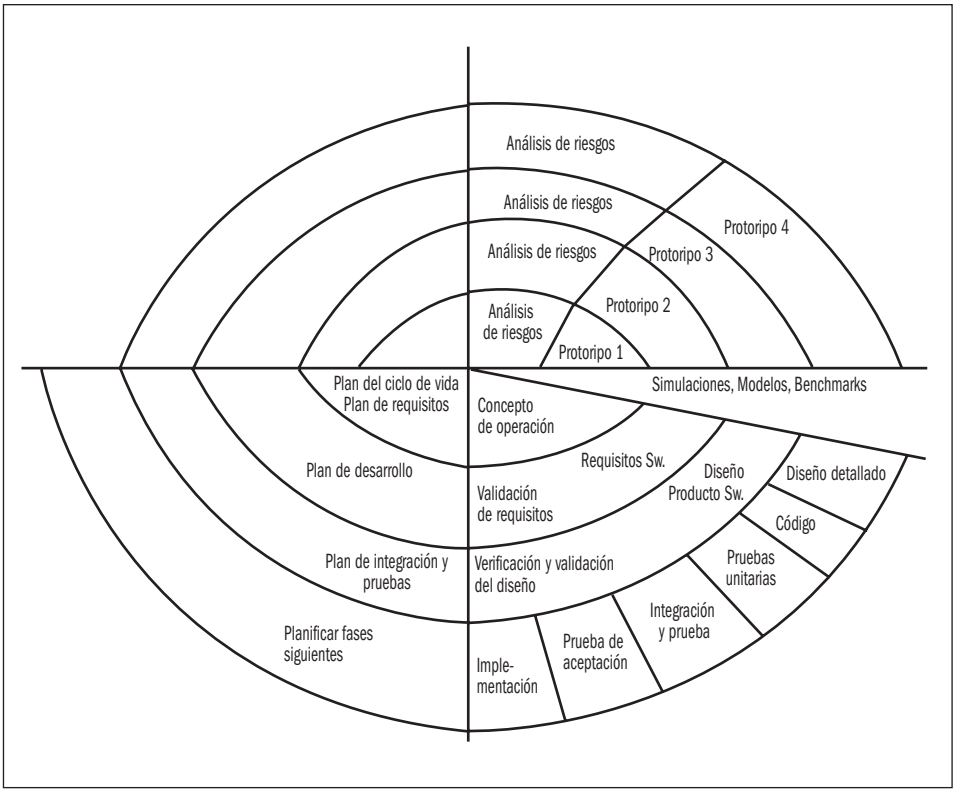

***Figura 12.** El espiral se repite las veces que sea necesario hasta que el cliente o usuario obtiene la satisfacción de sus necesidades, momento en el cual nos retiramos del espiral.*

III INGENIERÍA DEL SOFTWARE

En cada una de las etapas de los distintos ciclos de vida del software mencionados interviene la llamada ingeniería del software, que la definimos como el establecimiento de los principios basicos y métodos de la ingeniería, orientados a obtener software de calidad económico, que sea fiable y que funcione de manera eficiente sobre **máquinas reales**.

La ventaja más notoria de este modelo de desarrollo de software es que puede comenzarse el proyecto con un alto grado de incertidumbre, se entiende también como ventaja el bajo riesgo de retraso en caso de detección de errores, ya que se puede solucionar en la próxima rama del espiral.
Algunas de las las desventajas son: el costo temporal que suma cada vuelta del espiral, la dificultad para evaluar los riesgos y la necesidad de la presencia o la comunicacion continua con el cliente o usuario.
Se observa que es un modelo adecuado para grandes proyectos internos de una empresa, en donde no es posible contar con todos los requerimientos desde el comienzo y el usuario está en nuestro mismo ambiente laboral.

Podemos citar una aplicación que administre reclamos, pedidos e incidentes, como ejemplo para utilizar este modelo de ciclo de vida, en el que los sectores que utilizarán el sistema son demasiados y con intereses muy diversos como para lograr un relevamiento exhaustivo y completo de los requerimientos.

Ciclo de vida orientado a objetos

Esta técnica fue presentada en la década del 90, tal vez como una de las mejores metodologías a seguir para la creación de productos software.
Puede considerarse como un modelo pleno a seguir, como así también una alternativa dentro de los modelos anteriores.

Al igual que la filosofía del paradigma de la programación orientada a objetos, en esta metodología cada funcionalidad, o requerimiento solicitado por el usuario, es considerado un objeto. Los objetos están representados por un conjunto de propiedades, a los cuales denominamos **atributos,** por otra parte, al comportamiento que tendrán estos objetos los denominamos **métodos**.
Vemos que tanto la filosofía de esta metodología, los términos utilizados en ella y sus fines, coinciden con la idea de obtener un concepto de **objeto** sobre casos de la vida real.

III CASOS DE USO

Es una situación particular que se presenta en el funcionamiento del programa, que comienza cuando algún usuario o módulo del sistema solicita realizar alguna transacción o secuencia de eventos. En esta situación debemos identificar los objetos que participan, qué función cumple cada uno, y la colaboración que obtiene de otros objetos, o sea, los métodos invocados.

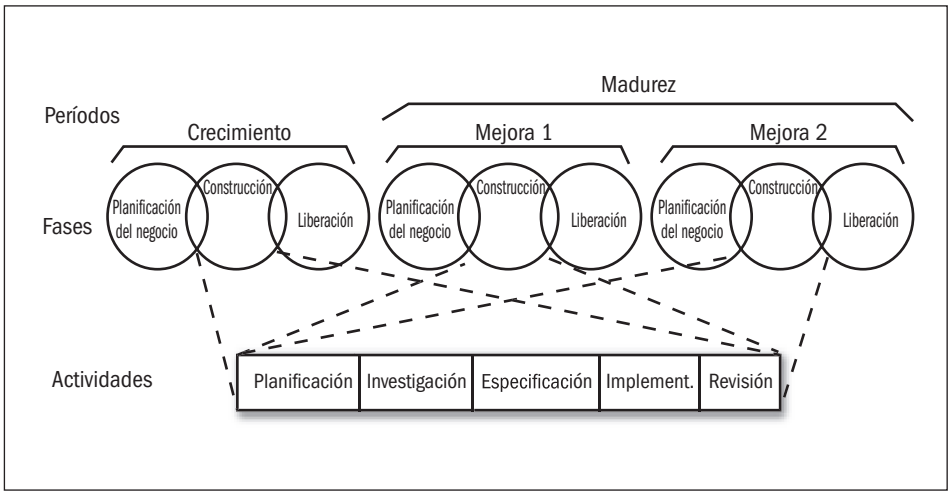

Figura 13. *Un modelo muy versátil, tanto para pequeños como para grandes proyectos.*

La característica principal de este modelo es la **abstracción** de los requerimientos de usuario, por lo que este modelo es mucho más flexible que los restantes, que son rígidos en requerimientos y definición, soportando mejor la incertidumbre que los anteriores, aunque sin garantizar la ausencia de riesgos. La abstracción es lo que nos permite analizar y desarrollar las características esenciales de un objeto (requerimiento), despreocupándonos de las menos relevantes.
Favorece la reducción de la complejidad del problema que deseamos abordar y permite el perfeccionamiento del producto.

En este modelo se utilizan las llamadas fichas **CRC (clase–responsabilidades–colaboración)** como herramienta para obtener las abstracciones y mecanismos clave de un sistema analizando los requerimientos del usuario. En la ficha CRC se escribe el nombre de la clase u objeto, sus responsabilidades (los métodos) y sus colaboradores (otras clases u objetos de los cuales necesita). Estas fichas, además, nos ayudan a confeccionar los denominados **casos de uso**.

No es correcto suponer que este modelo sólo es útil cuando se escoge para la implementación un lenguaje con orientación a objetos. Se puede utilizar independientemente del lenguaje elegido. Es un modelo a seguir, una técnica, y no nos obliga a utilizar ningún lenguaje en particular.

Como mencionamos, es un modelo muy versátil, y por ser uno de los últimos en aparecer, aprendió mucho de los anteriores. Las aplicaciones que podemos incluir como ejemplo para su uso van desde programas de monitoreo de procesos, grandes sistemas de transacciones sobre base de datos, hasta procesamiento por lotes.

Conclusión

Luego de ver algunos de los modelos de ciclo de vida más utilizados surge la pregunta con la respuesta más codiciada: ¿qué modelo de ciclo de vida elegir? Sabemos que ninguno predomina sobre los otros. Por ello, debemos elegir el modelo que mejor se adapte al proyecto que desarrollaremos. Podemos analizar, para guiarnos en nuestra elección, la **complejidad** del problema, el **tiempo** que disponemos para hacer la **entrega final**, o si el usuario o cliente desea **entregas parciales**, la **comunicación** que existe entre el equipo de desarrollo y el usuario y, por último, qué certeza (o incertidumbre) tenemos de que los **requerimientos** dados por el usuario son correctos y completos.

RESUMEN

Hemos visto en este capítulo por qué necesitamos una metodología para realizar el desarrollo de productos de software. Definimos la metodología que utilizamos. Vimos los rasgos generales del ciclo de vida del software. Cada una de sus etapas, las actividades y los modelos más conocidos que existen en el mercado.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Por qué necesitamos una metodología?

2 ¿Sirve el modelo de ciclo de vida Code & Fix?

3 ¿Existe algún modelo de ciclo de vida que predomine?

4 ¿Seguir un modelo de ciclo de vida, nos garantiza el éxito del desarrollo?

5 ¿Se puede medir la incertidumbre que tenemos sobre los requerimientos iniciales?

6 ¿La generación de programas prototipo, es exclusiva de un solo modelo de ciclo de vida?

7 ¿Podemos utilizar un lenguaje imperativo para el modelo de ciclo de vida orientado a objetos?

8 Enumere el ciclo de vida y los pasos que seguiría, si debiese desarrollar una aplicación que monitoree el estado de las redes de una empresa.

9 Realice una lista de requerimientos hipotéticos para una aplicación que deba ejecutar archivos de música, pida la misma lista a un usuario no programador y compare las listas. ¿Qué enfoques encuentra en cada lista?

10 A modo de encuesta, pregunte a sus colegas programadores, quién y por qué ha utilizado un ciclo de vida. Indague sobre los resultados obtenidos.