TEMA 2: LA INGENIERÍA DE SOFTWARE

Introducción

Los sistemas de software son abstractos e intangibles. No están restringidos por las propiedades de los materiales ni regidos por leyes físicas ni por procesos de fabricación. Debido a la falta de restricciones físicas los sistemas de software se vuelven rápidamente complejos, difíciles de entender y costosos de cambiar. Requieren de diferentes enfoques y no todas requieren de las mismas técnicas.

Aquí interesa el software de dimensión industrial que los software caseros o construidos por programadores aficionados. Los sistemas de software de esta clase tienden a tener un ciclo de vida largo y a lo largo del tiempo muchos usuarios llegan a depender de su funcionamiento correcto. La complejidad que el software tiene es una propiedad esencial de todos los sistemas de software de gran tamaño: se puede dominar pero no eliminar.

LA COMPLEJIDAD DEL SOFTWARE

La complejidad del software se deriva de cuatro problemas:

a. Complejidad del dominio del problema: Existe un desacoplamiento entre los usuarios de un sistema y sus desarrolladores: los usuarios suelen encontrar grandes dificultades al intentar expresar con precisión sus necesidades. Otra complicación es que los requisitos de un sistema de software cambian frecuentemente altera las reglas del problema original.

b. Dificultad de gestionar el desarrollo. Se debe utilizar de forma ideal un equipo lo más pequeño posible. Un mayor número de miembros implica una comunicación más compleja y por lo tanto una coordinación más difícil, particularmente si el equipo está disperso geográficamente.

c. Flexibilidad del software. El software ofrece la flexibilidad máxima por lo que un desarrollador puede expresar casi cualquier clase de abstracción y no necesita estar pendiente de distribuidores u otros entes para que les provean los elementos necesarios para la construcción. Suele empujar al desarrollador a construir por sí mismo prácticamente todos los bloques fundamentales sobre los que se apoyan estas abstracciones de más alto nivel.

d. Caracterización de problemas discretos. El software se maneja de manera determinista, o sea que se asume como un sistema discreto. Se comporta siempre de la misma manera manteniendo de forma constante su comportamiento y sus funciones.

Cuanto más complejo es un sistema, más abierto está al derrumbamiento total. El fracaso en dominar la complejidad del software lleva a proyectos retrasados, que exceden el presupuesto y que son deficientes respecto a los requerimientos fijados.
es una parte de lo denominado **Crisis del software**

La ingeniería de software busca apoyar el desarrollo de software profesional o de dimensión industrial en lugar de la programación individual. Incluye técnicas que apoyan la especificación, el diseño y la evolución del programa.
Este conjunto de técnicas, métodos y herramientas posibilita precisamente "dominar la complejidad". se refiere también a toda la documentación asociada y los datos de configuración requeridos para hacer que estos programas operen de manera correcta.

LA INGENIERÍA DE SOFTWARE

Características del software

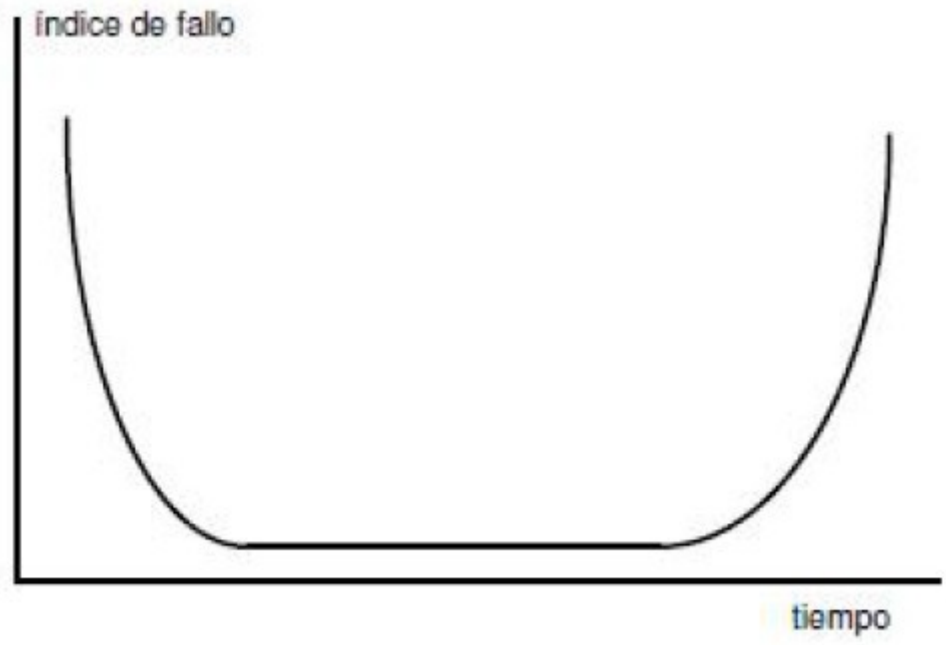
el software incluye no sólo los programas de computadora, sino también las estructuras de datos que manejan esos programas y toda la documentación que debe acompañar al proceso de desarrollo, mantenimiento y uso de dichos programas. El software es inmaterial y por ello tiene unas características completamente distintas al hardware. Entre ellas podemos citar:

1. El software se desarrolla, no se fabrica en sentido estricto.

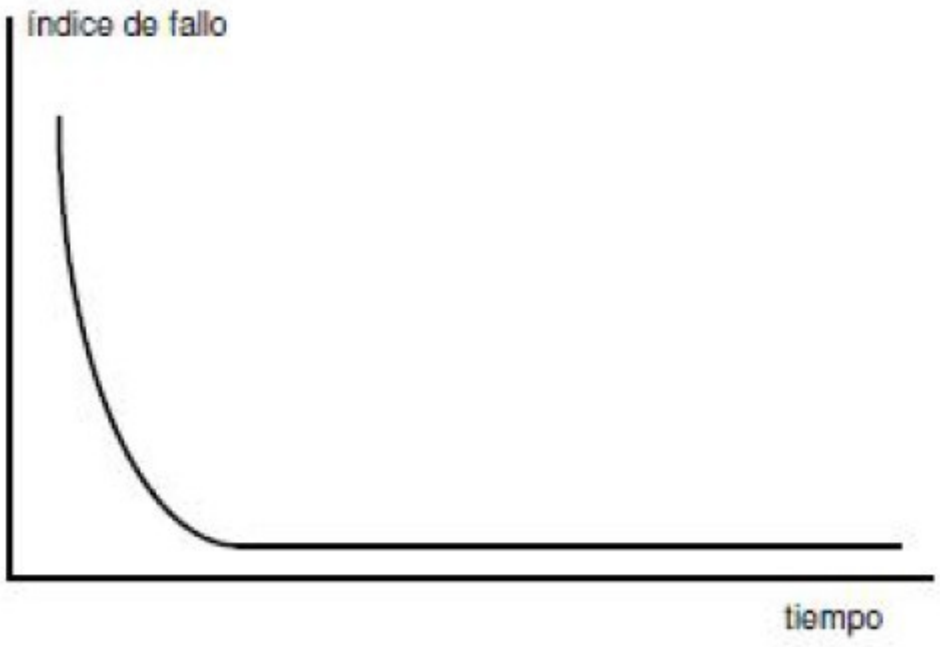
En la fase de producción del software pueden producirse problemas que afecten a la calidad y que en el caso del hardware no existen, o son fácilmente evitables. Por otro lado, en el caso de producción de hardware, el costo del producto acaba dependiendo exclusivamente del costo de los materiales empleados y del propio proceso de producción. Entonces los costos del software se encuentran en la ingeniería y no en la producción.

2. El software no se estropea: Se pueden comparar las curvas de índices de fallos del hardware y el software en función del tiempo.

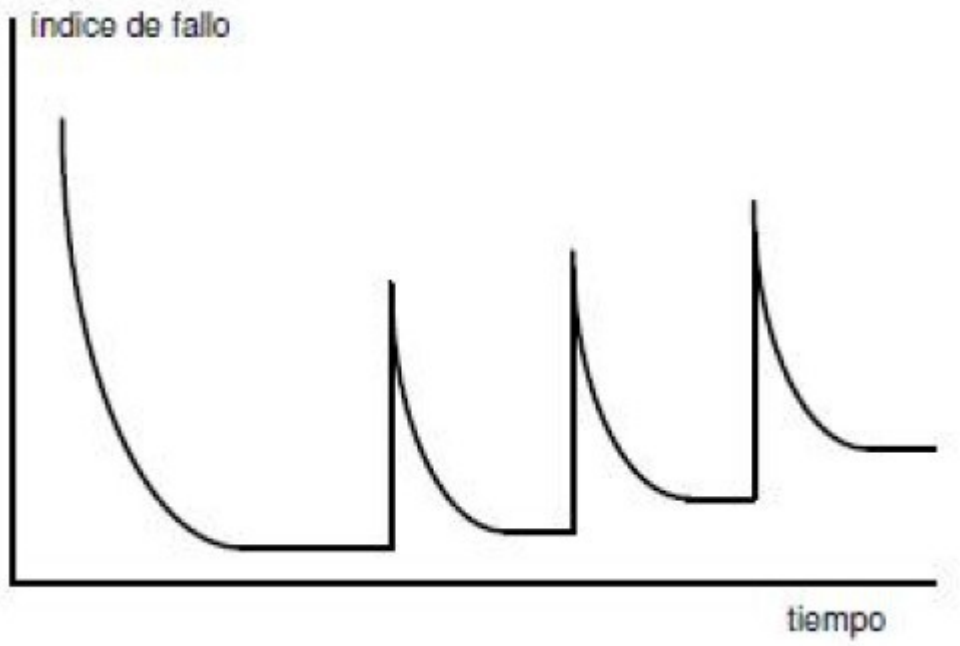
En el hardware fallos son debidos fundamentalmente a defectos de diseño o a la baja calidad inicial de la fase de producción. Una vez corregidos estos defectos, la tasa de fallos cae hasta un nivel estacionario y se mantiene así durante un cierto periodo de tiempo. Posteriormente, la tasa de fallos vuelve a incrementarse debido al deterioro de los componentes.



Para el software, la tasa de fallos es alta, debido a la presencia de errores no detectados durante el desarrollo. Una vez corregidos estos errores, la tasa de fallos debería alcanzar el nivel estacionario y mantenerse ahí indefinidamente.



El software sufre cambios, debidos al mantenimiento. El mantenimiento puede deberse a la corrección de errores latentes o a cambios en los requisitos iniciales del producto. Estos errores pueden corregirse, pero el efecto de los sucesivos cambios hace que el producto se aleje cada vez más de las especificaciones iniciales.



Se puede decir entonces que el software no se estropea, pero se deteriora.
cuando un componente software se deteriora, no se puede sustituir por otro. La solución no es sustituir el componente defectuoso por otro sino un nuevo diseño y desarrollo del producto.

3. La mayoría del software se construye a medida.

Para el diseño de hardware las características se comprueban en un catálogo y se prueban exhaustivamente .Estos componentes cumplen unas especificaciones claras y tienen unas interfaces definidas.

La mayoría del software se fabrica a medida, siendo la reutilización muy baja. Esto hace que el impacto de los costos de ingeniería sobre el producto final sea muy elevado

PROBLEMAS DEL SOFTWARE

La planificación y la estimación de costos son muy imprecisas.
A la hora de abordar un proyecto de una cierta complejidad es frecuente que surjan imprevistos y como consecuencia de esto se producirá una desviación en los costos del proyecto. Sin una planificación detallada es totalmente imposible hacer una estimación de costos Entre las causas de este problema se puede citar:

- No se recogen datos sobre el desarrollo de proyectos anteriores, con lo que no se acumula experiencia que pueda ser utilizada en la planificación de nuevos proyectos.
- Los administradores de proyectos no están especializados en la producción de software. Tradicionalmente, los responsables del desarrollo del software no saben de informática

La productividad es baja
Los proyectos de software tienen, por lo general, una duración mucho mayor a la esperada. Como consecuencia de esto los costos se disparan y la productividad y los beneficios disminuyen. Uno de los factores que influyen en esto, es la falta de unos propósitos claros o realistas a la hora de comenzar el proyecto. Debido a esto son muy frecuentes las modificaciones. Debido a la falta de documentación sobre cómo se ha desarrollado el producto o a que las sucesivas modificaciones han desvirtuado totalmente el diseño inicial, el mantenimiento de software puede llegar a ser una tarea imposible de realizar, pudiendo llevar más tiempo.

La calidad es mala
Como consecuencia de que las especificaciones son ambiguas o incluso incorrectas, y de que no se realizan pruebas exhaustivas, el software contiene numerosos errores cuando se entrega al cliente. Estos errores producen un fuerte incremento de costos durante el mantenimiento del producto, cuando ya se esperaba que el proyecto estuviese acabado.

El cliente queda insatisfecho
Debido al poco tiempo e interés que se dedican al análisis de requisitos y a la especificación del proyecto, a la falta de comunicación durante el desarrollo y a la existencia de numerosos errores en el producto que se entrega, los clientes suelen quedar muy poco satisfechos de los resultados.

DEFINICIÓN DE INGENIERÍA DE SOFTWARE

Ésta disciplina de la ingeniería se interesa por todos los aspectos de la producción de software. El enfoque sistemático que usa la ingeniería de software se conoce como ***Proceso de software***. Un proceso de software es una secuencia de actividades que conducen a la elaboración de un producto de software. Existen cuatro actividades fundamentales comunes a todos los procesos y son:

1. Especificación: Los clientes e ingenieros definen las funcionalidades del software que se producirá y las restricciones de su operación.
2. Diseño e implementación: Se diseña y programa cumpliendo con las especificaciones.
3. Validación: Se verifica el software para asegurar que sea lo que el cliente requiere.
4. Evolución del software, donde se modifica el software para reflejar los requerimientos cambiantes del cliente y el mercado.

Uno de los factores más significativos en la determinación de qué método o técnica aplicar es el tipo de aplicación. Estos tipos pueden ser:
1. Aplicaciones independientes. Se trata de sistemas de aplicación que corren en una PC local e incluyen toda la funcionalidad necesaria sin necesidad de conectarse a una red.
2. Aplicaciones interactivas basadas en transacción. Son las aplicaciones que se ejecutan remotamente y los usuarios acceden desde su propia PC o terminal.
3. Sistemas de control embebido. Sistemas de control de software que regulan y gestionan dispositivos de hardware.
4. Sistema de procesamiento por lotes. Batch. Procesan gran cantidad de entradas individuales para crear salidas correspondientes.
5. Sistemas de entretenimiento. Sistemas de uso personal. Juegos.
6. Sistemas para modelado y simulación. Sistemas que desarrollan científicos o ingenieros para modelar procesos o situaciones físicas que incluyen muchos objetos separados interactuantes.
7. Sistemas de adquisición de datos. Son sistemas que desde su entorno recopilan datos mediante sensores y los envían para su procesamiento a otro sistema.

Existen fundamentos de la ingeniería de software que se aplican a todos los sistemas de software:
• La organización que diseña el software necesita planear el proceso de desarrollo así como tener ideas claras acerca de lo que se producirá y el tiempo en que estará terminado.
• El software tiene que comportarse como se espera, sin fallas y cando se requiera estar disponible.
• Es importante comprender y gestionar la especificación y los requerimientos del software (lo que debe hacer).
• Tiene que usar de manera tan efectiva como sea posible los recursos existentes.

La ingeniería del software abarca un conjunto de tres elementos clave: métodos, herramientas y procedimientos
• Los métodos indican cómo construir técnicamente el software, y abarcan una amplia serie de tareas
• Las herramientas proporcionan un soporte automático o semiautomático para utilizar los métodos.
• Los procedimientos definen la secuencia en que se aplican los métodos

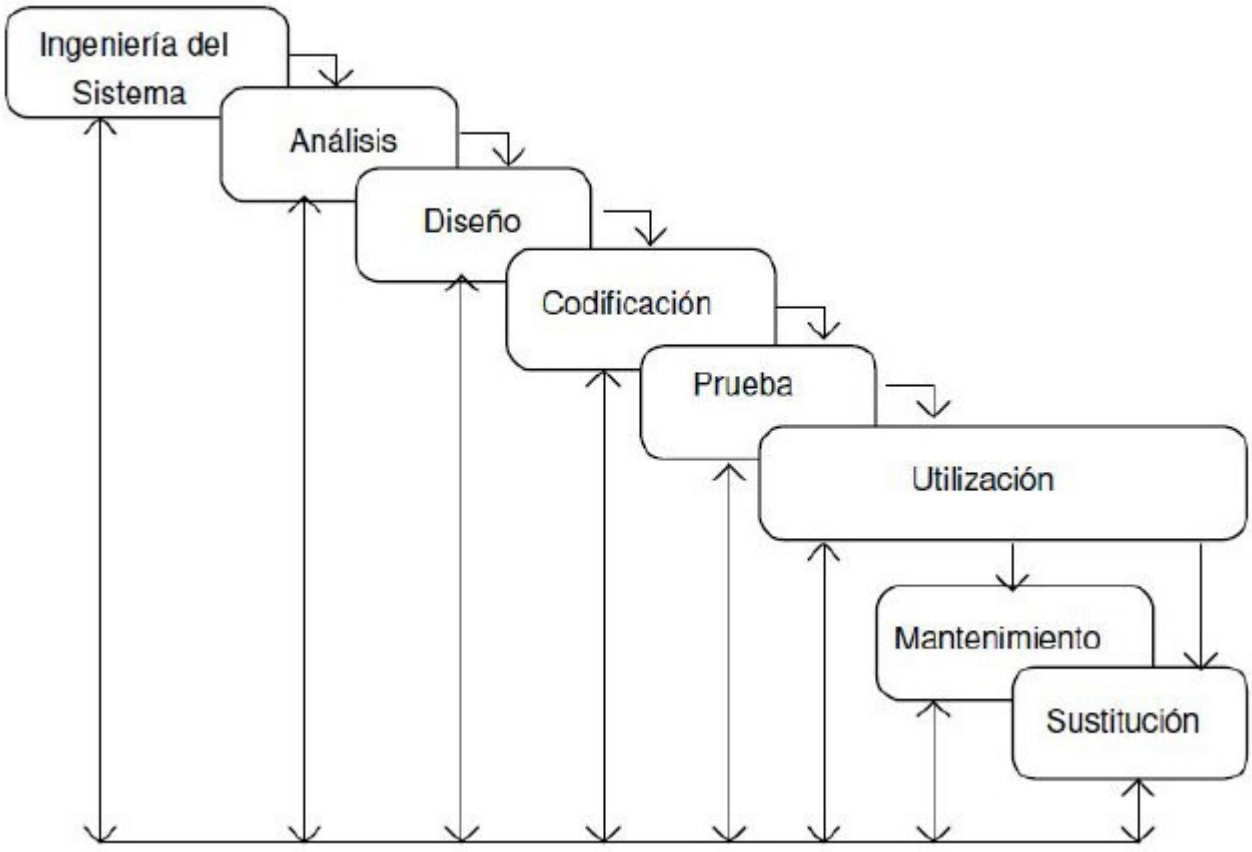
MODELOS DE PROCESO DE SOFTWARE (ciclo de vida)

Por ciclo de vida, se entiende la sucesión de etapas por las que pasa el software desde que un nuevo proyecto es concebido hasta que se deja de usar. Existen diversos modelos de proceso de software o de ciclo de vida, es decir, diversas formas de ver el proceso de desarrollo de software

- Modelo en cascada
- Desarrollo incremental o evolutivo
- Ingeniería de software orientada a la reutilización.

El modelo en cascada (waterfall) o ciclo de vida clásico

Es un ciclo de vida en sentido amplio, que incluye no sólo las etapas de ingeniería sino toda la vida del producto: las pruebas, el uso y el mantenimiento, hasta que llega el momento de sustituirlo.



El ciclo de vida en cascada exige un enfoque sistemático y secuencial del desarrollo de software, que comienza en el nivel de la ingeniería de sistemas y avanza a través de fases secuenciales sucesivas. Estas fases son las siguientes:

Etapa 1: Ingeniería y análisis del sistema:
el primer paso del ciclo de vida de un proyecto consiste en un análisis de las características y el comportamiento del sistema del cual el software va a formar parte.

Etapa 2: Análisis de requisitos del software:
Se debe comprender cuáles son los datos que se van a manejar, cuál va a ser la función que tiene que cumplir el software, cuáles son las interfaces requeridas y cuál es el rendimiento que se espera lograr.

Etapa 3: Diseño
El diseño se aplica a cuatro características distintas del software:
• la estructura de los datos
• la arquitectura de las aplicaciones
• la estructura interna de los programas
• las interfaces
El diseño es el proceso que traduce los requisitos en una representación del software

Etapa 4: Codificación
La codificación consiste en la traducción del diseño a un formato que sea legible para la máquina.

Etapa 5: Prueba
El objetivo es comprobar que no se hayan producido errores en alguna de las fases de traducción anteriores, especialmente en la codificación.

Etapa 6: Utilización
el software se entrega al cliente y comienza la vida útil del mismo.

Etapa 7: Mantenimiento
El software sufrirá cambios a lo largo de su vida útil. Estos cambios pueden ser debidos a tres causas:
- Que, durante la utilización, el cliente detecte errores en el software
- Que se produzcan cambios en alguno de los componentes del sistema informático
- Que el cliente requiera modificaciones funcionales

Etapa 8: Sustitución
La vida del software no es ilimitada y cualquier aplicación, por buena que sea, acaba por ser sustituida por otra más amplia, más rápida o más bonita y fácil de usar.

El ciclo de vida en cascada ha sufrido diversas críticas, debido a los problemas que se plantean al intentar aplicarlo a determinadas situaciones. Entre estos problemas están:
- En la realidad los proyectos no siguen un ciclo de vida estrictamente secuencial como propone el modelo. Siempre hay iteraciones.
- Es difícil que se puedan establecer inicialmente todos los requisitos del sistema. Los requisitos se van aclarando y refinando a lo largo detono el proyecto, según se plantean dudas
- Hasta que se llega a la fase final del desarrollo: la codificación, no se dispone de una versión operativa de las aplicaciones

El modelo de desarrollo incremental o evolutivo

El desarrollo evolutivo se basa en la idea de desarrollar una implementación inicial exponiéndola a los comentarios del usuario y refinándola a través de las diferentes versiones hasta que se desarrolla un sistema adecuado.
Existen dos tipos de desarrollo evolutivo:
- **Desarrollo exploratorio:** donde el objetivo del proceso es trabajar con el cliente para explorar sus

requerimientos y entregar un sistema final.

- **Prototipos desechables:** donde el objetivo del proceso de desarrollo evolutivo es comprender los requerimientos del cliente y entonces desarrollar una definición mejorada de los requerimientos del sistema. El prototipo se centra en experimentar con los requerimientos del cliente que no se comprenden del todo.

un enfoque evolutivo satisface las necesidades inmediatas de los clientes. La especificación se puede desarrollar en forma creciente. El desarrollo incremental refleja la forma en que se resuelven los problemas se avanza en una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores.

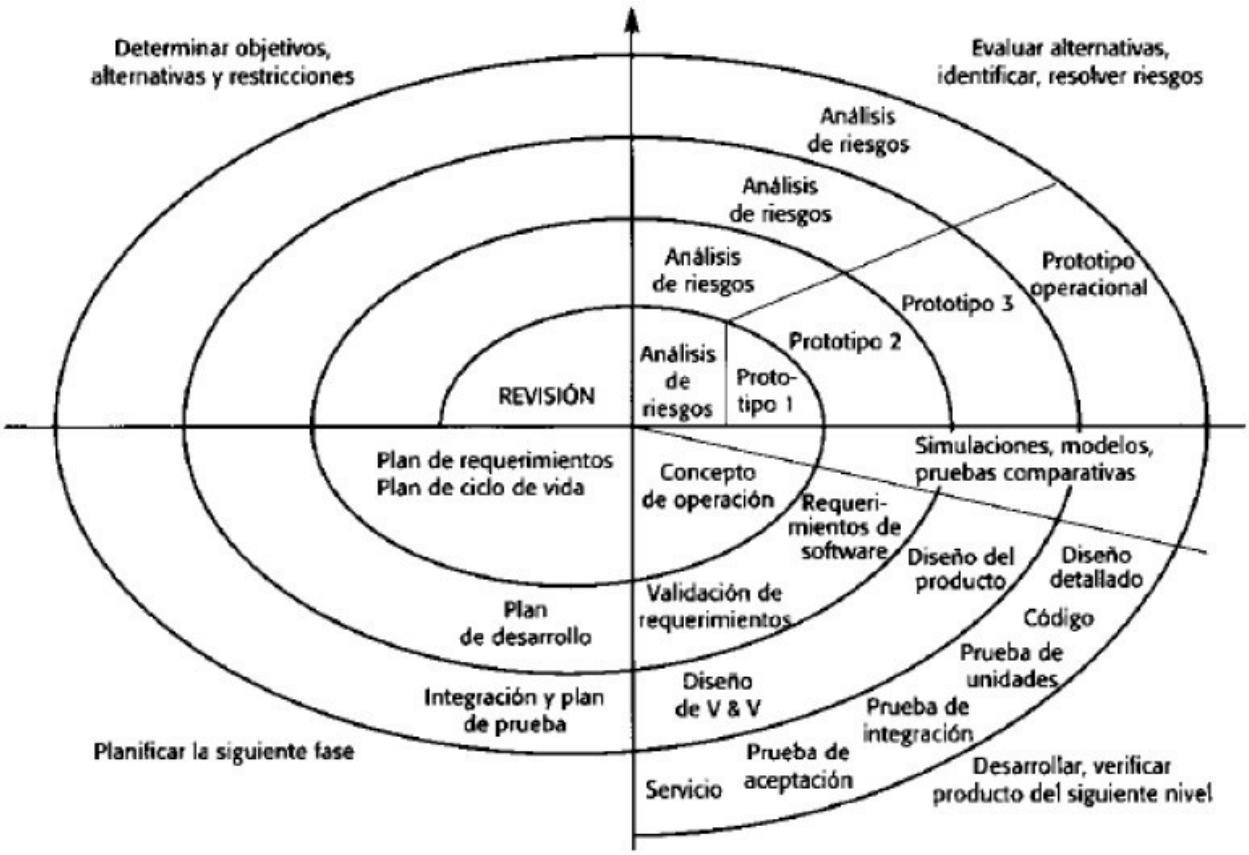
Para sistemas grandes, se recomienda un proceso mixto Esto puede implicar desarrollar un prototipo desechable utilizando un enfoque evolutivo para resolver incertidumbres en la especificación del sistema. Las partes del sistema bien comprendidas se pueden especificar y desarrollar utilizando un proceso basado en el modelo de cascada.

En general, cualquier aplicación que presente mucha interacción con el usuario, o que necesite algoritmos que puedan construirse de manera evolutiva, yendo de lo más general a lo más específico es un buen candidato para este tipo de modelo.
También es conveniente construir prototipos para probar la eficiencia de los algoritmos que se van a implementar, o para comprobar el rendimiento de un determinado componente del sistema y quedarse con el modelo que mejor resultados devuelva. En otros casos, el prototipo servirá para modelar y poder mostrar al cliente cómo va a realizarse la entrada y salida de datos en la aplicación, de forma que éste pueda hacerse una idea de como va a ser el sistema final, pudiendo entonces detectar deficiencias o errores en la especificación aunque el modelo no sea más que una cáscara vacía. No obstante, hay que tener en cuenta que el prototipo ***no es el sistema final***, puesto que normalmente apenas es utilizable.

Uno de los problemas que suelen aparecer siguiendo el paradigma de construcción de prototipos, es que con demasiada frecuencia el prototipo pasa a ser parte del sistema final, bien sea por presiones del cliente, que quiere tener el sistema funcionando lo antes posible o bien porque los técnicos se han acostumbrado a la máquina, el sistema operativo o el lenguaje con el que se desarrolló el prototipo.

El modelo en espiral de Boehm

El modelo en espiral combina las principales ventajas del modelo de ciclo de vida en cascada y del modelo de construcción de prototipos. Permite la utilización de prototipos en cualquier etapa de la evolución del proyecto.



El modelo en espiral define cuatro tipos de actividades, y representa cada uno de ellos en un cuadrante:

Establecimiento de objetivos y planificación

Consiste en determinar los objetivos del proyecto, las posibles alternativas y las restricciones.

Análisis de riesgo

En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso.

Desarrollo y validación

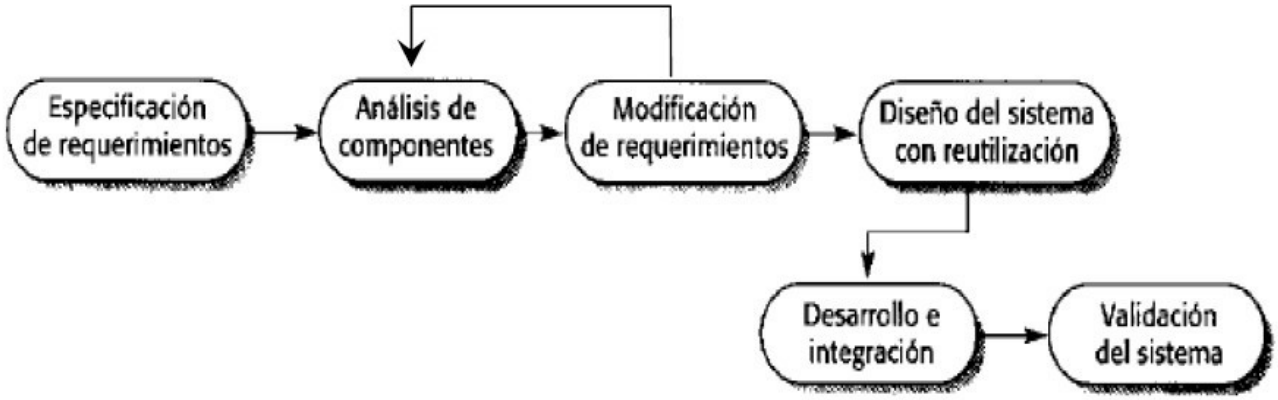
Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema.

Planeación

El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo dela espiral.

Con cada iteración, se construyen sucesivas versiones del software, cada vez más completas, y aumenta la duración de las operaciones del cuadrante de Desarrollo y validación, obteniéndose al final el sistema de ingeniería completo.

Ingeniería de software orientada a la reutilización



En la mayoría de proyectos de software hay cierta reutilización de software. Sucede con frecuencia de manera informal cuando las personas que trabajan en el proyecto conocen diseños o códigos similares a los requeridos. Los buscan, modifican e incorporan en sus sistemas. Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación.

Modificación de requerimientos
En esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles.

Diseño de sistema con reutilización
En esta etapa se diseña el marco conceptual del sistema o se reutiliza el existente.

Desarrollo e integración
Se diseña el software que no puede procurarse de manera externa y se integran los componentes y sistemas comerciales para crear el nuevo sistema.

La ingeniería de software orientada a la reutilización tiene la clara ventaja de reducir la cantidad de software a desarrollar y consecuentemente disminuir costos y riesgos. Pero puede conducir hacia un sistema que no cubra las necesidades reales de los usuarios.