Pedro López Santamaría

UM08889SIS15724

MAESTRIA EN CIENCIAS DE LA COMPUTACIÓN

SEGUNDA FASE

Primera Asignatura

"DESARROLLO DE APLICACIONES EMPRESARIALES"

ATLANTIC INTERNATIONAL UNIVERSITY

Honolulu, Hawai

Abril de 2010

Contenidos

Alcance

En este documento se describe en términos generales el desarrollo de los contenidos y objetivos de la primera asignatura que forma parte integral del currículo previamente aprobado.

Los aspectos generales relacionados con esta asignatura se muestran en el cuadro siguiente:

Primera Asignatura	
Title	**Development of business applications**
Título	**Desarrollo de aplicaciones empresariales**
Descripción	Conocer acerca de las diferentes etapas, metodologías y mejores prácticas del proceso de desarrollo de aplicaciones empresariales.
Objetivo	Comprender como brindar soporte a la gestión de los procesos de negocios aplicando las metodologías de desarrollo de aplicaciones (análisis, diseño, programación y pruebas).
Actividades	➢ Especificación conceptual de necesidades ➢ Análisis de requerimientos especificados ➢ Diseño inicial y general ➢ Diseño detallado, codificación, depuración y liberación
Referencias	➢ http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html ➢ http://sunset.usc.edu/cse ➢ http://www.monografias.com/trabajos5/inso/inso.shtml ➢ http://www.um.es/giisw/isbc ➢ http://www.sistemas.unam.mx/software.html ➢ http://es.wikipedia.org/wiki/Ingenieria_de_software
Bibliografía	➢ Introducción a la ingeniería de software, Manuel Imaz ➢ Ingeniería del Software: Un Enfoque Práctico, Roger Pressman

Es de particular importancia hacer notar que el énfasis principal de esta asignatura se centra en la investigación de las metodologías y mejores prácticas orientadas a facilitar los procesos de desarrollo de sistemas aplicativos, que puedan ser capaces de hacer disponible sobre la plataforma Internet, la prestación de los servicios que brindan las organizaciones.

Como es comprensible, estas metodologías orientadas a facilitar el desarrollo de aplicaciones pueden llegar a cubrir un amplio conjunto de otras facilidades complementarias, sin embargo, estas facilidades adicionales no forman parte del objetivo particular que se pretende alcanzar mediante el desarrollo de esta signatura.

Adicionalmente, la presentación de los contenidos de esta investigación se desarrolla de manera simplificada y general, utilizando un vocabulario lo menos técnico posible, con la finalidad de facilitar la mayor comprensión posible a los lectores no especializados en estos temas. Por tal motivo, se han omitido las exposiciones de procedimientos basados en sentencias, instrucciones o comandos de computadora.

Terminología

Con la finalidad de facilitar el mayor entendimiento posible, en toda disciplina del conocimiento humano es imprescindible el establecimiento de las definiciones y convenciones apropiadas, las cuales han de conformar el marco de referencia común acerca de los temas tratados.

Como es comprensible, todas aquellas personas involucradas de alguna manera con los procesos de desarrollo de aplicaciones, también se enfrentan con la realidad de conocer los términos propios de esta área del conocimiento en acelerada expansión.

En este apartado se exponen algunos de los términos básicos y necesarios para facilitar un acercamiento provechoso de los contenidos desarrollados en este documento. Adicionalmente, conviene dar a conocer que dichos términos no son los únicos utilizados en esta disciplina, ni tampoco las descripciones aquí expuestas hacen referencia a todas las usanzas posibles de los mismos.

Algunos de los conceptos elementales y sus correspondientes descripciones, estrechamente relacionados con las metodologías orientadas al desarrollo de aplicaciones, son expuestos a continuación:

- **Programación Orientada a Objetos**

  La Programación Orientada a Objetos (POO u OOP según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan *estado* (es decir, datos), *comportamiento* (esto es, procedimientos o *métodos*) e identidad (propiedad del objeto que lo diferencia del resto).

  La programación orientada a objetos expresa un programa como un conjunto de estos componentes, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

- **Herramientas CASE**

  Las herramientas conocidas con el nombre de “Ingeniería de Software Asistida por Computadora” (CASE, *Computer Aided Software Engineering*) corresponden a un grupo de aplicaciones orientadas a incrementar la productividad en los procesos de desarrollo de software, con la finalidad de reducir los costos y los tiempos asociados a estos procesos.

El alcance funcional de estas aplicaciones asistentes puede considerar herramientas orientadas a apoyar las labores a lo largo de todo el ciclo de desarrollo de los productos de software. Partiendo desde el diseño mismo de los proyectos, la estimación de costos y tiempos, la generación automática de parte del código requerido, la compilación automática del código generado, la construcción de la documentación técnica y funcional a partir del diseño, y finalmente, la detección de inconsistencias y errores.

- **Desarrollo Rápido de Aplicaciones**

  El proceso conocido como Desarrollo Rápido de Aplicaciones (RAD, *Rapid Application Development*) consiste en una metodología orientada a facilitar la construcción de software, la cual fue propuesta inicialmente por el autor James Martin de los laboratorios de IBM.

  El método comprende el desarrollo interactivo, la construcción de prototipos, y la utilización de herramientas asistentes CASE (*Computer Aided Software Engineering*).

  Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, la utilidad y la rapidez de ejecución.

Introducción

En las últimas décadas del siglo XX, las reducciones de costo en los dispositivos electrónicos han contribuido a que el software llegue a ser un elemento que participa en muchos de los dispositivos utilizados por las sociedades industrializadas.

El software se halla cada vez más presente en la sociedad como elemento incorporado a diversos mecanismos arraigados. Los automóviles modernos, por ejemplo, poseen entre 10 y 100 procesadores para dirigir todo tipo de funciones, que van desde la gestión del reproductor de música hasta el control del sistema de frenado.

Importancia del software

Según datos tomados de la obra *Software Conspiracy, Mark Minasi, McGraw Hill, 2001*, en el año 1996, la contribución del software a la economía de los E.E.U.U., se resume a continuación:

- es la principal fuente de superávit por exportaciones en la balanza comercial
- 24,000 millones de dólares de ingresos por exportaciones de software y 4,000 millones gastados en importaciones, arrojan un superávit anual de 20,000 millones de dólares
- otros datos comparativos generales (también en millones de dólares): agricultura, 26-14-12; industria aeroespacial; 11-3-8; industria química, 26-19-7; industria automovilística, 21-43-(22); productos manufacturados, 200-265-(64)

Adicionalmente, el software también ejerce un papel muy influyente en la infraestructura general de la economía. No sólo tiene un papel determinante en Internet, sino también en otros importantes sectores, tales como transporte, energía, medicina y finanzas.

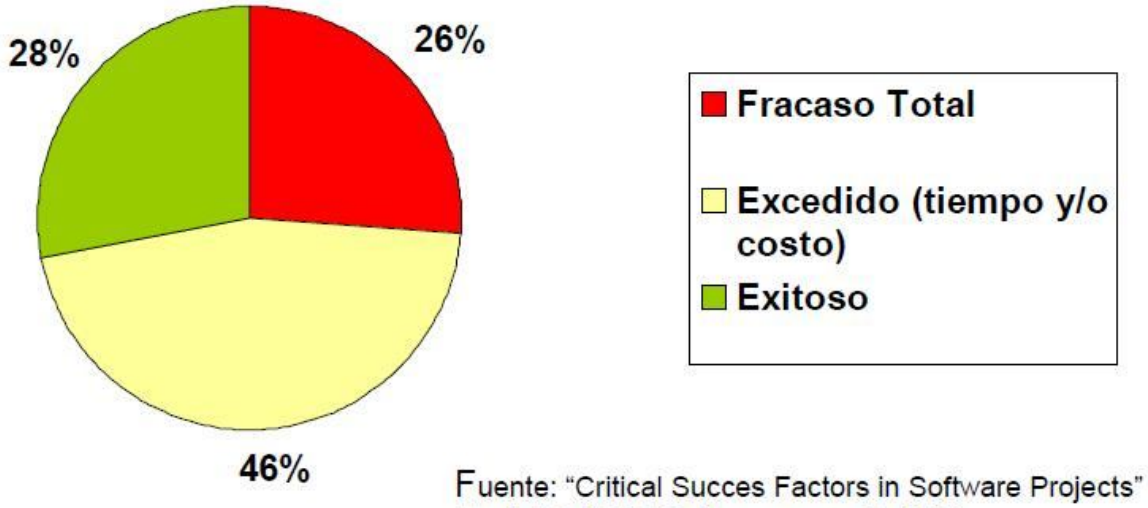
Los costos involucrados con la relación entre la adquisición de software y hardware cada vez más se aproximan a cero. El costo inicial de la propiedad del software es aproximadamente 5 veces el costo del hardware, sin embargo, el grupo consultor Gartner estima que en la actualidad el costo aproximado por mantenimiento de una computadora personal a lo largo de 5 años asciende a 7 dólares por cada kilobyte de memoria instalada en la misma.

Problemas evidentes

En un estudio llevado a cabo por la firma IBM en el año 1994, se revelan los resultados alarmantes mencionados a continuación en forma general:

- el 55% de los sistemas costaron más de lo previsto
- el 68% excedieron el tiempo previsto para su desarrollo
- el 88% se tuvo que volver a diseñar por completo

Como referencia histórica, en la siguiente ilustración se puede visualizar una representación gráfica del éxito general logrado en los proyectos de software emprendidos en el año 1998.



Fuente: "Critical Succes Factors in Software Projects" Reel, J.S. IEEE Software mayo de 1999

Al concluir el largo proceso de desarrollo del Sistema de Automatización Avanzada (FAA, 1982-1994), se pueden observar las siguientes conclusiones:

- el promedio de producción industrial estaba estimado en 100 dólares por cada línea de código fuente, y se preveía pagar 500 dólares por línea
- se terminó por pagar entre 700 a 900 dólares por línea
- el monto total por trabajos cancelados ascendió a 6,000 millones de dólares

Por otra parte el Departamento de Estadística Laboral (1997) reveló algunos hallazgos poco alentadores, los cuales son mencionados a continuación:

- dos de cada seis nuevos sistemas que se ponen en funcionamiento sufren cancelaciones
- los sistemas grandes tienen aproximadamente un 50% de probabilidades de ser cancelados

- la medida de tiempo empleado en un proyecto usualmente se excede al menos en un 50% con respecto al plazo previsto
- al menos tres cuartas partes se los sistemas son considerados “fracasos operativos”

Accidentes

Algunos expertos en diferentes disciplinas de la informática coinciden en que “*la manera más probable de destruir el mundo es por accidente*”. Es en este contexto en donde entran en el juego los desarrolladores de software – los expertos continúan indicando “*nosotros somos los que provocamos los accidentes*”.

Nathaniel Borenstein, constructor de MIME en su obra Programming as if People Mattered Friendly Programs, Software Engineering and Other Noble Delusions, Princeton University Press, Princeton, NJ, 1991, expone el caso denominado Therac-25 (1985-87).

- se trataba de un aparato de radioterapia dotado con un controlador de software
- se retiró el mecanismo de interbloqueo a nivel del hardware, pero el software no tenía soporte para la funcionalidad de interbloqueo
- el software falló al mantener las constantes vitales: un flujo de electrones, es decir, un flujo más intenso de radiación mediante una placa para generar rayos X
- a consecuencia de ello se produjeron varias muertes por quemaduras
- el programador no tenía experiencia en programación concurrente
- para mayor información favor consultar: http://sunnyday.mit.edu/therac-25.html

Se podría llegar a pensar que se aprendería de la experiencia y que un desastre de este tipo no volvería a suceder jamás. Sin embargo, la Agencia Internacional de Energía Atómica anunció una “emergencia radiológica” el 22 de mayo del 2001 en Panamá. Algunos de los detalles más significativos de este caso se mencionan a continuación:

- 28 pacientes sufrieron sobreexposición: 8 murieron, 3 de ellos como consecuencia directa de la sobreexposición; con la probabilidad de que al menos tres cuartas partes de los 20 que sobrevivieron pudieran desarrollar “serias complicaciones, las cuales en algunos casos, eventualmente, podrían resultar mortales”
- los investigadores indicaron que el equipo de radioterapia “funcionaba perfectamente”; y señalaron que la razón de la emergencia estaba relacionada con la “entrada de datos”
- si los datos se introdujeron en varios bloques protegidos dentro de un lote, entonces, la dosis de radiación se computó de manera incorrecta
- al menos, la FDA llegó a la conclusión de que “la interpretación de los datos del bloqueo de flujo controlado por software” fue uno de los factores causantes del desastre

- para obtener mayor información relacionada con este caso favor visitar el sitio de Internet: http://www.fda.gov/cdrh/ocd/panamaradexp.html

La Agencia Espacial Europea documentó el caso denominado Ariane–5, ocurrido en junio de 1996. Algunos de los detalles más relevantes de este caso se mencionan a continuación:

- pérdida absoluta de misiles no tripulados poco después del despegue
- causado por una excepción en el código de un programa escrito en lenguaje Ada
- ni siquiera se precisó la sección defectuosa del código posteriormente
- se debió a un cambio en el entorno físico, es decir, se infringieron supuestos no documentados
- mayor información: http://www.esa.int/htdocs/tidc/Press/Press96/ariane5rep.html

En los desastres provocados por fallas de software, son más comunes los casos de accidentes similares al de Ariane que los causados por los aparatos de radioterapia. No es muy probable que los errores en el código sean la causa; normalmente, la causa del problema se remonta al análisis de las necesidades, en este caso, a deficiencias que ocurren en el momento de articular y evaluar presunciones claves acerca del entorno.

A continuación se resume un caso relacionado con el Servicio de Ambulancias de Londres ocurrido en el año 1992:

- pérdida de llamadas, doble servicio brindado debido a llamadas duplicadas
- a consecuencia de una mala selección del programador: experiencia insuficiente
- mayor información: http://www.esa.int/htdocs/tidc/Press/Press96/ariane5rep.html

El desastre del Servicio de Ambulancias de Londres se debió en realidad a fallas de gestión. Los informáticos que coordinaron el desarrollo del software pecaron de ingenuidad y aceptaron una oferta de una empresa desconocida, la cual estaba menos calificada que las otras compañías más acreditadas. Adicionalmente, cometieron el terrible error de poner en producción repentinamente con el nuevo sistema, sin detenerse a comparar y validar la ejecución del mismo con los resultados del sistema ya existente.

En el corto plazo, es posible anticipar que estos problemas pueden llegar a acentuarse debido al uso generalizado del software en nuestra infraestructura cívica y social.

"La demanda de software ha crecido mucho más rápido que nuestra capacidad para construirlo de manera confiable. Además, aumenta la demanda por obtener un tipo de software más práctico, fiable y robusto que el que se está desarrollando hoy en día. Nos hemos hecho peligrosamente dependientes de los grandes sistemas de software, cuyo

comportamiento no es del todo comprensible, y que a menudo tienden a fallar de manera imprevista".

Calidad del software

Como es muy comprensible, el nivel de la calidad alcanzado por un software determinado no se puede medir únicamente por la cantidad de errores o fallas. Es posible que se pueda probar un software y depurarlo, eliminando la mayoría de los errores posibles que pueden hacer que falle, pero al final nos encontraremos ante un programa que es imposible utilizar, y que la mayoría de las veces no logra hacer lo que se espera, debido a que presenta muchos casos especiales. Para solucionar este problema, es preciso considerar la calidad desde el principio.

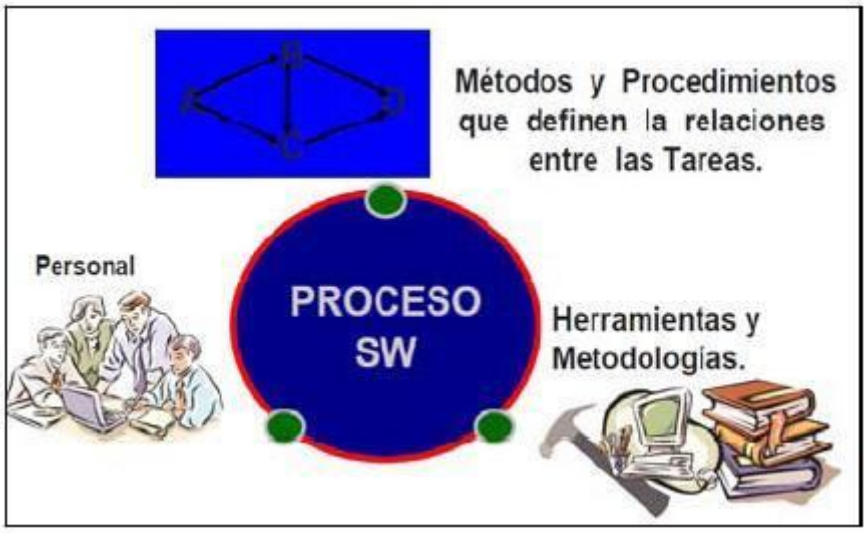
Importancia del diseño

¿Sabemos lo que nos hace falta para poder producir software de buena calidad?. John Murria, experto en control de calidad del software de FDA, citado en la obra Software Conspiracy, Mark Minasi, McGraw Hill, 2000, expone la siguiente declaración: “En los Estados Unidos, la calidad de los automóviles mejoró cuando Japón mostró a la comunidad industrial internacional otros métodos más eficaces de fabricación. De igual manera, alguien tendrá que enseñar a la industria del software que éste también se puede desarrollar de un modo más eficiente”.

Resulta bastante evidente reconocer que el código o la programación avanzada no lo es todo a la hora de construir software. En la realidad práctica, supone solamente una pequeña parte. No se debe llegar a pensar ingenuamente en el código como parte de la solución; ya que de hecho, normalmente forma parte del problema.

Entonces, para comenzar, conviene encontrar palabras distintas a “código” para referirnos al software, otros términos que sean menos dramáticos, más directos y menos vinculados a la tecnología, ya que en breve se han de quedar obsoletos.

En la siguiente ilustración es posible visualizar un esquema generalizado y simplificado que muestra las diferentes categorías de elementos que intervienen en los procesos de construcción de productos de software.



Algunas recomendaciones que conviene considerar en el diseño a la hora de planificar la construcción de software de infraestructura, por parte de los diseñadores se mencionan a continuación:

- Pensar a largo plazo nunca viene mal (¡y es barato!)
- No es posible añadir calidad al final del proceso: se hace necesario comprarla a cada paso, confiando más en las revisiones periódicas y continuas; resulta más efectivo y mucho menos costoso
- Fomentar al máximo la delegación de tareas y el trabajo en equipo
- Un diseño defectuoso perjudica al usuario: esto normalmente da como resultado software difícil de utilizar, incoherente y poco flexible
- Un diseño defectuoso también afecta al programador: redunda en interfaces pobres, significativa proliferación de errores, y mayores dificultades para añadir nuevas funcionalidades posteriormente

Resistencia al cambio

No deja de ser curioso que muchos estudiantes de informática suelan resistirse a considerar la construcción de software como una labor de ingeniería. Quizás esto se deba a que piensan que las técnicas de ingeniería le restarán lo místico a su trabajo o que no se adecuarán a su don de innatos virtuosos. Quizás piensen que la aplicación de técnicas de ingeniería hace su trabajo menos emocionante, o que éstas pueden restar vistosidad y mérito a sus dones inherentes para la programación. Sin embargo, las técnicas de ingeniería para el desarrollo de software suelen permitir que el talento de los estudiantes alcance mayores niveles de eficacia.

De la obra Comparing the Effectiveness of Software Testing Strategies., Victor R. Basili y Richard W. Selby: IEEE Transactions on Software Engineering Vol. SE-13, No. 12, diciembre de 1987, págs. 1278 – 1296, se pude concluir que incluso los programadores profesionales se engañan a sí mismos. Durante un experimento, en el cual participaron 32 informáticos de la NASA se pusieron en práctica tres técnicas diferentes para probar algunos programas de pequeño tamaño.

A estos programadores se les solicitó que evaluaran la proporción de errores que esperaban encontrar en cada método. Los resultados demostraron que sus intuiciones eran erróneas. Creyeron que las revisiones realizadas utilizando la modalidad de caja negra sobre las especificaciones era el método más efectivo, pero en realidad la modalidad de lectura directa de código resultó ser más eficaz (a pesar de que el código no estaba documentado). Al aplicar el método de pruebas basado en la lectura de código, ¡encontraron los errores en la mitad de tiempo!.

Técnicas de ingeniería

El proceso de construcción de software es materia de la disciplina informática conocida como *Ingeniería de Software*, la cual en términos muy generales consiste en un proceso complejo que involucra diversas tareas relacionadas con el desarrollo y la gestión de los sistemas informáticos.

La ingeniería de software como disciplina considera varios modelos o paradigmas de desarrollo, en los cuales se puede apoyar para lograr la construcción de software en forma ordenada y sistemática. Entre los métodos más destacados se encuentran el *Modelo en Cascada* y el *Modelo en Espiral*, principalmente por ser los más utilizados y más completos.

La descripción, propósitos y modelos de desarrollo aplicables en la ingeniería de software serán comentados brevemente más adelante en este documento académico.

Resumen

Es importante mencionar, que el desarrollo de los contenidos correspondientes a esta asignatura académica se realiza en tres secciones principales, a saber: "*Ingeniería de Software*", "*Desarrollo en Cascada*" y "*Desarrollo en Espiral*".

En cada una de estas secciones se exponen los aspectos más sobresalientes de estas metodologías orientadas a apoyar el desarrollo de sistemas aplicativos; iniciando con una descripción general de cada uno de estos procesos, y finalizando con la enumeración de algunas de sus ventajas y desventajas más relevantes.

Este documento culmina con un apartado de conclusiones, en el cual se intenta exponer una serie de comentarios que evidencian el grado de aprovechamiento obtenido, a medida que el autor ha avanzado y profundizado en el desarrollo de los contenidos de esta asignatura.

Primera Sección
Ingeniería de Software

Capítulos.

1. Descripción
2. Ventajas
3. Desventajas

Capítulo 1
Ingeniería de Software: Descripción

En contraposición a los elementos físicos que integran los sistemas computacionales, conocidos con el nombre de *hardware*, se suele denominar *programas*, *equipamiento lógico* o *software*, a los diversos conjuntos de componentes intangibles, que conforman las instrucciones y datos necesarios para que una computadora pueda funcionar correctamente, de tal manera que ésta pueda realizar y completar satisfactoriamente alguna tarea determinada.

Definición

Probablemente la definición de *software*, mayormente aceptada desde el punto vista formal, es la atribuida al Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, *The Institute of Electrical and Electronics Engineers*), en su estándar 729, la cual se expresa así:

«Software es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de cómputo».

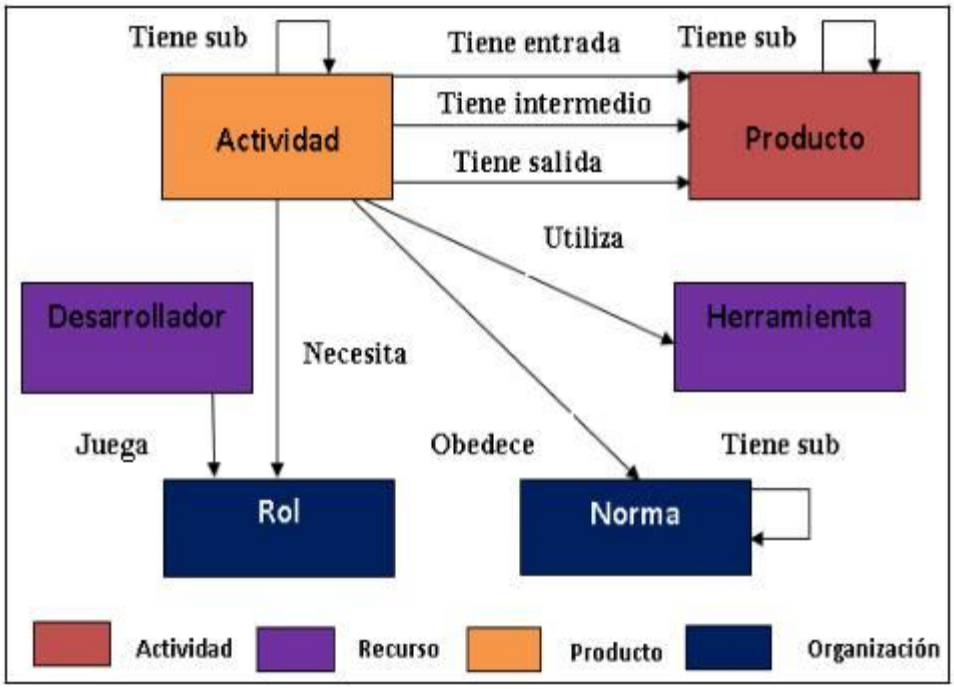
Bajo esta definición, el concepto de software va más allá de los programas de cómputo en sus todas sus formas: código fuente, interpretable, binario o ejecutable, sino que además se debe incluir la documentación y todos los otros elementos relacionados: es decir, abarca todo lo intangible que rodea al hardware.

El término *software* fue usado por primera vez en este sentido por el matemático y estadista estadounidense John W. Tukey en 1957. Sin embargo, en las ciencias de la computación y la ingeniería de software, este término corresponde únicamente a los elementos intangibles que se encuentran involucrados en el procesamiento de información, realizado mediante los sistemas informáticos: es decir, suele estar en referencia directa a los *programas* y los *datos* únicamente.

Proceso de construcción

Existe la tendencia general de identificar el proceso de construcción de un programa informático con la programación. Formalmente, esta vinculación es cierta cuando se trata de programas pequeños para uso didáctico o personal, sin embargo, dista en gran medida cuando se trata de proyectos para desarrollar sistemas informáticos de gran complejidad y envergadura.

En la siguiente figura se puede visualizar una representación generalizada y simplificada, que muestra los diferentes tipos de elementos utilizados para modelar o representar los procesos de construcción de software, así como sus correspondientes relaciones potenciales.



En términos muy generales, el proceso formal de construcción de software desde el punto de vista de la ingeniería involucra los siguientes pasos:

- *Identificar la necesidad*. Consiste en reconocer la carencia de un programa para solucionar un problema ó identificar la posibilidad de automatización de una tarea.
- *Recoger los requisitos del programa*. Se debe especificar claramente qué es lo que debe hacer el programa y para qué se necesita.
- *Realizar el análisis de los requisitos del programa*. Se debe establecer con claridad cómo debe realizar el programa las cosas que debe hacer. También es necesario especificar en esta fase, cuáles han de ser las pruebas necesarias para comprobar la validez del programa.
- *Diseñar la arquitectura del programa*. Se debe descomponer el programa en partes de complejidad abordable.
- *Implementar el programa*. Consiste en realizar un diseño detallado, especificando completamente todo el funcionamiento del programa, tras lo cual, la fase de codificación debería resultar inmediata.

- *Implantar (instalar) el programa*. Consiste en poner el programa en funcionamiento, junto con los componentes que puedan ser requeridos (bases de datos, redes de comunicaciones, etcétera).

Es posible afirmar entonces, que la *Ingeniería del Software* se centra en los aspectos de planificación y diseño del programa, mientras que antiguamente la realización de un programa consistía únicamente en escribir el código (programación artesanal).

Tipología

Es posible clasificar los sistemas informáticos (software) en múltiples y variadas maneras, de acuerdo al criterio elegido para su clasificación. Sin embargo, es necesario reconocer que la agrupación del software en categorías, siempre resulta en cierto modo, arbitraria, y muchas veces difusa y confusa.

A continuación se muestra un ejemplo de clasificación general basado en el criterio de utilización o aprovechamiento:

- *Software de sistema.* Son los programas que permiten el adecuado funcionamiento de los componentes de hardware. Su objetivo es abstraer tanto como sea posible al desarrollador de aplicaciones de los detalles internos del computador particular que se utiliza, especialmente de las características físicas de la memoria, dispositivos de comunicaciones, impresoras, pantallas, teclados, etcétera. Incluye entre otros: *sistemas operativos, controladores de dispositivos (drivers), herramientas de diagnóstico, servidores, sistemas de ventanas, y utilitarios.*
- *Software de desarrollo*. Generalmente son programas orientados a proporcionar facilidades para ayudar al desarrollador a construir programas informáticos, mediante la utilización de diferentes lenguajes de programación. Incluye entre otros: *editores de texto, compiladores, intérpretes, enlazadores, depuradores, entornos de desarrollo integrados (IDE).*

- *Software de aplicación*. Estos son los programas que permiten a los usuarios finales llevar a cabo una o varias tareas específicas, en cualquier campo de trabajo susceptible de ser automatizado o asistido, con especial énfasis en la gestión de las actividades humanas. Incluye entre otros: *diversas aplicaciones de negocios, empresariales, organizacionales y gubernamentales, aplicaciones ofimáticas y del hogar, bases de datos, y aplicaciones de entretenimiento y multimedia (audio, video, y juegos).*

Formas

El software es representado en múltiples formas durante diferentes momentos de su ciclo de construcción. Algunas de las formas más significativas se mencionan a continuación:

- *Código fuente*. Se refiere a las instrucciones escritas directamente por los desarrolladores. Contiene el conjunto de instrucciones específicas que deben ser ejecutadas por la computadora.

- *Código objeto*. Se origina como resultado de la utilización de un compilador sobre el código fuente. Consiste en una representación traducida a otro lenguaje de las instrucciones contenidas en código fuente. El código objeto no es directamente entendible por el ser humano, pero tampoco es directamente ejecutable por la computadora. Se trata de una representación intermedia entre el código fuente y el código ejecutable.

- *Código ejecutable*. Se obtiene como resultado de enlazar uno o varios fragmentos de código objeto. Constituye un archivo en representación binaria con un formato tal que el sistema operativo es capaz de cargar dentro de la memoria de una computadora, y proceder a su ejecución. El código ejecutable es directamente entendible por la computadora.

Programación

Se suele denominar programación a la actividad humana orientada a la construcción de un programa de computadora, mediante la especificación de un conjunto detallado de instrucciones que una computadora puede ejecutar.

Un algoritmo es una secuencia clara (no ambigua), finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa normalmente implementa (traduce a un lenguaje de programación específico) un algoritmo. Es importante reconocer que es la secuencia de instrucciones en sí misma la que debe ser finita, no el número de veces que estas instrucciones pueden ser ejecutadas.

Los programas se suelen subdividir en partes menores (módulos), de modo tal que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa.

A lo largo del tiempo se han propuesto diversas técnicas de programación, cuyo objetivo es mejorar tanto el proceso de construcción de software como su mantenimiento posterior. Entre ellas se pueden mencionar la programación lineal, estructurada, modular y orientada a objetos.

Los requisitos más relevantes y fundamentales que se deben cumplir durante el proceso de desarrollo de un programa se mencionan a continuación:

- *Corrección*. Un programa es correcto si hace lo que debe hacer. Para determinar si un programa cumple en forma satisfactoria es muy importante especificar claramente qué debe hacer antes de empezar a construirlo, y una vez terminado es necesario compararlo con lo que realmente hace.

- *Claridad*. Es primordial que el código del programa fuente sea lo más claro y legible posible para mejorar el mantenimiento posterior del software. Durante todo el proceso de escritura del código fuente, se deben buscar errores y corregirlos. Más concretamente, cuando el programa está concluido, se requiere hacer ampliaciones o modificaciones, según la demanda de las nuevas necesidades, esta labor puede ser llevada a cabo por el mismo programador que construyó el programa o por otros.

- *Eficiencia*. Debe consumir la menor cantidad de recursos posibles. Normalmente al hablar de eficiencia se suele hacer referencia al consumo de tiempo (procesamiento) y a la utilización de memoria (almacenamiento).

La eficiencia y la claridad de un programa pueden ser objetivos contrapuestos. Muchas veces es posible obtener mayor claridad sacrificando parte de la eficiencia o viceversa.

Metodologías

La ingeniería de software se puede considerar como la aplicación de los principios científicos de la ingeniería al proceso de desarrollo y gestión del software.

Esto mediante la utilización en la forma más eficiente y óptima posible de herramientas preestablecidas; objetivo que siempre busca la ingeniería. No comprende únicamente la resolución de problemas, sino más bien, teniendo en cuenta las diferentes soluciones, entonces, elegir la más apropiada.

Durante décadas la meta de la ingeniería de software ha sido encontrar procesos o metodologías predecibles y repetibles que mejoren la productividad y la calidad.

En la ingeniería de software se tienen disponibles varios enfoques metodológicos o paradigmas que se pueden utilizar para realizar la construcción metódica de software, entre los cuales podemos destacar los descritos a continuación, principalmente, por ser los más utilizados y los más completos:

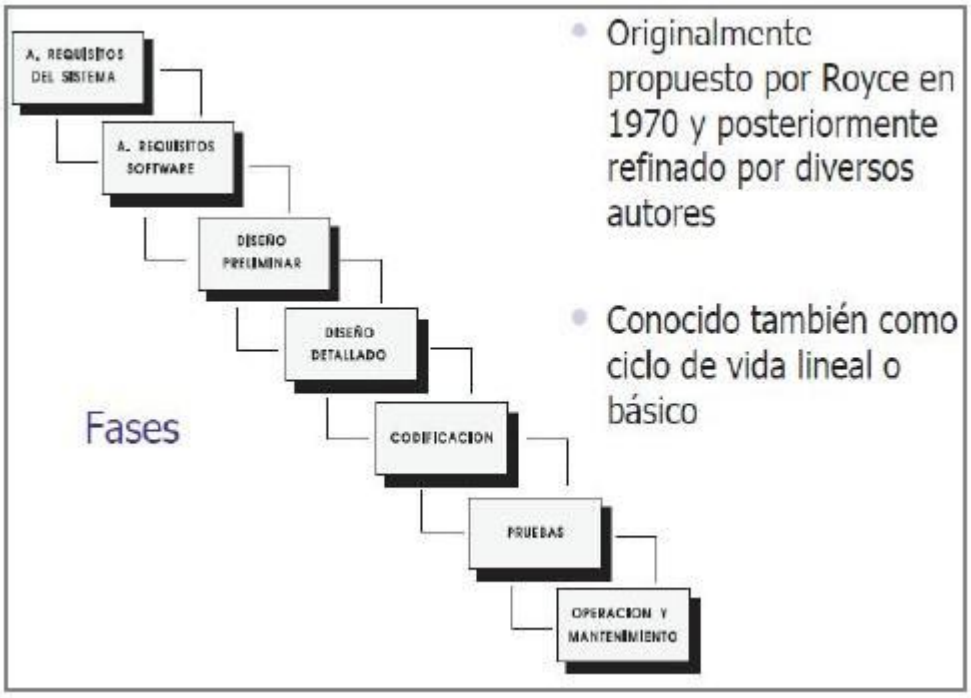
- *Modelo en cascada*.

Permite ordenar rigurosamente las etapas del ciclo de vida del software, de forma tal que el inicio de cada etapa debe esperar a la finalización de la etapa inmediatamente anterior.

Un ejemplo generalizado de estas etapas es el siguiente: *Análisis de Requisitos*, *Diseño del Sistema*, *Diseño del Programa*, *Codificación (Programación)*, *Pruebas*, *Implantación*, y *Mantenimiento*.

De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación de los recursos afectados, aumentando en forma muy significativa los costos generales del desarrollo.

En la siguiente figura se puede visualizar una representación generalizada y simplificada que muestra las fases o etapas del modelo de desarrollo en cascada.



La palabra *cascada* sugiere una demanda incremental, mediante la metáfora de la fuerza de la gravedad, es decir, hace alusión al esfuerzo, tiempo y costo requeridos para introducir un cambio en las fases más avanzadas de un proyecto.

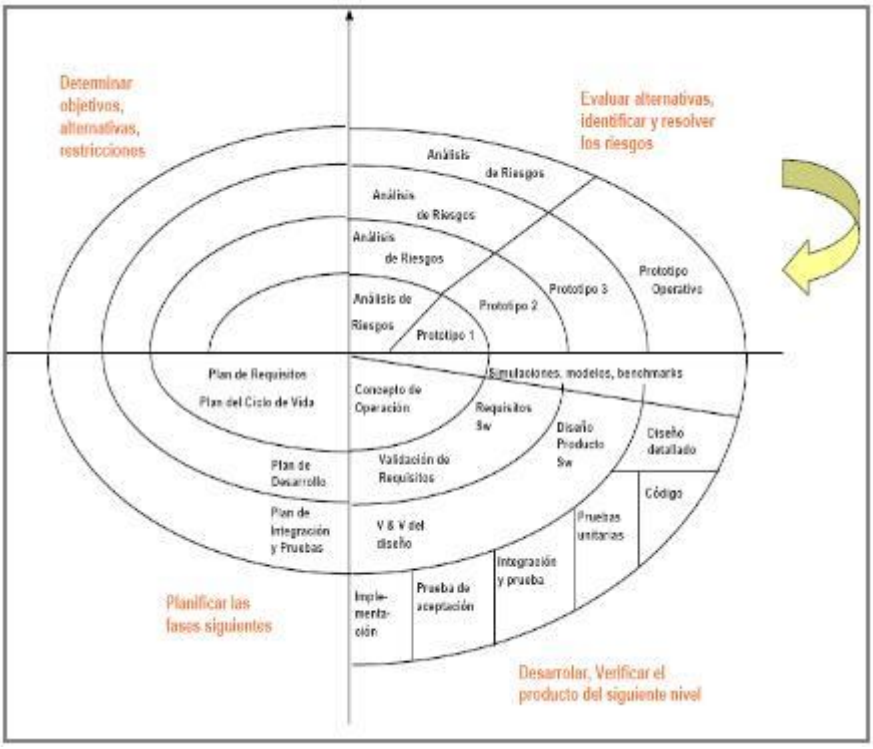
Si bien este método ha sido ampliamente criticado desde el ámbito académico y la industria, aún sigue siendo el paradigma más utilizado hasta el día de hoy.

- *Modelo en espiral*.

Consiste en una serie de ciclos que se repiten en forma de espiral. En un principio se suele decir que dentro de cada ciclo de la espiral se puede seguir un Modelo en Cascada, aunque no necesariamente tiene por qué ser así siempre.

En este enfoque se establecen una serie de fases que se denominan "Modelos de Ciclo de Vida", los cuales simulan los distintos estados de un producto de software desde su concepción inicial, pasando por su puesta en producción y posterior mantenimiento, hasta su retiro final de servicio activo.

En la siguiente figura se puede visualizar una representación generalizada y simplificada que muestra los ciclos continuos de fases o etapas del modelo de desarrollo en espiral.



Cada modelo de ciclo de vida en espiral tiene en cuenta el riesgo que aparece a la hora de desarrollar software. Para ello, se comienza considerando las posibles alternativas de

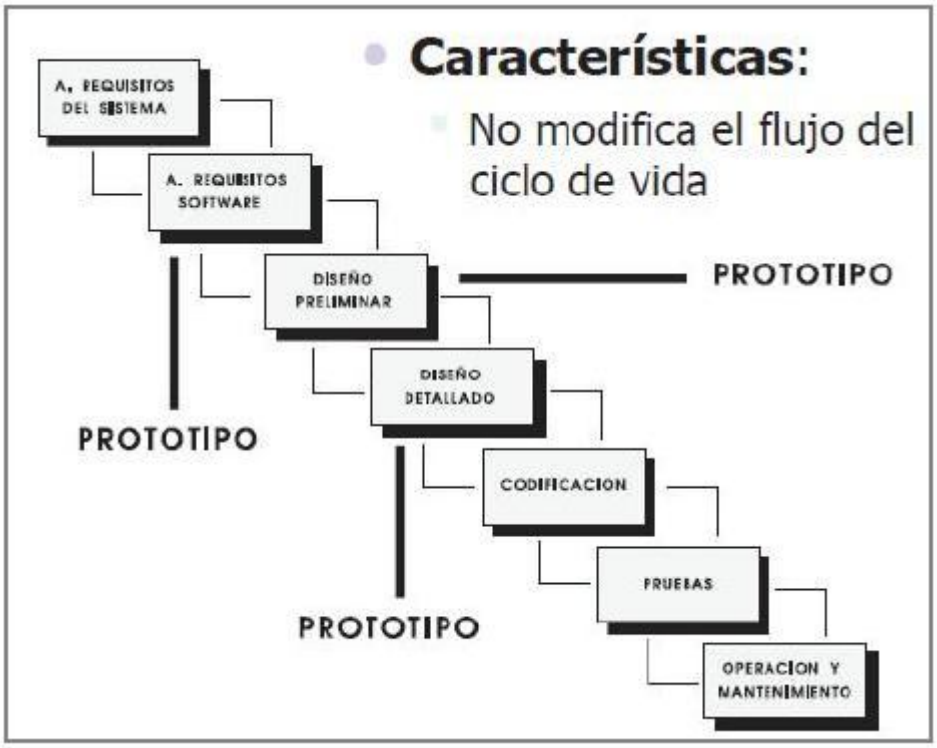
desarrollo, luego se opta por escoger la que presenta el riesgo más "aceptable", y con esta alternativa seleccionada se diseña un ciclo de la espiral.

Si el cliente o usuario receptor del software desea seguir adicionando mejoras en el producto, entonces se vuelve a evaluar las alternativas disponibles y se realiza nuevamente otra vuelta de la espiral. De esta manera se puede continuar hasta que llegue un momento en el cual el producto desarrollado sea aceptado y no merezca la pena seguir mejorándolo con otro nuevo ciclo.

- *Modelo de prototipos*.

También llamado Modelo de Desarrollo Evolutivo, se inicia con la definición de los objetivos globales que el software debe cumplir, luego se identifican los requisitos conocidos y las áreas del esquema en donde es necesaria más definición. Entonces se plantea con rapidez una iteración de construcción de prototipos y se presenta el modelado (en forma de un diseño rápido).

El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final (por ejemplo, la configuración de la interfaz con el usuario y el formato de los despliegues de datos).



En la figura anterior es posible visualizar una representación generalizada y simplificada que muestra en forma gráfica las fases o etapas del modelo de desarrollo por prototipos.

El diseño rápido conduce a la construcción de un prototipo, el cual es evaluado por el cliente o el usuario para brindar retroalimentación al desarrollador; gracias a ésta técnica se pueden refinar más fácilmente los requisitos del software que se ha de desarrollar.

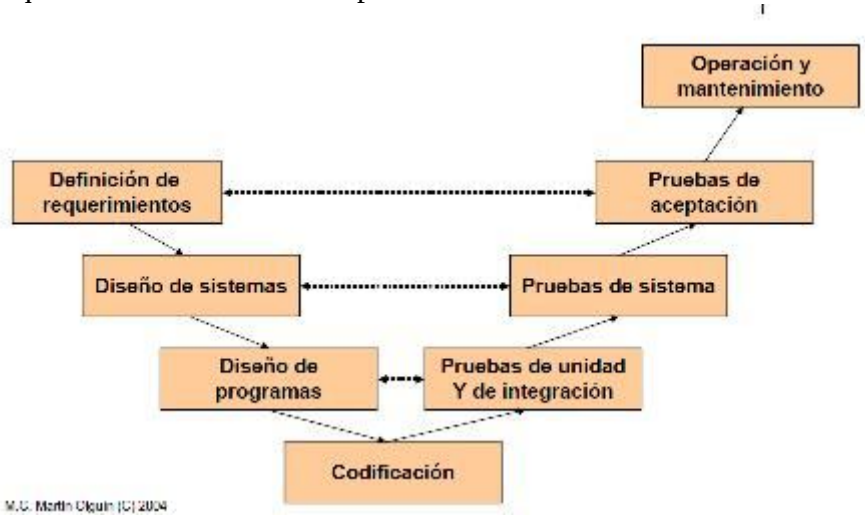
La iteración evolutiva de estos prototipos ocurre cuando alguno se ajusta para satisfacer las necesidades del cliente o el usuario receptor. Este proceso permite que el desarrollador entienda mejor la naturaleza de lo que se debe hacer y al mismo tiempo que el cliente pueda percibir resultados tangibles en plazos relativamente cortos.

- *Método en V.*

Este modelo describe las actividades y los resultados que se producen durante el desarrollo del software y fue concebido originalmente por Administración Federal Alemana como una herramienta para regular en forma sistemática los procesos de desarrollo de software.

Es una representación gráfica en forma de letra "**V**" del ciclo de vida del desarrollo del sistema, que describe y resume los pasos principales que se deben seguir en conjunción con las correspondientes entregas de los sistemas de validación.

En la siguiente figura se puede visualizar una representación generalizada y simplificada que muestra las fases o etapas del modelo de desarrollo en "**V**".



La parte izquierda de la "**V**" representa la corriente donde se definen las especificaciones del sistema. La parte derecha de la "**V**" representa la corriente donde se comprueba el sistema (con base en las especificaciones definidas en la parte izquierda). La parte de abajo, donde convergen ambas corrientes, representa la fase de desarrollo.

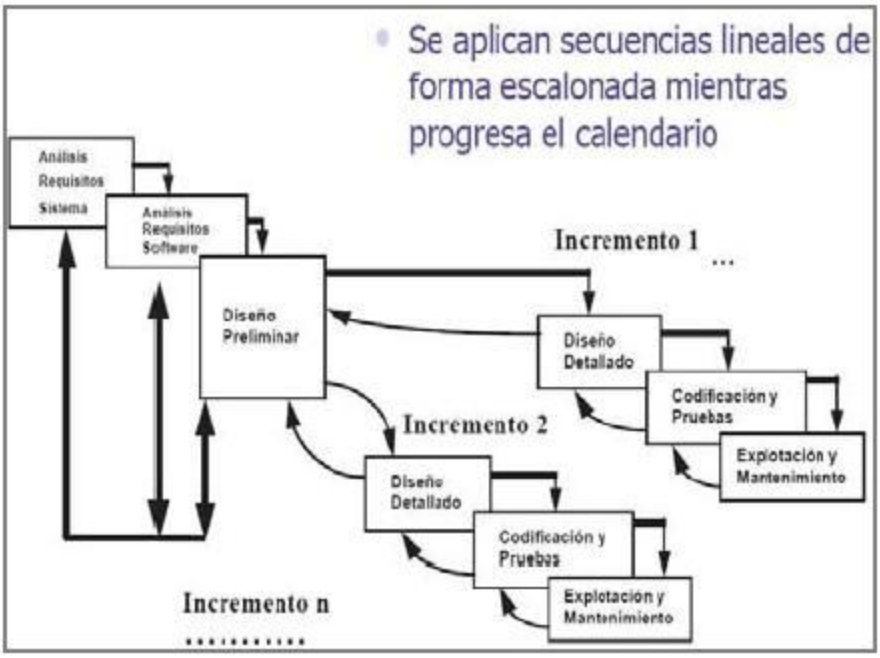
La *corriente de especificación* consiste principalmente de los pasos siguientes: *Especificaciones de requerimientos de usuario*, *Especificaciones funcionales*, y *Especificaciones de diseño.*

La *corriente de comprobación*, por su parte, suele consistir de los pasos siguientes: *Calificación de instalación*, *Calificación operacional*, y *Calificación de rendimiento*.

La *corriente de desarrollo* puede consistir (depende del tipo de software y del alcance del proyecto) en: *Personalización*, *Configuración* o *Codificación*.

- *Desarrollo incremental por etapas.*

Es similar al Modelo de Prototipos, ya que se presta para mostrar al cliente o al usuario receptor el software en diferentes estados sucesivos de desarrollo.



En la siguiente figura se puede visualizar una representación generalizada y simplificada de las fases o etapas del modelo de desarrollo iterativo o incremental.

Se diferencia principalmente en que las especificaciones no son conocidas en detalle al inicio del proyecto y por lo tanto se van desarrollando simultáneamente con las diferentes versiones de los objetos resultantes del proceso que conforman el producto de software.

En general se pueden distinguir las siguientes fases: *Especificación conceptual*, *Análisis de requerimientos*, *Diseño inicial*, *Diseño detallado*, *Codificación*, *Depuración*, y *Liberación*.

Estas diferentes fases del proceso se pueden repetir en cada etapa sucesiva del diseño.

Resumen

En términos muy generales es posible afirmar que la Ingeniería de Software designa un conjunto de técnicas destinadas a facilitar la construcción de productos de software, más allá de la actividad elemental de programación tradicional.

Suele formar parte de esta disciplina algunas otras actividades de las ciencias computacionales provenientes de una rama más genérica denominada Ingeniería Informática, como por ejemplo, la gestión de proyectos y el mantenimiento de sistemas informáticos.

Adicionalmente, algunos autores consideran que el término *Desarrollo de Software* es un término más apropiado que el término *Ingeniería de Software* para referirse al proceso de construcción formal de software. Por ejemplo, Pete McBreen (autor de la obra "Software Craftmanship") manifiesta que el término *Ingeniería de Software* implica elevados niveles de rigor y prueba de procesos que no son apropiados para todo tipo de desarrollo de software.

Capítulo 2
Ingeniería de Software: Desafíos

La construcción de productos de software que incorporen calidad en forma rentable, constituye un desafío en extremo complejo, principalmente debido a que involucra toda una gama de factores tan diversos como subjetivos, los cuales muchas veces se contraponen entre sí. Por ejemplo, para elevar los niveles de calidad es posible invertir en más esfuerzo orientado a las labores de revisión y pruebas, lo cual se puede traducir en un incremento directo de los costos involucrados.

A continuación se listan algunos desafíos, ciertamente interesantes:

- El software se desarrolla, no se fabrica.
- El software no se descompone, se echa a perder.
- Aunque la industria promueve el ensamblaje de componentes, la mayoría del software es hecho a la medida.

Es en este entorno de demandas tan heterogéneas donde se requiere profundizar en el estudio de necesidades, diseño formal de requerimientos, programación de especificaciones, y demás actividades involucradas en el proceso de construcción de software y su posterior liberación en producción.

La Ingeniería de Software ha surgido como una respuesta viable a la creciente demanda de software que cumpla con las exigencias de calidad funcional (confiable, eficiente y versátil) financieramente sustentable (dentro de los tiempos acordados y los presupuestos designados).

Una de las definiciones de la Ingeniería de Software nos permite vislumbrar que esta disciplina está siendo orientada conforme a principios y metodologías científicas.

“La Ingeniería del Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora, además de la documentación requerida para desarrollarlos, operarlos y mantenerlos. Este proceso también se conoce como desarrollo de software o producción de software. [Bohem, 1976]”.

Los ingenieros de sistemas no sólo están relacionados con el software, sino también con el hardware y las interacciones de los sistemas con los usuarios y su entorno. Estas interacciones propias del proceso de construcción aunado a las interacciones propias de los usuarios y su entorno, constituyen elementos e influencias difíciles de cuantificar y medir en forma objetiva.

Afortunadamente, otra de las definiciones de la Ingeniería de Software nos muestra que esta disciplina también está siendo orientada hacia la investigación y obtención de métricas objetivas.

"Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de la ingeniería al software. [IEEE, 1993]".

En resumen, es posible afirmar que quienes impulsan formalmente la Ingeniería de Software han sido capaces de reconocer los desafíos inherentes del proceso de construcción de software, y en consecuencia están orientando esta emergente disciplina en forma científica.

Capítulo 3
Ingeniería de Software: Avances

Uno de los aspectos primordiales que se puede considerar como un avance significativo de la Ingeniería de Software, es la clarificación de los atributos que debe cumplir un buen producto de software. A continuación se listan algunos de los conocidos:

- Mantenibilidad: El software debe poder evolucionar para cumplir con las necesidades de cambio de los clientes.
- Confiabilidad: El software debe ser fiable, seguro, no debe causar daños físicos o económicos en el caso de una falla del sistema.
- Eficiencia: El software debe aprovechar al máximo los recursos del sistema.
- Usabilidad: El software debe ser fácil de utilizar.

Otro avance relevante consiste en la conceptualización de diferentes metodologías orientadas hacia la formalización de las actividades inherentes al desarrollo de software. Algunos de las metodologías o modelos más divulgados son los siguientes:

- Modelo en Cascada (ciclo de vida clásico)
- Modelo en "V"
- Modelo de Construcción de Prototipos
- Modelos Evolutivos (Incremental, Espiral, Espiral WIN-WIN)

La pregunta que suele surgir es ¿Qué factores se deben considerar cuando se estructura un proyecto de construcción de software?. A continuación se mencionan algunos de los factores más ampliamente reconocidos:

- Complejidad del proyecto (dificultad del problema, tamaño del software)
- Tiempo de desarrollo.
- Modularidad.
- Calidad.
- Comunicación requerida.

Es a partir de las complejas relaciones que existen entre estos factores que surge la necesidad de medir y anticipar su impacto en los procesos de desarrollo, y por ende, en los productos de software finales.

Es en el campo de las mediciones metódicas en donde la Ingeniería de Software también ha alcanzado algunos avances importantes. En general estas métricas se pueden clasificar en dos grandes grupos, a saber, medidas relacionadas al tamaño y medidas relacionadas a la función.

Las primeras se relacionan con el tamaño de la salida derivada de alguna actividad. En esta categoría, la métrica más comúnmente conocida es la denominada "Líneas de Código (LCD)". Este tipo de métricas dependen directamente del lenguaje de programación utilizado y en general no son apropiadas como herramientas para medir la "Programación Orientada a Objetos (POO)".

Las segundas se relacionan con la funcionalidad del software. En esta clasificación las mediciones más comunes son las denominadas "Puntos de Función (PF)" y "Puntos de Objeto (PO)".

Otros aspectos conectados con las complejas relaciones entre los factores que intervienen en la construcción de software, son los vinculados con la apropiada administración de los riesgos derivados de estas relaciones.

Como se va a describir más adelante, es en el campo de la administración metódica del riesgo en donde la Ingeniería de Software también ha alcanzado algunos avances importantes.

Algunos de los riesgos potenciales mejor reconocidos por la Ingeniería de Software y el nivel donde se identifica su impacto, son los siguientes:

- Rotación de personal — Proyecto
- Cambio de administración — Proyecto
- No disponibilidad del hardware — Proyecto
- Cambio de requerimientos — Proyecto y producto
- Retrasos en la especificación — Proyecto y producto
- Subestimación del tamaño — Proyecto y producto
- Bajo desempeño de la herramienta CASE — Producto
- Cambio de tecnología — Negocio

Algunas de las etapas consideradas por la Ingeniería de Software en la administración de riesgos potenciales, se listan a continuación:

- Identificación de riesgos: Listado de riesgos potenciales
- Análisis de riesgos: Listado de priorización de riesgos
- Valoración de las probabilidades y consecuencias

- Planeación de riesgos: Anulación de riesgos y elaboración de planes de contingencia
- Planes para evitar o minimizar el impacto
- Supervisión de riesgos: Valoración del impacto potencial
- Valoración constante: Revisión continuada de los planes de mitigación conforme se vaya presentando la información actualizada del riesgo.

Finalmente, la Ingeniería de Software también ha demostrado algunos avances importantes en el área de la estimación de los costos involucrados en las diferentes etapas del proceso de desarrollo y posterior liberación de los productos finales. A continuación se listan algunas de las técnicas más utilizadas en las estimaciones de costos:

- Modelado del algoritmo de costos: Utiliza un modelo con información histórica de costos, relaciona una métrica con el costo del proyecto. Se estima la métrica y se predice el esfuerzo.
- Opinión de expertos: Se consultan expertos en las técnicas de desarrollo propuestas y el dominio de la aplicación. Cada uno estima el costo y se consensa después de varias iteraciones.
- Estimación por analogía: Cuando se han completado proyectos del mismo dominio de la aplicación, entonces es posible estimar con base en la experiencia.
- Ley de Parkinson: Establece que el trabajo se expande para llenar el tiempo disponible. El costo se determina más por los recursos disponibles que por los objetivos logrados.

A pesar de los significativos avances evidenciados por la Ingeniería de Software hasta la fecha, principalmente aquellos fundamentados en la aplicación de técnicas científicas, aún no se constituye en una “ciencia” razonablemente exacta y determinante. Sin embargo, es prudente reconocer que si ha llegado a constituirse en el mejor conjunto de herramientas disponibles para la construcción formal productos de software.

Segunda Sección
Ingeniería de Software: "Desarrollo en Cascada"

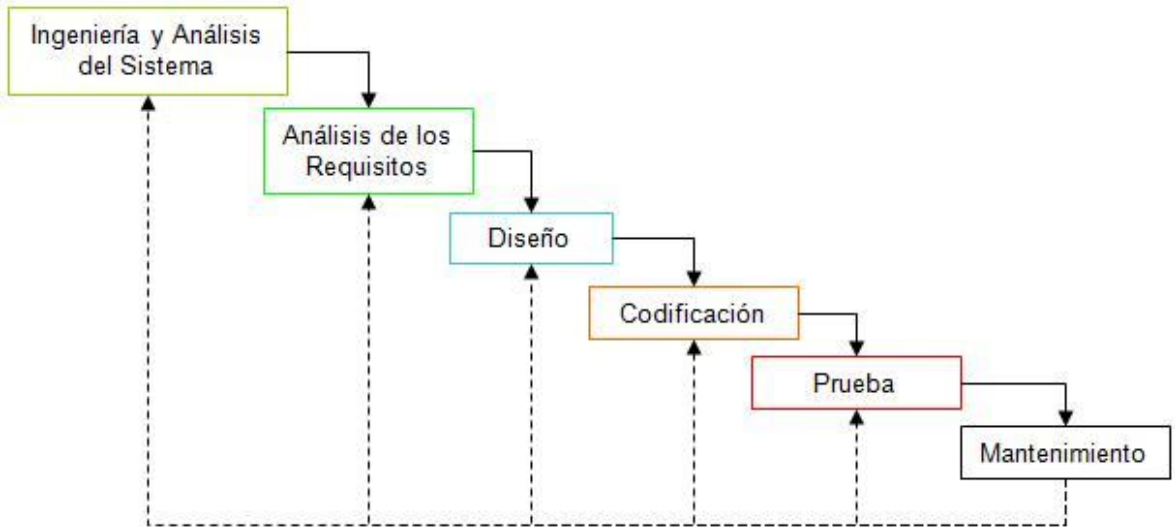
Capítulos.

1. Descripción
2. Ventajas
3. Desventajas

Capítulo 1
Desarrollo en Cascada: Descripción

En Ingeniería de Software el desarrollo en cascada, también llamado modelo en cascada, es el enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma tal que el inicio de cada etapa debe esperar a la finalización de la etapa inmediatamente anterior.

En la siguiente figura se puede visualizar una representación generalizada y simplificada que muestra las fases o etapas del modelo de desarrollo en cascada.



Sus principales características son las siguientes:

- Cada fase empieza cuando se ha terminado la fase anterior.
- Para pasar de una fase a otra es necesario conseguir todos los objetivos de la etapa previa.
- Al final de cada fase el personal técnico y los usuarios tienen la oportunidad de revisar el progreso del proyecto.

De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando proporcionalmente los costos del desarrollo. La palabra cascada sugiere, mediante la metáfora de la fuerza de la gravedad, los tiempos y esfuerzos incrementales necesarios para introducir un cambio en las fases más avanzadas de un proyecto.

Fases o Etapas

Este modelo es conocido también como ciclo de vida lineal o básica. Este modelo admite la posibilidad de hacer iteraciones. Se define como una secuencia de fases, tal como fue mostrado en la figura anterior, en la que al final de cada una de estas fases se reúne la documentación necesaria, con la finalidad de garantizar que se cumplen las especificaciones y los requisitos correspondientes antes de pasar a la fase siguiente.

Las fases o etapas características de la metodología de desarrollo en cascada se describen a continuación en forma general:

- *Análisis del sistema*.
  Se descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo. Como resultado surge el Documento de Diseño del Software (SDD), que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.

- *Análisis de requisitos*.
  Se analizan las necesidades de los usuarios finales del software para determinar qué objetivos se deben cubrir. De esta fase surge una memoria llamada Documento de Especificación de Requisitos (SRD), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.

  Es importante señalar que en esta etapa se debe consensuar todo lo que se requiere del sistema y que ha de permanecer y guiar el proyecto en las etapas siguientes, no pudiéndose incorporar nuevos requerimientos a medida que avanza el proceso de elaboración del software.

- *Diseño del sistema*.
  Esta es la fase en donde se realizan los algoritmos y estructuras de datos necesarias para lograr el cumplimiento de los requerimientos del usuario, así como también los análisis necesarios para saber qué herramientas usar en la etapa de Codificación.

- *Codificación*.
  Es la fase de programación o implementación propiamente dicha. Aquí se implementa el código fuente, haciendo uso tanto de prototipos, así como pruebas y ensayos para corregir errores. Dependiendo del lenguaje de programación y su respectiva versión, se construyen las librerías y componentes reutilizables necesarios, dentro del mismo proyecto para hacer que la programación sea un proceso más rápido y confiable.

- *Pruebas*.
  Los elementos o componentes una vez que han sido programados, se ensamblan para conforma el sistema, y mediante pruebas de integración se procede a comprobar que la totalidad del sistema funciona correctamente antes de ser liberado en producción.

- *Producción*.
  Una vez completado en proceso de desarrollo, el software obtenido se pone en producción. En esta etapa se implantan los niveles de hardware y software que complementan el proyecto. Normalmente, la implantación es la fase de mayor duración y con más cambios en el ciclo de elaboración de un proyecto. Es una de las fases finales del proyecto.

Una vez que el sistema ha sido liberado en producción, suelen surgir cambios, bien para corregir errores o bien para introducir mejoras. Estos cambios son recogidos mediante los Documentos de Cambios.

Variantes

Existen algunas variantes de este modelo. Es especialmente destacada la que hace uso de prototipos, en la cual se establece un ciclo antes de llegar a la fase de mantenimiento, verificando que el sistema final esté libre de fallos.

Como resultado de un breve análisis sobre el modelo en cascada, se puede afirmar que éste es realmente básico y que su "ventaja" más relevante, si así se lo puede llamar, es su simplicidad, debido a que la secuencia de pasos es bastante intuitiva y natural, los cuales, sin embargo, no dejan de ser necesarios a la hora de desarrollar los productos de software, y además resultan ser de gran utilidad para explicar el ciclo de desarrollo al personal no técnico, tales como los clientes y los usuarios finales.

Sin embargo para emprender proyectos de construcción de software a gran escala, algunos autores opinan que este tipo de modelo no es recomendable, pues debido a su rigidez para incorporar cambios a las especificaciones iniciales, puede causar muchos problemas durante y después del proceso mismo de desarrollo, no solamente a los desarrolladores del mismo sino también a las personas que esperan recibir el producto para utilizarlo, es decir, a los clientes y/o usuarios finales.

Capítulo 2
Desarrollo en Cascada: Ventajas

Mediante este capítulo se describe en forma general algunas de las ventajas más sobresalientes de la metodología de desarrollo de software denominado Modelo en Cascada.

En términos muy generales esta metodología permite a los desarrolladores asegurar los beneficios siguientes:

- Este modelo es sencillo ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
- Se tiene todo bien organizado y no se mezclan las actividades de cada fase.
- Ayuda a prevenir que no se sobrepasen las fechas de entrega y los costos esperados.
- Bajo riesgo para desarrollos bien comprendidos utilizando una tecnología bien conocida.
- Es perfecto para proyectos que son rígidos, y además donde se especifiquen muy bien los requerimientos y se conozca muy bien la herramienta a utilizar.

Capítulo 3
Desarrollo en Cascada: Desventajas

En este capítulo se explica en forma general algunas de las desventajas más evidentes del Modelo de Desarrollo en Cascada.

- Su inflexibilidad en la división del proyecto en distintas etapas.
- Esto hace difícil poder responder a la demanda de cambios en los requerimientos del cliente.
- Se tarda mucho tiempo en pasar por todo el ciclo.
- El mantenimiento se realiza directamente en el código fuente.
- Las revisiones de proyectos de gran complejidad son muy difíciles.
- Para obtener resultados tangibles se debe llegar a la etapa final del proyecto. Un error importante detectado hasta el momento en que el sistema esté funcionando puede resultar desastroso.

En la vida real, un proyecto de desarrollo rara vez sigue una secuencia lineal, esta circunstancia puede derivar en una mala implementación del modelo, lo cual a su vez, puede conducir el proyecto hacia el fracaso.

Difícilmente un cliente se encuentra en la capacidad de establecer desde el principio todos los detalles de los requerimientos necesarios, por lo cual al trabajar con este modelo, se suele invertir un extenuante esfuerzo en la primera fase del proyecto, con la finalidad de reducir al máximo la incidencia de cambios inesperados en las fases subsiguientes del proyecto.

Los resultados y/o mejoras de las fases intermedias no son visibles para los clientes o los usuarios finales, sino que el producto solamente es visible y tangible una vez que está completamente terminado. Esta situación provoca una gran inseguridad por parte de los clientes, quienes se ven obligados a esperar ansiosos los avances en el proceso de desarrollo del producto. En este contexto también resulta bastante común tener que lidiar con requerimientos que no se habían tomado en cuenta, y que surgieron al momento de la implementación, lo cual suele provocar que se regrese nuevamente a la fase de requerimientos.

Tercera Sección
Ingeniería de Software: "Desarrollo en Espiral"

Capítulos

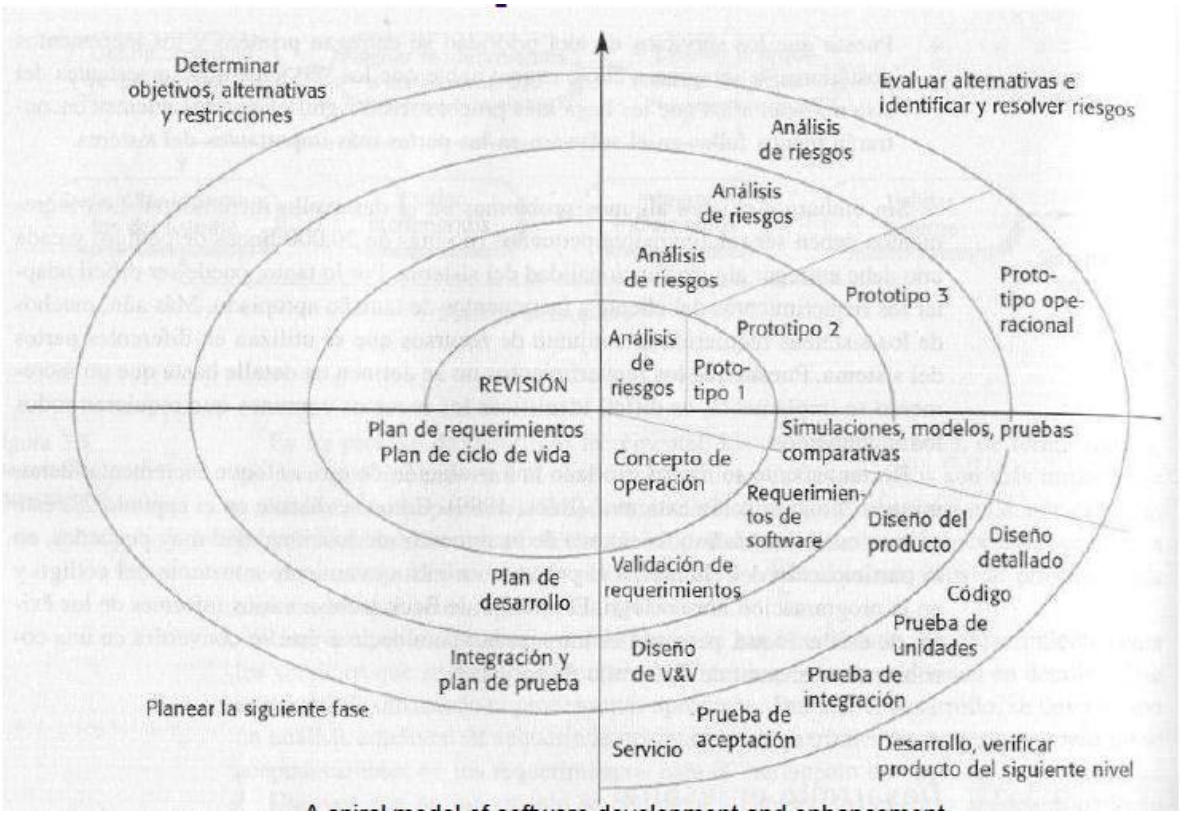
1. Descripción
2. Ventajas
3. Desventajas

Capítulo 1
Desarrollo en Espiral: Descripción

El Modelo en Espiral es un modelo que organiza en forma evolutiva las actividades orientadas al desarrollo de software. Combina la naturaleza iterativa de la construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial.

Durante la primera vuelta alrededor de la espiral se definen los objetivos, las alternativas y las restricciones, además se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay alguna incertidumbre en los requisitos, se puede usar la construcción de prototipos en el cuadrante de ingeniería para dar asistencia tanto al encargado del desarrollo como al cliente.

En la siguiente figura se puede visualizar una representación generalizada y simplificada que muestra en forma gráfica los ciclos continuos de fases o etapas del modelo de desarrollo en espiral, los cuales se organizan en cuatro cuadrantes específicos.

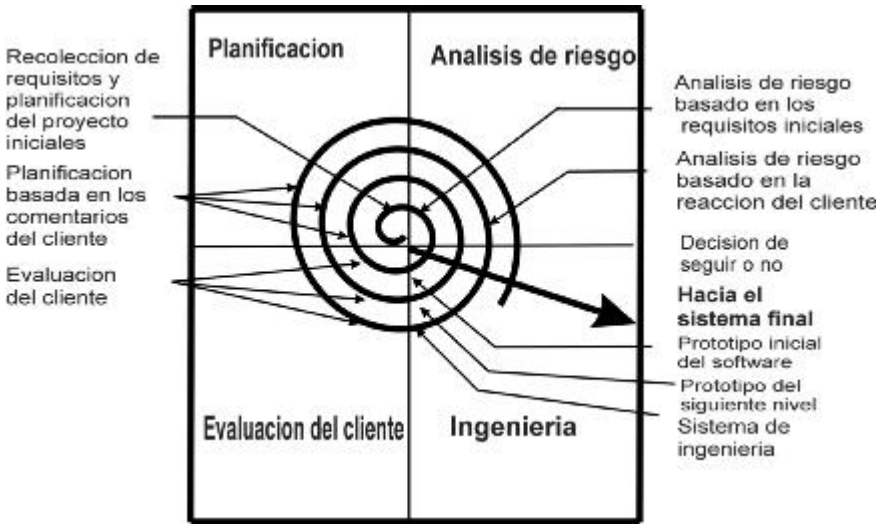


Según algunos autores, las actividades de este modelo se pueden organizar en forma de una espiral de múltiples niveles, en donde cada ciclo o iteración representa un conjunto definido de actividades. Normalmente, las actividades de un determinado nivel se eligen en función del análisis de riesgo resultante del nivel anterior, comenzando por el ciclo más interno. De esta forma el software se va desarrollando en una serie de versiones incrementales.

Características

El modelo espiral ha sido concebido para potenciar las mejores características tanto del ciclo de vida clásico, como de la construcción de prototipos, añadiendo al mismo tiempo un elemento adicional: el análisis de riesgo. Tal como se puede observar en los cuadrantes de la siguiente figura, el modelo representado mediante la espiral, define cuatro actividades principales:

- Planificación: Determinación de objetivos, alternativas y restricciones.
- Análisis de riesgo: Análisis de alternativas e identificación/resolución de riesgos.
- Ingeniería: Desarrollo del producto del "siguiente nivel".
- Evaluación del cliente: Valorización de los resultados de la ingeniería.



Normalmente, el cliente evalúa el trabajo de ingeniería (cuadrante de evaluación de cliente) y sugiere modificaciones. Sobre la base de los comentarios del cliente se produce la siguiente fase de planificación y de análisis de riesgo. En cada ciclo alrededor de la espiral, la culminación del análisis de riesgo resulta en una decisión de "seguir o no seguir".

Con cada iteración alrededor de la espiral (comenzando en el centro y siguiendo hacia el exterior), se construyen sucesivas versiones del software, en donde se espera que cada

versión sea más completa que su predecesora, lo cual resulta al final en un producto totalmente operacional.

Algunas de las principales actividades de este modelo se mencionan a continuación:

- Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo.
- Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.
- Cada ciclo empieza identificando: Los objetivos de la porción correspondiente, las alternativas y las restricciones.
- Se evalúan las alternativas respecto a los objetivos y las restricciones.
- Se formula una estrategia efectiva para resolver las fuentes de riesgos (simulación, prototipado, etc.).
- Se plantea el alcance del próximo prototipo.
- Una vez resueltos los riesgos, entonces, se sigue el ciclo en cascada.
- Cada ciclo se completa con una revisión que incluye todo el ciclo anterior y el plan requerido para el siguiente.

Como ya ha sido mencionado brevemente, un aspecto muy sobresaliente de este modelo es que se preocupa de los riesgos que suelen presentarse en todo proyecto de desarrollo, lo cual permite una administración más razonable de los compromisos a adquirir con los clientes y usuarios finales.

Reseña histórica

Considerando las características del Modelo Espiral, es posible afirmar que es más avanzado que otros modelos disponibles. La versión revisada de este modelo fue propuesta por Barry Boehm en el año de 1988 y es un método que como su nombre lo describe, consiste en una espiral mediante el cual es posible adicionar y trabajar en forma dinámica los procesos respectivos para el desarrollo del software.

Este método se fundamenta en una serie de ciclos repetitivos, los cuales son administrados mediante determinados principios básicos de desarrollo, con la finalidad de producir productos de software de gran calidad, incluso en proyectos de gran magnitud.

El paradigma del modelo en espiral constituye actualmente el enfoque más realista para el desarrollo de software y de sistemas a gran escala. Utiliza un enfoque evolutivo para la ingeniería de software, permitiendo al desarrollador y al cliente entender y consensuar los riegos con la finalidad de reaccionar en forma consistente en cada nivel evolutivo.

Adicionalmente, promueve la construcción de prototipos en forma incremental como un mecanismo de reducción de riesgos, permitiendo a los desarrolladores la flexibilidad de aplicar este enfoque de generación de prototipos en cualquier fase del proceso de desarrollo.

Algunos de los modelos más utilizados en forma complementaria para construir versiones incrementales en cada ciclo se mencionan a continuación:

Modelo espiral con cascada

Esta estrategia permite aprovechar el carácter formal de las etapas propias del modelo de desarrollo en cascada, al momento de estar trabajando en un determinado ciclo del modelo de desarrollo en espiral.

Este enfoque es particularmente útil cuando se está desarrollando un producto de software de gran envergadura, principalmente debido a permite controlar de manera eficiente los aspectos relacionados con la calidad del producto resultante.

Por otra parte, algunos autores afirman que este enfoque combinado no es recomendable para desarrollar proyectos pequeños, principalmente debido a que se suele adicionar en forma innecesaria la rigidez inherente de este método de desarrollo, sino que en su lugar, en estos casos resulta más provechoso utilizar el modelo en cascada en su forma nativa.

Modelo espiral con prototipos

Esta estrategia permite aprovechar la versatilidad propia del modelo de desarrollo en prototipos, cuando se está trabajando en un determinado nivel o ciclo del modelo de desarrollo en espiral.

La generación de prototipos es particularmente útil para apoyar la construcción de software de gran magnitud, principalmente debido a que permite la posibilidad de mostrar a los interesados, mediante versiones intermedias, los avances alcanzados en el ciclo de desarrollo correspondiente.

Por otra parte, algunos detractores señalan que la utilización de este enfoque incremental, permite la introducción de algunas debilidades, para las cuales también se han propuesto alternativas, sin embargo, la experiencia ha demostrado que estas soluciones propuestas no siempre funcionan. Por ejemplo, la potencial extensión no controlada de los tiempos y el incremento de los costos conforme se adicionan nuevas funcionalidades al proceso de desarrollo.

Adicionalmente, algunos críticos enuncian que cuando este método incremental es utilizado para desarrollar proyectos pequeños, también es posible obviar algunos de los pasos que no son estrictamente necesarios, con la finalidad de agilizar el proceso de desarrollo en general. En el caso particular, es posible señalar la construcción de múltiples prototipos, pues resulta bastante evidente que si el software es pequeño, entonces, es muy posible que el propio prototipo inicial tenga el mismo alcance que el producto final.

Modelo espiral WIN-WIN

Esta estrategia permite introducir los llamados "hitos de fijación o anclaje", los cuales también proporcionan los denominados "hitos de decisión", al momento de estar trabajando en un determinado ciclo del modelo de desarrollo en espiral, de tal manera que los clientes o usuarios finales puedan sentirse satisfechos con los resultados logrados en dicho ciclo, antes de continuar con el proyecto de desarrollo.

Algunos autores también afirman que este enfoque permite generar un clima de confianza en la relación que se da entre los desarrolladores y los clientes, el cual es significativamente necesaria para el éxito de cualquier proyecto de desarrollo de software.

Capítulo 2
Desarrollo en Espiral: Ventajas

Mediante este capítulo se describe en forma general algunas de las ventajas más sobresalientes del método de desarrollo en espiral.

- No se requiere contar con una definición completa de los requisitos para empezar las actividades de desarrollo.

- Al entregar resultados parciales a partir de la conclusión del primer ciclo del desarrollo se hace más fácil validar los requisitos acordados.

- Permite reducir el riesgo general del proyecto, debido a que si algo no resulta según lo previsto, únicamente de ve afectado el tiempo y demás recursos invertidos en el ciclo de desarrollo correspondiente.

- Permite implementar mejores controles orientados a reducir el riesgo de sufrir retrasos, principalmente debido a que al identificar los problemas en etapas tempranas del proceso de desarrollo se cuenta con más tiempo para subsanarlos.

- El análisis del riesgo se hace de manera clara y explícita. Este énfasis permite considerar los mejores elementos de los modelos incrementales utilizados.

- Permite incorporar y supervisar objetivos de calidad.

- Integra las actividades de desarrollo con las actividades de puesta en producción y también con las correspondientes al mantenimiento posterior.

Capítulo 3
Desarrollo en Espiral: Desventajas

En este apartado se explica en forma general algunas de las desventajas más evidentes del modelo de desarrollo en espiral.

- *Tiempos prolongados.*
  La construcción de múltiples versiones mejoradas en forma incremental en cada ciclo del proceso del proceso de desarrollo, demanda la inversión de tiempos considerables en cada uno de dichos ciclos. Si no se administran correctamente los controles necesarios, esta situación suele presentarse en forma de retrasos en la entrega del producto final.

- *Costos onerosos.*
  Debido a la magnitud de los esfuerzos que son demandados durante la utilización de esta metodología de desarrollo, se hace necesario mantener un estricto control sobre las inversiones económicas correspondientes, lo cual requiere de administradores hábiles y experimentados.

- *Experiencia requerida.*
  Como es comprensible, en proyectos de gran magnitud, resulta muy evidente que la administración de las diferentes actividades involucradas en cada ciclo de desarrollo, suele demandar de líderes muy experimentados en la apropiada administración de los riesgos e imprevistos inherentes a estos proyectos, con la finalidad de mantener los costos dentro de los presupuestos designados y también cumplir con los tiempos acordados con quienes esperan recibir el producto.

Conclusiones

Acerca de los temas relacionados con el análisis de las diferentes metodologías orientadas al desarrollo de productos de software, sería posible incluso escribir un libro completo, sin embargo, para efectos prácticos de esta asignación académica, es más provechoso comentar en forma resumida y general algunas de las fortalezas y debilidades más significativas de estas metodologías utilizadas en la construcción de aplicaciones organizacionales.

- Dentro del proceso de Ingeniería del Software los tres factores más importantes son: Personal, Métodos y Procedimientos, además de Herramientas y Técnicas.

- Existen modelos y procesos aplicados en las diferentes etapas del desarrollo de software.

- La disciplina de ingeniería del software está basada en procesos y modelos que permiten la definición, evaluación y medición del software.

- La facilidad de entendimiento humano y la comunicación son fundamentales para lograr una buena definición y ejecución de los procesos de construcción de software.

- La apropiada medición de los procesos inherentes de un proyecto de desarrollo de software es primordial, pues gracias a este principio es posible identificar las fuerzas y las debilidades de los mismos y a su vez del proyecto en su totalidad.

- La recolección de métricas objetivas de los procesos es esencial para introducir mejoras en los mismos, y además se pueden utilizar para evaluar la eficacia de los cambios realizados.

- Un producto de software constituye: los ejecutables, el código fuente y las descripciones de arquitectura, de tal manera que:

- Medir un producto de software implica: la medición del tamaño del producto, la estructura del producto y la calidad del producto.

- Dos de los modelos estandarizados que son utilizados más frecuentemente para la medición de la calidad del software son: CMMI e ISO 9000.

Comentarios

La Ingeniería del Software trata de áreas muy diversas de las ciencias computacionales y de la informática en particular, tales como compiladores, sistemas operativos o desarrollos de aplicaciones sobre Internet. Es una disciplina o área de la información o ciencias de la computación que ofrece métodos o técnicas para desarrollar y mantener software de calidad que permiten resolver problemas de todo tipo.

El término *ingeniería del software* empezó a usarse a finales de la década de los sesenta, para expresar la novedosa área de conocimiento que se estaba desarrollando en torno a las problemáticas que ofrecía el software en ese momento.

En esa época, el crecimiento espectacular de la demanda de sistemas de computación cada vez más y más complejos, asociado a la inmadurez propia del incipiente sector informático (totalmente ligado al electrónico) y a la falta de métodos y recursos, provocó lo que se llamó *la crisis del software* (en palabras de Edsger Dijkstra) entre los años 1965 y 1985.

Durante esa época muchos proyectos importantes superaban con creces los presupuestos y fechas estimados, algunos de ellos eran tan críticos (sistemas de control de aeropuertos, equipos para medicina, entre otros) que sus implicaciones iban más allá de las pérdidas millonarias que causaban.

La crisis del software disminuyó, no tanto por la mejora en la gestión de los proyectos, sino en parte porque no resulta razonable estar en crisis más de veinte años, y en parte porque se estaban haciendo esfuerzos importantes en los procesos de diseño y metodologías de desarrollo.

Así pues, desde 1985 hasta el presente, han ido apareciendo herramientas, metodologías y tecnologías que se presentaban como la solución definitiva al problema de la planificación, previsión de costos y aseguramiento de la calidad en el desarrollo de software.

Entre estas propuestas se encuentran la programación estructurada, la programación orientada a objetos y a los aspectos, las herramientas CASE, el lenguaje de programación ADA, los métodos de documentación, los estándares CORBA, los servicios web y el lenguaje UML (entre otros), las cuales en su debido momento fueron anunciadas como la solución a los problemas de la ingeniería del software, la llamada “bala de plata” (por el término en inglés *silver bullet*), sin embargo estos problemas aún persisten, por lo cual, cada año surgen nuevas ideas e iniciativas encaminadas en esta dirección.

Los mitos del software son creencias relacionadas a los productos de software y a los procesos empleados para construirlos, los cuales es posible rastrearlos desde los primeros días de la computación como disciplina. Para efectos prácticos, a continuación se mencionan las tres categorías más significativas de estos mitos.

Mitos de la administración

Los gestores con responsabilidad sobre el software, al igual que otros gestores en la mayoría de las disciplinas, normalmente están bajo la presión de cumplir las propuestas, hacer que no se retrasen los proyectos y a la vez mejorar la calidad. Un gestor de software se aferra frecuentemente a un mito del software.

Mito: Si se falla en la planificación siempre es posible adicionar más técnicos (analistas, programadores, revisores, etc.) para recuperar los tiempos de retraso.

Mitos del cliente

En muchos casos, el cliente cree en los mitos que existen sobre el software, debido a que los gestores y desarrolladores de software hacen muy poco para corregir la información errónea. Los mitos conducen a que el cliente se ilusione con una falsa expectativa y finalmente se expone a quedar insatisfecho con el desarrollador del software.

Mito: Si los requisitos del proyecto cambian continuamente, los cambios se pueden acomodar fácilmente, ya que el software es flexible por naturaleza.

Mitos de los desarrolladores

Los mitos en los que aún creen muchos desarrolladores se han ido fomentando a lo largo de 50 años de incultura informática. Durante los primeros días del desarrollo del software la programación se veía como un arte. Como es comprensible, las viejas prácticas y actitudes tardan en morir.

Mito: Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.

La ingeniería de software es una tecnología de múltiples capas, cualquier enfoque de ingeniería debe apoyarse sobre un compromiso de organización de calidad.

Capas de la Ingeniería de Software

El fundamento de la ingeniería de software corresponde a la capa denominada “procesos”. Los procesos de la ingeniería de software constituyen la unión que mantiene juntas las capas de tecnología con la finalidad de fomentar un desarrollo racional y oportuno de la ingeniería de software. Además, la capa de procesos define un marco de trabajo que sirve de referencia a un amplio conjunto de otras áreas clave de la ingeniería de software, tales como los métodos.

Los métodos de la ingeniería de software indican cómo se debe construir técnicamente el software. Los métodos comprenden una gran gama de tareas que incluyen análisis de requisitos, diseño lógico y físico, construcción de programas, pruebas integrales, puesta en producción y mantenimiento.

Finalmente, las herramientas de la ingeniería de software tienen como finalidad proporcionar un enfoque de automatización aplicable a los procesos y los métodos.

Referencias

- http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html
- http://sunset.usc.edu/cse
- http://www.monografias.com/trabajos5/inso/inso.shtml
- http://www.um.es/giisw/isbc
- http://www.sistemas.unam.mx/software.html

- http://es.wikipedia.org

Bibliografía

- Introducción a la ingeniería de software, Manuel Imaz.
- Ingeniería del Software: Un Enfoque Práctico, Roger Presuman.