Primera Parte

Introducción

Capítulo I

La Ingeniería de Sistemas

La ingeniería de sistemas

Tabla de contenido

1.- Ingeniería de sistemas e ingeniería de software 5
2.- Evolución de la ingeniería de sistemas.....5
3.- Tecnología de información.....9
4.- El ciclo de vida del software 10
5.- Enfoques alternativos del ciclo de vida.....11
5.1.- Modelo en cascada.....11
5.2.- El modelo incremental 12
5.3.- El modelo de desarrollo evolutivo 13
5.4.- El modelo en espiral.....15
5.5.- Combinaciones.....15
6.- Importancia de las metodologías.....15
6.1.- Problemas de comunicación.....16
6.2.- Confusión entre el qué y el cómo.....16
6.3.- Manejo de la complejidad de los sistemas 17
6.4.- Participación del usuario.....17
7.- Algunas metodologías.....18
7.1.- La metodología estructurada.....18
7.2.- La Ingeniería de Información.....19
7.3.- Ingeniería de Redesarrollo 19
7.4.- El marco de soluciones de Microsoft - MSF.....21
7.5.- El Rational Unified Process - RUP 23

La ingeniería de sistemas

1.- Ingeniería de sistemas e ingeniería de software

Algunos autores definen la ingeniería como “el arte de aplicar los conocimientos científicos a la invención, perfeccionamiento o utilización de la técnica industrial en todas sus determinaciones”. Otra definición muy aceptada es la siguiente: “Ingeniería es una disciplina cuyo objetivo es crear productos confiables a bajo costo”.

En particular, la ingeniería de sistemas se ocupa de los sistemas y puede definirse como “la aplicación de conocimientos y principios de ingeniería a actividades de diseño y construcción de sistemas”.

La ingeniería de software forma parte de la ingeniería de sistemas y comprende el conjunto de principios y conocimientos de ingeniería aplicados a la especificación, diseño, construcción, prueba y mantenimiento de los componentes de software de un sistema. Esto quiere decir que la ingeniería de software centra su interés en el desarrollo de componentes eficientes, es decir, capaces de utilizar en la mejor forma posible los recursos de computación y comunicación disponibles.

2.- Evolución de la ingeniería de sistemas

La ingeniería de software y la de sistemas nacieron como disciplinas hacia finales de los años 60, cuando se acuñó el término software engineering para referirse a los procedimientos de diseño y construcción de programas. Desde entonces, las técnicas para desarrollar programas y sistemas han venido evolucionando, guiadas por diferentes propuestas metodológicas: estructuradas, de orientación a objetos, etc.

Uno de los aportes cruciales en los inicios de la ingeniería de sistemas lo constituye el trabajo de Edsger Wybe Dijkstra presentado en su célebre artículo *Go To Statement Considered Harmful* (Las sentencias Go To son consideradas dañinas*)*, publicado en la revista Communications of the ACM, Vol. 11 (1968). En dicho artículo, Dijkstra demostraba que

todo programa puede escribirse sin utilizar la instrucción Go To y utilizando únicamente las tres instrucciones de control siguientes:

- El bloque de instrucciones consecutivas, que se ejecutan una después de otra.
- La instrucción condicional

  IF condición
  THEN instrucción-1
  ELSE instrucción-2

  conocida normalmente como estructura IF-THEN-ELSE.

  Si la condición se cumple, se ejecutará "instrucción-1", en caso contrario, se ejecuta "instrucción-2".
- El lazo o loop condicional "WHILE condición DO instrucción", que ejecuta la instrucción repetidamente mientras la condición se cumpla.

  En su lugar, se puede utilizar también la forma "UNTIL condición DO instrucción", que ejecuta la instrucción hasta que la condición se cumpla. Los dos loops se diferencian entre sí porque en la forma WHILE la condición se comprueba al principio, mientras que en la forma UNTIL la condición se comprueba al final del loop.

Los programas que utilizan sólo estas tres estructuras básicas o sus variantes (como la instrucción condicional CASE), y no utilizan la instrucción GOTO, se llaman estructurados.

A partir del trabajo de Dijkstra, la "programación estructurada" (llamada también "programación sin GOTO") se convirtió en la forma de programar más extendida y el término "estructurado" pasó a ser una especie de sello de calidad, a tal punto que las diferentes propuestas metodológicas que fueron apareciendo en el mercado, añadían a su nombre el término estructurado, como lo hicieron Edward Yourdon, Tom De Marco y Ganes & Sarson.

Durante los años 80 fue ganando terreno la idea de que la programación y el desarrollo de sistemas debían dejar de ser una "artesanía" para convertirse en una verdadera rama de la ingeniería y, junto con estas ideas, las diferentes metodologías, como las que arriba citamos, fueron ganando aceptación universal.

A principios de los años 90 las dos metodologías de mayor importancia fueron la Ingeniería de la Información, desarrollada por

James Martin, y la metodología IBM/AD Cycle, creada por la empresa IBM. Ambas incluían consideraciones acerca del uso de herramientas CASE (computer aidded software engeeniering) para el desarrollo de sistemas y pasaron a ser el estándar de referencia para los productores de tales herramientas.

La "revolución de los microcomputadores" y las aplicaciones "tipo windows", permitieron que la década de los 90 nos trajera nuevas maneras de desarrollar aplicaciones -como el enfoque cliente/servidor-, con nuevos lenguajes de programación -Power Builder, Oracle Forms, Visual Basic, C++, Java, etc.- y nuevos enfoques para las tareas de desarrollo de software, dando lugar a la aparición de la programación orientada a objetos (OOP - Object Oriented Programming).

La nueva forma de concebir el software incorpora técnicas y metodologías diferentes a las formas utilizadas tradicionalmente para desarrollar sistemas; técnicas que se fundamentan en el denominado paradigma de la POO, el cual, en líneas generales, concibe el universo computacional como un universo poblado por objetos, cada uno responsable de sí mismo y comunicándose con los demás por medio de mensajes.

Al igual que antaño ocurriera con la programación estructurada, a medida que la programación orientada a objetos fue ganando terreno, la industria comenzó a utilizar sus conceptos en otras actividades, más allá de la programación. Ya en 1996, James Martin y James J. Odell, en su obra Object Oriented Methods - Pragmatic Considerations, señalaban que "las nociones empleadas por los lenguajes de programación OO pueden ser aplicadas como una filosofía general para todo el proceso de desarrollo de sistemas".

En el libro citado, sus autores también señalaban que:

> "Principalmente, esta interpretación más amplia significa que OO es una forma de organizar nuestras ideas acerca del mundo. Esta organización se basa en los tipos de cosas - o tipos de objetos - en nuestro mundo. De esta forma, podemos definir los atributos de estos tipos de objetos, las operaciones que se ejecutan sobre estos tipos de objetos, las reglas basadas en ellos, …"
>
> "En lugar de una unidad física que contiene variables y métodos, un enfoque OO más general nos brinda una forma de organizar conceptualmente nuestro conocimiento."

En el tiempo han ido apareciendo diferentes metodologías OO, muchas de ellas pueden ser catalogadas como orientadas a objetos,

porque se basan en el paradigma de orientación a objetos. Entre ellas, las de mayor difusión han sido:

- Object-Oriented Design (OOD), Grady Booch.
- Object Modeling Technique (OMT), James Rumbaugh.
- Object Oriented Analysis (OOA), Coad/Yourdon.
- Hierarchical Object Oriented Design (HOOD), ESA.
- Object Oriented Structured Design (OOSD), Wasserman.
- Object Oriented Systems Analysis (OOSA), Shaler y Mellor.

En la actualidad, las metodologías más importantes para el análisis y diseño de sistemas han confluido en lo que se ha ido convirtiendo en un importante lenguaje estándar para el modelaje de sistemas, denominado UML (Unified Modeling Language - Lenguaje Unificado de Modelaje), bajo el respaldo de la organización OMG (Object Management Group o Grupo de Administración de Objetos).

UML es, fundamentalmente, la creación de Grady Booch, James Rumbaugh e Ivar Jacobson, conocidos en el medio como “los tres amigos”, quienes habiendo desarrollado metodologías para desarrollo de software con la orientación a objetos, tuvieron la oportunidad de reunir esfuerzos y, como consecuencia de la aceptación general que encontró su propuesta, se pudo crear el Object Management Group, como una organización sin fines de lucro que reúne a la gran mayoría de las empresas líderes en el mercado de la informática: IBM, HP, Sun Microsystems, Microsoft, Intellicorp, Oracle, Texas Instruments, Rational, etc.

UML, como arriba señaláramos, es un lenguaje de modelaje que permite especificar y documentar un sistema y sus componentes. Las herramientas de modelaje que componen UML permiten poner en “blanco y negro” la información sobre la estructura (elementos estáticos) y el comportamiento (elementos dinámicos) de un sistema.

UML no es ni una metodología, ni un lenguaje de programación; existen, sin embargo, diferentes metodologías, como RUP (Rational Unified Process), y sistemas CASE, como Rational y Magic Draw, que ofrecen generadores de código a partir de una especificación en UML, para una gran variedad de lenguajes de programación, como Java y C++.

UML es un lenguaje de modelaje visual, de propósito general, para el desarrollo de software orientado a objetos, que permite representar un sistema y sus componentes con la suficiente precisión como para desarrollar esos componentes -generar el código- y, a la vez, con la

suficiente “independencia tecnológica” como para permitir que el desarrollador utilice las herramientas de construcción que sean de su preferencia.

Como ya indicamos, desde su aparición, alrededor del desarrollo orientado a objetos se propusieron diversos métodos y técnicas, con muchos aspectos en común, pero con distintas notaciones, que dificultaban su aplicación o limitaban el espectro de herramientas y lenguajes de programación que los desarrolladores podían utilizar. El gran valor que tiene UML radica en la estandarización que establece, lo cual facilita su uso universal, así como el desarrollo de productos de software con la capacidad de generar código a partir de una especificación UML, para las más diversas plataformas. De hecho, la mayoría de los productores de sistemas CASE han ido incorporando UML en su conjunto de herramientas.

La OMG, adicionalmente, ha anunciado su Arquitectura Centrada en Modelos (MDA - Model Driven Architecture), señalando que MDA es el siguiente paso en la resolución de problemas de estandarización e integración a través de especificaciones abiertas e independientes de los proveedores.

MDA busca lograr la integración y la interoperatividad de todos los componentes, a través del ciclo de vida de los sistemas, desde el modelaje del negocio y las tareas de diseño, hasta la construcción, integración, prueba, implantación y mantenimiento.

3.- Tecnología de información

En los últimos años se ha ido popularizando el término tecnología de información para denominar a ese gran conjunto que conforman la tecnología de computación, la tecnología de telecomunicaciones y la ingeniería de sistemas.

Podríamos decir que la aparición de la tecnología de información es el producto de tres grandes cambios en el uso de los computadores dentro de las empresas:

- La computación en redes

  Fruto del matrimonio de la tecnología de computación con la tecnología de telecomunicaciones, la computación en redes le da a sus usuarios tanto herramientas personales, como de trabajo en grupo, haciendo posible que las empresas reorganicen sus procesos de trabajo y cambien la naturaleza de

las tareas que se cumplen en sus diferentes unidades de negocio.

- Los sistemas integrados

  El desarrollo de la tecnología de computación y la ingeniería de sistemas, permite que hoy día, además de los de sistemas de apoyo a la administración, las empresas dispongan de facilidades para manejar su información electrónica e instantáneamente, sin barreras departamentales, con alcance corporativo.

- La computación ínter empresarial

  Las nuevas tecnologías -bases de datos ínter empresariales, sistemas de respuesta oral, mensajería electrónica, etc.- han hecho posible el replanteamiento de las relaciones de las empresas con otras organizaciones, para funcionar como negocios integrados, para convertirlas, en la práctica, en una especie de "empresa ampliada".

4.- El ciclo de vida del software

Dentro de los esfuerzos por convertir las tareas de desarrollo de sistemas en una rama de la ingeniería, otra de las contribuciones cruciales ha sido la de crear o definir un esquema del proceso o procedimiento genérico, denominado también ciclo de vida.

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE - Institute of Electrical and Electronics Engineers) define el ciclo de vida de los sistemas como “Una aproximación lógica a la adquisición, el suministro, el desarrollo, la utilización y el mantenimiento del software” (IEEE 1074).

Mientras que el estándar ISO 12207-1, lo define como “Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la implantación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requerimientos hasta la finalización de su uso”.

En cualquier caso, es claro que un modelo de ciclo de vida define un conjunto de fases, etapas y actividades, a través del cual se desarrollan y mantienen los sistemas, lo cual contribuye a crear un marco de referencia para planificar y administrar los proyectos de desarrollo de sistemas.

La primera definición formal de ciclo de vida del software, el modelo en cascada, se atribuye a Winston Royce, a fines de los años 70. Desde entonces, muchos proyectos de desarrollo han seguido ese modelo,

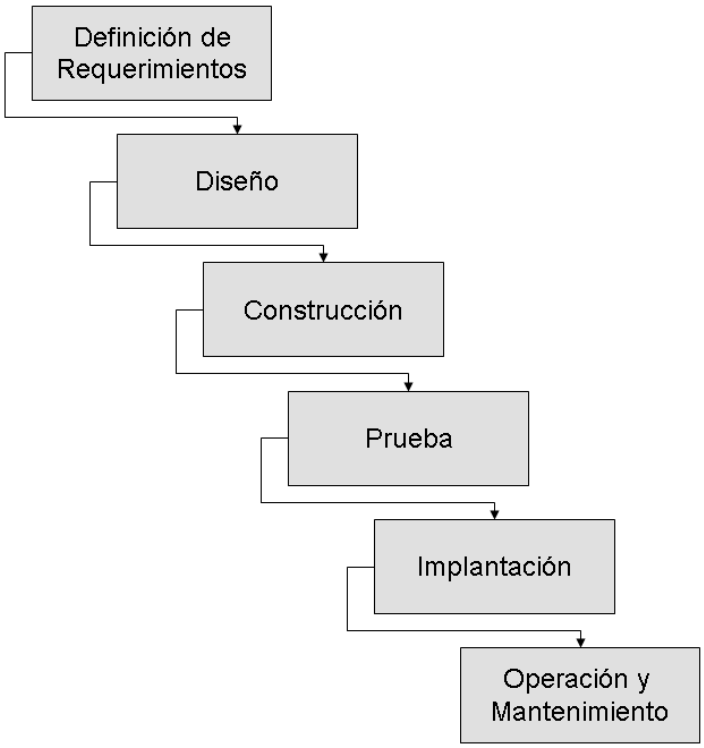
aunque con el tiempo ha sido sujeto a numerosas críticas y las metodologías que han ido surgiendo han presentado enfoques alternativos, que muy brevemente describiremos en los párrafos siguientes.

El gran valor de todos estos modelos radica en que, por una parte, establecen una guía para los ingenieros de sistemas -con el fin de ordenar las diversas actividades técnicas en un proyecto de desarrollo de sistemas- y, por otra parte, suministran un marco para administrar los proyectos, pues permiten estimar recursos, definir puntos de control y crear esquemas para monitorear el progreso.

5.- Enfoques alternativos del ciclo de vida

5.1.- Modelo en cascada

El modelo en cascada que, como ya mencionáramos, constituyó el primer intento de disciplinar "el arte de desarrollar sistemas", es el más simple de todos los modelos; en él se establece que el desarrollo de un sistema puede realizarse a través de una secuencia simple de fases (análisis, diseño, programación, prueba e implantación). Cada fase tiene un conjunto de metas bien definidas, y las actividades que integran cada fase contribuyen al cumplimiento de dichas metas.



El modelo del ciclo de vida en cascada, aportó a la ingeniería de sistemas varios principios o criterios muy importantes:

- Se debe planificar un proyecto antes de iniciarlo.
- Se debe definir el funcionamiento deseado del sistema (requerimientos), antes de diseñar su arquitectura.
- Se debe diseñar un sistema antes de programarlo.
- Se debe probar un sistema después de construirlo y antes de ponerlo en funcionamiento.
- Se deben documentar los resultados de cada actividad.

5.2.- El modelo incremental

La experiencia ha demostrado que los riesgos asociados con el desarrollo de grandes sistemas son enormes; esa misma experiencia ha demostrado que la forma de reducir los riesgos es construir los sistemas por “pedazos”, desarrollando sólo una parte del sistema y reservando otros módulos para “pedazos” o versiones posteriores.

El desarrollo incremental ofrece una forma de llevar a la práctica el criterio de construir por “pedazos” y concibe el proceso de construcción como un proceso que, primero, desarrolla una versión básica -un release básico- del sistema y, paulatinamente va incorporando nuevos módulos para ir atendiendo subconjuntos de requerimientos no atendidos por las versiones anteriores.

El modelo de desarrollo incremental es perfectamente compatible con el modelo de cascada, pues las diferentes versiones pueden ser desarrolladas siguiendo el mismo patrón secuencial de fases.

El modelo de desarrollo incremental brinda ventajas significativas, pues:

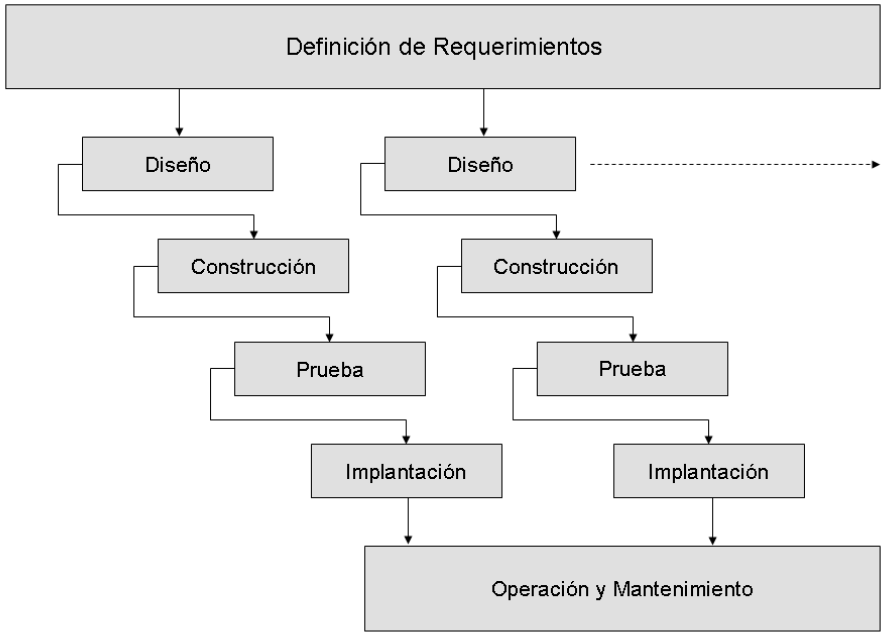
- El usuario puede recibir los beneficios del sistema sin tener que esperar a que esté totalmente construido.
- Construir un sistema pequeño es siempre menos riesgoso que construir un sistema grande.
- Una versión puede incluir la corrección de aquellos errores que se hayan cometido en una versión anterior.
- Al reducirse el tiempo de puesta en marcha de un sistema, se reduce la posibilidad de que los requerimientos funcionales cambien mientras se desarrolla el sistema.

5.3.- El modelo de desarrollo evolutivo

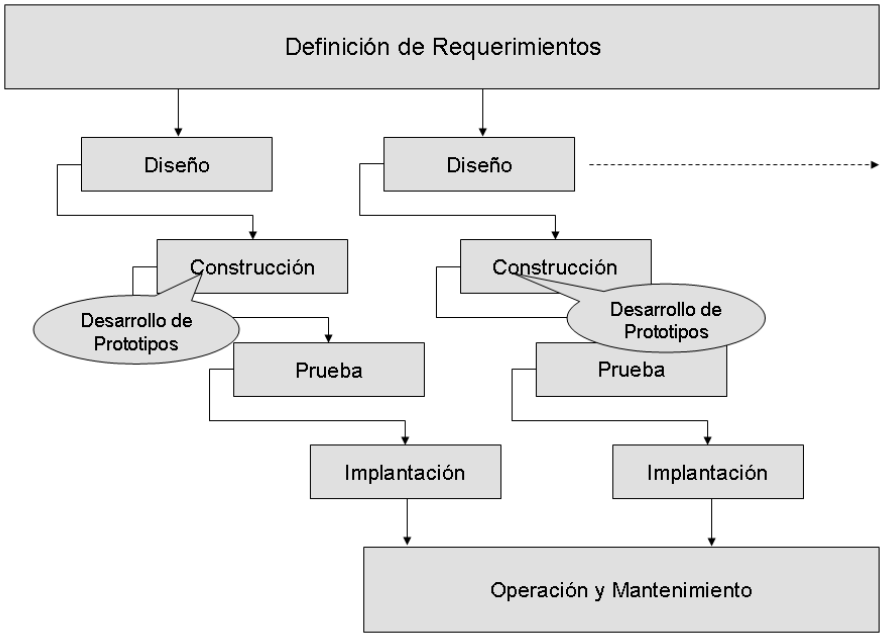
Al igual que el modelo de desarrollo incremental, el modelo de desarrollo evolutivo propone construir un sistema en diferentes versiones -o releases- sucesivas. Sin embargo, mientras que la aproximación incremental busca atender un conjunto completo de requerimientos que se definen al inicio del proyecto, el modelo evolutivo asume que los requerimientos no son totalmente conocidos, sino que van conociéndose a medida que el proyecto avanza.

En el modelo evolutivo, al inicio de una versión, se examinan cuidadosamente los requerimientos detallados y sólo aquellos que hayan sido bien comprendidos se seleccionan para formar parte de la versión.

La versión inicial se desarrolla e implementa, los usuarios comienzan a utilizarla y proveen retroalimentación a los desarrolladores. Con base en esa retroalimentación, se actualiza la especificación de requerimientos y ello permite desarrollar una segunda versión del producto. Este proceso se repite indefinidamente: se instala una versión, los usuarios dan su "feedback", se actualizan las especificaciones funcionales y se desarrolla una nueva versión.



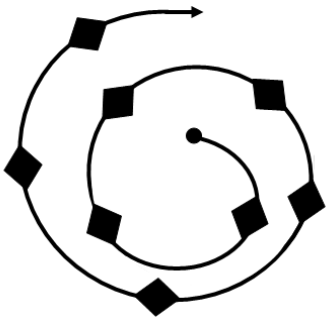
En la práctica, este modelo se combina con el desarrollo incremental, insertándolo en la fase de construcción, para definir los requerimientos detallados de los componentes, a través del desarrollo de prototipos, en cuya construcción se aplica el esquema evolutivo.



El “prototipado” de requerimientos consiste en el desarrollo de una especie de “maqueta” de un módulo y su implementación en ambiente de pruebas, con el propósito de “descubrir” los requerimientos detallados del módulo. Un prototipo se construye de manera rápida y se instala para que los usuarios o clientes puedan experimentar con el prototipo y puedan dar el feedback sobre lo que está bien o no y sobre lo que a ellos les gustó o no les gustó, lo cual permitirá al desarrollador incorporar los cambios necesarios para “refinar y mejorar” el prototipo.

El “prototipado” viene siendo utilizado con mucha frecuencia desde los años 80 y, tanto los diseñadores como los usuarios, encuentran que es mucho más eficiente definir los requerimientos detallados con base en la manipulación de un prototipo, que definiendo largas piezas de documentación, para especificar los detalles de un módulo que, por mucho esfuerzo que se haga, siempre resultará una especificación de requerimientos ambigua y demasiado extensa.

5.4.- El modelo en espiral



En el modelo en espiral del ciclo de vida, el esfuerzo de desarrollo es iterativo. Tan pronto como se completa una iteración -un módulo o esfuerzo de desarrollo-, se da inicio a la siguiente.

En cada iteración se siguen cuatro pasos:

- Determinar qué se desea lograr.
- Determinar las alternativas que pueden ser tomadas para lograr esas metas y, por cada una, analizar los riesgos y resultados finales, con el fin de seleccionar la mejor.
- Seguir la alternativa seleccionada.
- Antes de proceder a la siguiente versión, establecer qué se ha completado, hasta dónde se ha llegado y qué está faltando.

El modelo en espiral resulta muy útil para desarrollar sistemas pequeños en los que se trabaja muy de cerca con el usuario.

5.5.- Combinaciones

Por lo general, la mayoría de las propuestas metodológicas vigentes dividen el ciclo de desarrollo de sistemas combinando algunas de las formas arriba descritas y establecen varias fases o etapas para las que se utilizan diversas denominaciones. Una de ellas, de uso bastante generalizado es la siguiente:

- Fase I - Planificación de Tecnología de Información
- Fase II - Análisis y Diseño General
- Fase III - Diseño Detallado y Construcción
- Fase IV - Pruebas e Implantación
- Fase V - Producción/Mantenimiento

Muchos autores no incluyen, dentro del ciclo de desarrollo de sistemas, la fase de planificación de tecnología de información, sino que consideran que el desarrollo de un sistema se inicia en su fase de análisis y diseño general.

6.- Importancia de las metodologías

Más allá de su contribución a convertir el desarrollo de sistemas en una rama de la ingeniería, la importancia de las diferentes metodologías reside en que integran un conjunto de conceptos, principios y técnicas,

para ofrecer una respuesta coherente a las grandes dificultades que tradicionalmente encuentran los proyectos de construcción de sistemas:

- Problemas de comunicación
- Confusión entre el **qué** y el **cómo**
- Dificultad para manejar la complejidad
- Participación del usuario

6.1.- Problemas de comunicación

Todos los analistas de sistemas saben muy bien lo difícil que resulta describir un procedimiento; es mucho más simple demostrarlo que describirlo rigurosamente. No es de extrañar, por lo tanto, que la comunicación entre el analista y el representante funcional esté rodeada de dificultades, ya que gran parte de las fases del desarrollo de un sistema incluye la descripción de procedimientos.

Las dificultades normales de comunicación entre analista y representante funcional se hacen mayores porque, usualmente, ni el analista conoce los detalles y las particularidades del negocio, ni el usuario entiende las cosas con las que el analista trabaja: programas, especificaciones, descripción de formatos, etc.

Las técnicas y herramientas que ofrecen las diferentes metodologías buscan resolver esas dificultades de comunicación, mediante el uso de modelos y representaciones gráficas que eliminan las ambigüedades en las comunicaciones entre el analista y los representantes funcionales.

6.2.- Confusión entre el qué y el cómo

Los “métodos artesanales” de desarrollo de sistemas, por denominarlos de alguna manera, no establecían una clara diferencia entre la definición de lo qué el sistema debe hacer y la definición de la forma cómo lo llevará a cabo. Cuando no se hace una adecuada distinción entre el qué y el cómo, los requerimientos funcionales pueden quedar mal enunciados o definidos en forma incompleta. A su vez, una definición incompleta o incorrecta de las especificaciones y requerimientos del negocio, lleva a desarrollar sistemas que no satisfacen las necesidades de la organización o que cumplen sólo en forma parcial sus objetivos o que, en el peor de los casos, no se adaptan a la empresa.

Hoy día, las metodologías existentes, permiten diferenciar el qué del cómo y facilitan al analista hacer una separación entre los aspectos conceptuales del sistema (qué hace el sistema), sus aspectos de funcionamiento (cómo funciona o cómo será utilizado) y sus componentes físicos.

6.3.- Manejo de la complejidad de los sistemas

La complejidad del problema que se trata de resolver es una de las causas más importantes de errores de software y de sistemas mal definidos. A mayor complejidad, mayor dificultad para entender el problema y mayor probabilidad de diseñar una solución equivocada.

En nuestros días, el gerente de informática se encuentra arrinconado entre la espada y la pared. Por un lado, la tecnología información ofrece un "paraíso" de sistemas integrales, lo cual lo pone en el camino de atacar sistemas extremadamente complejos y, por el otro, para emprender el desarrollo de sistemas de alcances muy amplios se requiere la participación de numerosos analistas y programadores, cuya productividad decrece en la medida que su cantidad aumenta y cuya dirección se hace muy compleja. Esta realidad hace que el gran reto de la ingeniería de sistemas sea construir esos grandes sistemas integrados, componiendo o agrupando pequeños proyectos de dimensiones "manejables" -equipos de trabajo que no excedan de unas ocho personas- bajo una arquitectura única.

Las metodologías que se ofrecen en el mercado, hoy día, permiten trabajar en forma escalonada o jerárquica (de lo general a lo detallado) y, de esa forma, ayudan a poner en perspectiva los requerimientos funcionales como un todo, a segmentar los grandes sistemas y a definir proyectos de dimensiones manejables, capaces de crear paulatinamente una arquitectura integrada de sistemas.

6.4.- Participación del usuario

La falta de una determinación clara de los roles que deben jugar las diferentes unidades usuarias en el ciclo de vida de los sistemas es causa importante de que los sistemas no se diseñen adecuadamente y de que, por ende, las empresas no utilicen satisfactoriamente sus recursos de computación e informática. En efecto, ¿cómo es posible desarrollar los sistemas necesarios para el negocio si no se cuenta con una guía constante por parte de los expertos del negocio? Sin embargo, la solución no es sólo "asignar un representante funcional". Es necesario establecer el ambiente (técnicas, herramientas, responsabilidades, procedimientos) donde usuarios, analistas y programadores puedan trabajar productivamente. Las metodologías que existen hoy día, permite establecer ese ambiente adecuado.

En general, todas las metodologías de uso más común facilitan la posibilidad de establecer guías claras para el manejo de equipos de

desarrollo de sistemas, capaces de lograr la eficacia -hacer lo correcto- y la eficiencia -hacerlo en la mejor forma-.

7.- Algunas metodologías

En la actualidad existe una gran variedad de propuestas metodológicas. Algunas constituyen una evolución de la metodología estructurada, incorporando elementos que permitan aprovechar los ambientes modernos de desarrollo de sistemas. Otras afirman ser metodologías con "orientación a objetos" y, dentro de ellas, algunas adoptan el estándar UML, arriba citado.

Si bien no pretendemos hacer ni un recuento exhaustivo ni una crítica de las metodologías que existen en el mercado, a continuación citaremos algunas de las más populares, pues en el desarrollo de este libro iremos "echando mano" a algunos de sus conceptos más importantes.

7.1.- La metodología estructurada

Dentro del campo de la ingeniería de sistemas, la metodología estructurada, a partir de los años 70, ha ocupado un lugar muy importante, pues ha sido un instrumento eficaz en el desarrollo de sistemas de alta calidad. Alrededor de las metodologías estructuradas se han integrado un conjunto de experiencias, técnicas y herramientas que sirven para analizar, diseñar y construir sistemas.

La metodología estructurada tiene tres componentes fundamentales:

- Diseño estructurado, definido como "el arte de diseñar los componentes de un sistema y las interrelaciones entre esos componentes en la mejor forma posible", o como "el proceso de especificar la arquitectura, organización y estructura de los componentes interconectados de un sistema con el fin de resolver un problema bien especificado".
- Análisis estructurado, que puede definirse como "el arte de especificar un problema de tal forma que su solución pueda ser desarrollada utilizando el diseño estructurado".
- Programación estructurada, que fue el primer componente de esta metodología, puede ser definida como "un método que, utilizando un grupo específico de estructuras de control, permite desarrollar componentes de software flexibles, simples y fáciles de mantener".

Dos de los aportes más importantes de la metodología estructurada fueron, primero, canalizar el diseño de los sistemas centrando la atención en la información del negocio y, segundo, enfatizando el diseño top-down

(o de arriba-hacia-abajo), partiendo de una visión global, la cual es sucesivamente descompuesta hasta llegar a los componentes primitivos de cada una de las aplicaciones.

7.2.- La Ingeniería de Información

En los años 80, la ingeniería de información cobró enorme importancia pues centraba su objetivo en la administración de los recursos de información dentro de una empresa, con el fin de apoyar o dar soporte a sus objetivos estratégicos. Con estas concepciones, la ingeniería de sistemas comenzó a usar “pantalones largos”, pues la tecnología de información pasaba de ser una herramienta de apoyo administrativo a ser un elemento facilitador para el logro de los objetivos estratégicos del negocio.

James Martin, creador de la ingeniería de información, la definió como “un grupo interrelacionado de técnicas formales con las cuales se construyen los modelos del negocio, tanto de datos como de procesos, con el fin de utilizarlos en la creación y mantenimiento de sistemas ...”.

Si bien desde el punto de vista de las técnicas y las herramientas, la ingeniería de información no era muy diferente a las metodologías estructuradas, introdujo el concepto de trabajar con los repositorios -organización de todos los recursos informáticos de la empresa- y las herramientas CASE.

7.3.- Ingeniería de Redesarrollo

A medida que las empresas fueron reconociendo que su cartera de sistemas era una pieza fundamental para su funcionamiento, se fue otorgando mayor importancia a las tareas de “mantener” el software; es así como comienza a hablarse de la ingeniería de redesarrollo. Ésta forma parte de la ingeniería de sistemas y centra su atención en el reemplazo, la modernización y el mantenimiento de los sistemas existentes en una empresa, con el fin de integrarlos dentro de una arquitectura común y permitir, así, que los recursos de información de esa empresa puedan, como conjunto, apoyar y dar soporte a sus operaciones y a sus objetivos estratégicos.

La ingeniería de redesarrollo incluye técnicas y herramientas que permiten rescatar, renovar o rehabilitar la cartera de sistemas instalados en una empresa y hacer efectiva la “promesa básica” de la ingeniería de información: apoyar la gestión de la empresa en el logro de sus objetivos estratégicos.

La gran mayoría de las empresas que se sirven de computadores para

procesar sus datos han hecho inversiones significativas en desarrollo de sistemas. Estas empresas, al adoptar nuevas tecnologías o nuevas metodologías que centran su atención en la eficacia y eficiencia de los procesos del negocio, se encuentran con una cartera de sistemas obsoleta -conocida en la jerga informática como los sistemas "legacy" (herencia)- que ofrece grandes dificultades de mantenimiento y limitan las posibilidades de dar un soporte integral a las operaciones del negocio. Normalmente ello plantea la necesidad de rehacer una gran cantidad de sistemas, para lo cual no existen ni los recursos ni el tiempo necesario. La ingeniería de redesarrollo busca ofrecer soluciones a esos problemas.

En muchos casos, una gran cantidad de estos sistemas "legacy" cumple satisfactoriamente con los requerimientos funcionales básicos -el sistema de nómina procesa correctamente los pagos a los empleados, el sistema de cuentas por cobrar produce estados de cuenta correctos, etc.-. Sin embargo, se hace difícil integrar estos sistemas y sus archivos o bases de datos, porque hacer cualquier cambio o adición a su estructura resulta muy complejo, bien sea por la documentación -muchas veces inexistente-, por la estructura de sus componentes -como es el caso de los "programas espagueti"- o por la tecnología bajo la que fueron desarrollados -como es el caso de aplicaciones bajo IDMS, COBOL, PL/1 no compatibles con los sistemas en red-.

Es así que han ido apareciendo diversos productos de "middleware"[1], que junto con la ingeniería de redesarrollo han ido cobrando cada día más importancia, pues buscan ofrecer soluciones para integrar en su totalidad la cartera de sistemas y hacer de ésta el instrumento de apoyo que requiere la empresa para el manejo eficiente de sus operaciones y para el logro de sus objetivos estratégicos.

La ingeniería de redesarrollo incluye, a su vez, tres diferentes tipos de disciplinas:

- La Reingeniería
- La Ingeniería en Reverso
- La Resistematización o Ingeniería Transversal

La reingeniería tiene como objetivo desarrollar nuevos componentes de software y nuevas estructuras de datos a partir de componentes o estructuras de datos existentes.

[1] El middleware es software que actúa como intermediario entre ambientes no compatibles y permiten que el usuario final trabaje en ellos como si se tratara de ambientes integrados.

La ingeniería en reverso se plantea un ciclo de trabajo que va en sentido opuesto (de allí su nombre) al ciclo normal de desarrollo de sistemas, al cual se le denomina también ingeniería "hacia adelante" (forward engineering).

La aplicación de la ingeniería en reverso permite regenerar la documentación preparada originalmente durante el desarrollo del sistema o aplicación, la cual es, en la mayoría de los casos, obsoleta o incompleta al momento de querer realizar un cambio a ese sistema. La regeneración de esta documentación ayuda a mejorar la productividad de las tareas de mantenimiento a la cartera de aplicaciones instaladas, pues, como lo indica F. J. Buckley, en la publicación "Standars", IEEE Computer, Vol. 22, No. 11, Nov. 1989, pp. 69:

> "Para mantener el software necesitamos entenderlo primero. Esto supone la aplicación de alguna mecánica para comprender la construcción del software, su marco conceptual y su arquitectura. Necesitamos alguna visión de alto nivel que no contenga todos los detalles del listado fuente. Necesitamos un documento que ayude a una persona que no esté familiarizada con el programa a entender rápidamente lo que hace el programa. En principio, la documentación preparada durante el desarrollo de software es la más apta para dar esta visión de alto nivel"

A largo plazo, la ingeniería en reverso se plantea el objetivo de extraer o rescatar, de las aplicaciones en producción, las especificaciones del negocio para almacenarlas en un "repositorio" o base de datos de especificaciones. Esto, a su vez, permitiría desarrollar nuevas versiones de esas aplicaciones haciendo uso de ingeniería "hacia adelante" y herramientas CASE.

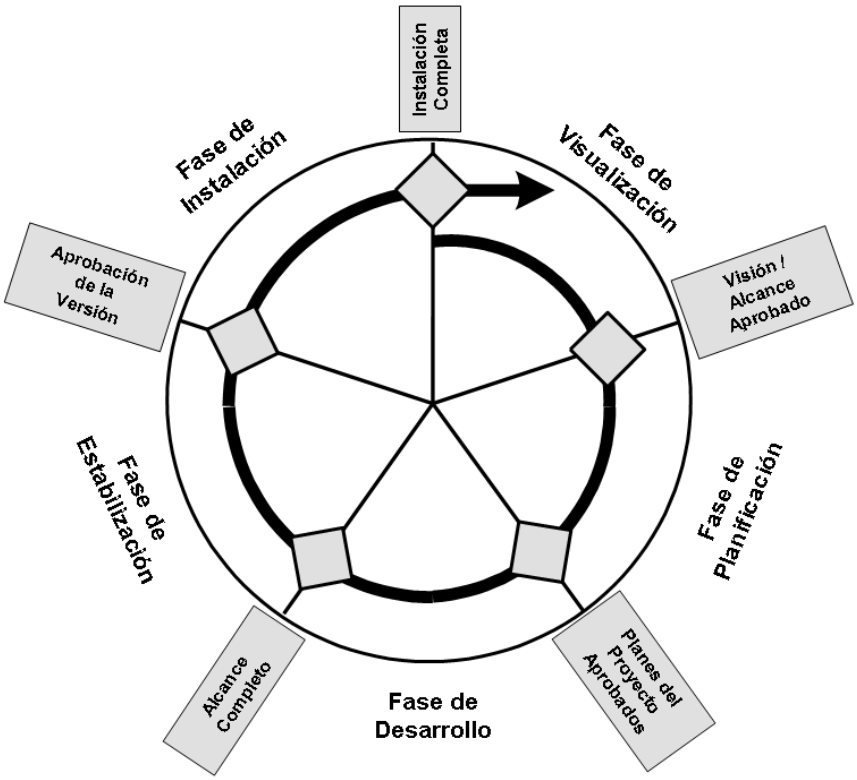
La resistematización o ingeniería transversal combina la ingeniería en reverso y la reingeniería, pues busca desarrollar nuevos componentes partiendo de especificaciones o modelos que han sido creados con ingeniería en reverso.

La resistematización, puede ser útil para renovar aquellas aplicaciones que actualmente producen resultados satisfactorios para el negocio, pero a las que se requiere añadir nuevas funciones o cambiar de plataforma tecnológica.

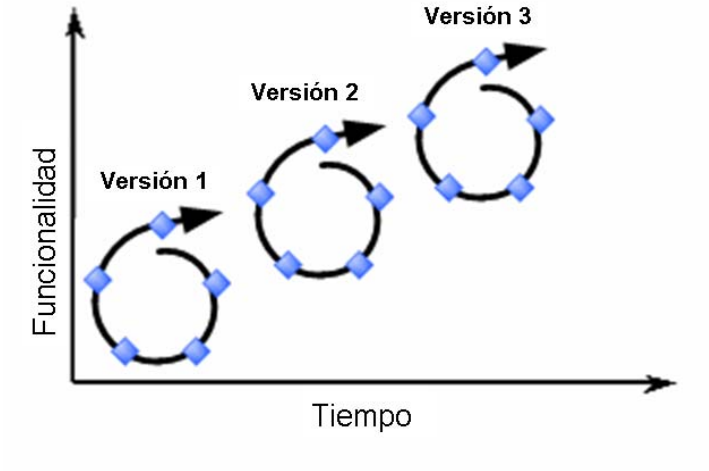
7.4.- El marco de soluciones de Microsoft - MSF

La versión 3.0 de MSF (Microsoft Solutions Framework) constituye una metodología que integra las diferentes concepciones del ciclo de vida.

En esta metodología se adopta la estrategia de desarrollo de sistemas por versiones, que en la terminología que utiliza Microsoft se denomina solución.

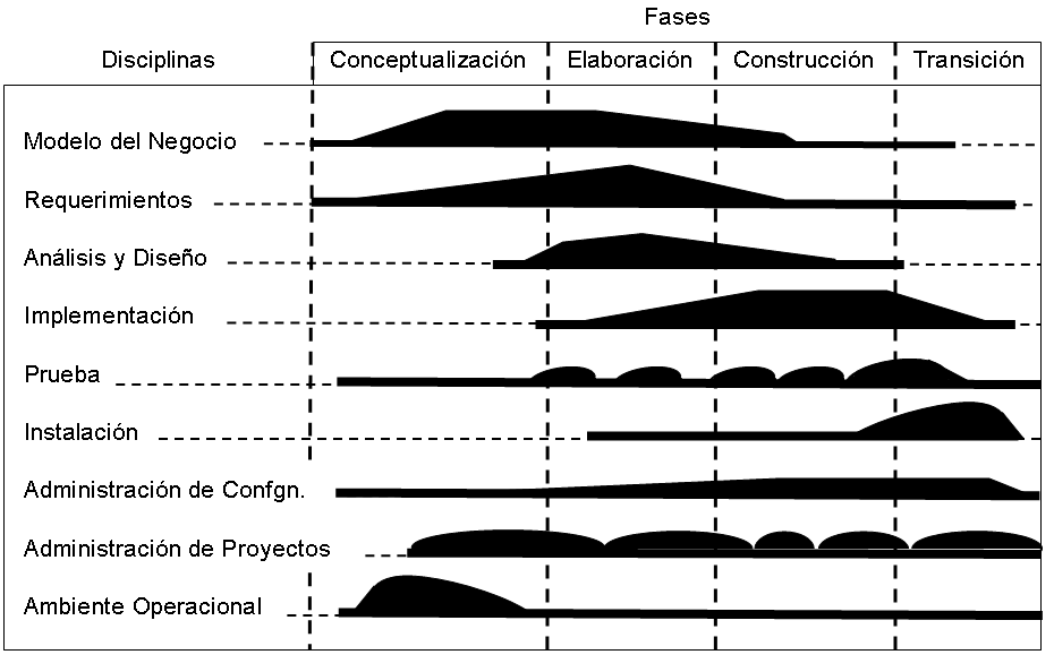


Cada solución se desarrolla desde su concepción hasta su implantación definitiva en producción, siguiendo una secuencia de cinco fases (Visualización, Planificación, Desarrollo, Estabilización e Instalación), cada una de las cuales finaliza con el cumplimiento de un hito -milestone- específico, tal como se aprecia en la figura.



7.5.- El Rational Unified Process - RUP

El modelo RUP (Rational Unified Process) presenta el ciclo de desarrollo desde dos perspectivas diferentes: la perspectiva gerencial y la perspectiva de desarrollo.



Desde la perspectiva gerencial, el ciclo de desarrollo de un sistema se cumple a través de cuatro fases: Conceptualización, Elaboración, Construcción y Transición.

1. **Preparación Inicial o Conceptualización** (Inception)

   El objetivo principal de esta fase es establecer los objetivos del sistema, en ella se establece el caso del negocio (business case), con el fin de delimitar el alcance del sistema y el alcance del proyecto.

2. **Preparación Detallada o Elaboración** (Elaboration)

   El objetivo principal de esta fase es establecer la arquitectura del producto. En ella se realiza el levantamiento de la mayor parte de los requerimientos funcionales, analizando los riesgos que pudieran amenazar el logro de los objetivos del sistema.

3. **Construcción** (Construction)

   El objetivo principal de esta fase es desarrollar el producto. En esta fase, a través de sucesivas iteraciones e incrementos,

se desarrolla un producto de software, listo para operar, normalmente denominado Versión Beta.

4. **Transición** (Transition)

   El objetivo principal de esta fase es instalar el producto, una vez realizadas las pruebas de aceptación y habiendo efectuado los ajustes y correcciones que sean requeridos.

Desde la perspectiva de desarrollo, el ciclo RUP se cumple a través del desarrollo de sucesivas versiones que paulatinamente van siendo cada vez más completas. Las actividades que se cumplen en cada iteración, esto es, para producir cada nueva versión, se agrupan en Flujos de Trabajo Centrales -Core Workflows-.

Cada uno de estos flujos centra su atención en un aspecto específico del sistema: Modelaje del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba, Instalación, Administración de Cambios, Administración de Proyectos y Ambiente Operacional.