
1


Ingeniería
de Software

1. Ingeniería de software

La ingeniería de software es una disciplina de ingeniería que ofrece métodos, técnicas para desarrollar y entregar software de calidad. Mantiene las aplicaciones de software utilizando tecnologías y manejo de proyectos.

La ingeniería de software se relaciona con diversas áreas de la informática. Es aplicable en negocios, investigación científica, medicina, producción, banca, meteorología, derecho, internet, entre otras.

Se puede definir como "*Un conjunto de conocimientos y técnicas aplicadas al diseño, construcción y documentación de los programas*", para desarrollar, operar y mantener los objetivos de la ingeniería de software, los cuales son:

- Mejorar la calidad de los productos de software.
- Aumentar la productividad y trabajo de los ingenieros del software.
- Facilitar el control del proceso de desarrollo de software.
- Suministrar a los desarrolladores las bases para construir software de alta calidad en una forma eficiente.
- Definir una disciplina que garantice la producción y el mantenimiento de los productos de software, dentro del costo estimado.

El proceso de desarrollo de software se refiere a la traducción de las necesidades del usuario en requerimientos de software, éstos son transformados en el diseño del sistema, posteriormente el diseño es implementado en código y finalmente el código es probado y documentado para su uso operativo. De manera general define ¿Quién hace? ¿Qué?, ¿Cuándo se hace? y ¿Cómo alcanzar el objetivo deseado?

1.1 Historia de la ingeniería de software

Inicialmente la programación era un arte que no disponía de métodos sistemáticos en los cuales basarse para la realización de productos de software. El software se realizaba sin ninguna planificación, posteriormente, desde mediados de los 60 hasta finales de los 70 se caracterizó por el establecimiento del software como un

producto que se desarrollaba para la distribución de sectores específicos de la sociedad.

La búsqueda de la calidad y el permitir reducir los costos de las soluciones ha sido uno de los objetivos más perseguidos desde los inicios de la informática. A mediados de los 60, la creación de un producto de software se convertía en una tarea angustiosa, por lo cual fue necesario introducir una serie de herramientas y procedimientos que facilitaran por un lado, la labor de creación de nuevo software y por otro, la comprensión y el manejo del mismo. Éstos fueron los inicios de la ingeniería del software. Con el paso del tiempo, la evolución de estos métodos han llevado a reconocer la ingeniería del software como una verdadera disciplina, derivada de una investigación seria y de un estudio minucioso.

Durante los primeros años de la era de la computadora el desarrollo del software se realizaba sin ninguna planificación lo que trajo consigo, que muchas de las empresas que desarrollaban software comenzaran a descontrolarse en cuestión de planificación de un proyecto y los costos comenzaron a incrementar. Los programadores trataban de tener una buena organización y con un esfuerzo heroico, a menudo los proyectos salían con gran éxito.

La mayoría del software que se desarrollaba era considerado software propietario, es decir era desarrollado para el uso exclusivo de un reducido grupo de personas u organización. Debido a este entorno personalizado del software, el diseño era un proceso implícito, realizado en la mente de alguien y la documentación normalmente no existía.

Los sistemas de computadora se extienden desde la mitad de la década de los sesenta hasta finales de los ochenta. La multiprogramación y los sistemas multiusuario introdujeron nuevos conceptos de interacción hombre-máquina. Las técnicas interactivas abrieron un nuevo mundo de aplicaciones y nuevos niveles de sofisticación del hardware y del software. Los sistemas de tiempo real podían recoger, analizar y transformar datos de múltiples fuentes, controlando así los procesos y produciendo salidas más rápidas Los avances en los dispositivos de

almacenamiento en línea condujeron a la primera generación de sistemas de gestión de bases de datos.

Las empresas desarrolladoras de sistemas tenían que corregir su software existente cuando se detectaban errores, cambiaban los requisitos de los usuarios o se tenía que adaptar a nuevos dispositivos de hardware que se hubieran adquirido. A este conjunto de actividades se le nombro mantenimiento del software que forma parte importante del desarrollo de software.

Con los sistemas distribuidos en múltiples computadoras, cada una ejecutando funciones concurrentes y comunicándose con alguna otra, incrementó notablemente la complejidad de los sistemas informáticos. Las redes de área local y de área global, las comunicaciones digitales de gran capacidad de transmisión y la creciente demanda de acceso "instantáneo" a los datos, supusieron una fuerte presión sobre los desarrolladores del software. La principal característica fue la llegada y amplio uso de los microprocesadores.

Entran las potentes máquinas personales controladas por sistemas operativos sofisticados, en redes globales y locales, acompañadas por aplicaciones de software avanzadas.

La industria del software ya es la cuna de la economía del mundo. Las técnicas para el desarrollo del software están cambiando en la forma en que la comunidad construye programas informáticos. Las tecnologías orientadas a objetos están desplazando rápidamente los enfoques de desarrollo de software más convencionales en muchas áreas de aplicaciones.
Sin embargo, un conjunto de problemas relacionados con el software ha persistido a través de la evolución de los sistemas basados en computadora, y estos problemas continúan aumentando, como los que se listan a continuación:

1. La habilidad de construir nuevos programas no pueden ir al mismo ritmo de la demanda de éstos.

2. La dificultad de construir programas lo suficientemente rápido como para cumplir las necesidades del mercado y de los negocios.
3. El uso excesivo de computadoras y dependencia de las operaciones fiables del software. Cuando el software falla, pueden ocurrir daños económicos enormes.
4. El no poder construir software informático que tengan fiabilidad y la calidad necesaria para el usuario.
5. El soporte y mejora de los programas es amenazada por diseños pobres y recursos inadecuados.

En respuesta a estos problemas, la práctica de la ingeniería del software se está adoptando en toda la industria.

En nuestra actualidad ingeniería de software ha cambiado la cultura del mundo debido al extendido uso de la computadora. El correo electrónico (e-mail), la WWW, permiten a la gente comunicarse en nuevas formas. El software baja el costo y mejora la calidad de los servicios.

La ingeniería de software se puede considerar como la ingeniería aplicada al software, esto es con base a herramientas preestablecidas, la aplicación de las mismas de la forma más eficiente y óptima; objetivos que siempre busca la ingeniería. No es sólo la resolución de problemas, sino más bien teniendo en cuenta las diferentes soluciones, elegir la más apropiada.

En los años noventa los estándares de la ingeniería de software y la madurez de proceso han caracterizado la industria del software como una disciplina madura. En un nivel más técnico, la ingeniería de software comienza con una serie de tareas que hacen modelos y que resultan en una especificación completa de requisitos y una representación comprensiva de diseño del software que será construido.

El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos. Ha pasado de ser una resolución de problemas especializadas y una herramienta de análisis de información, a ser una industria por sí misma. Cada uno de estos elementos compone una configuración que se crea como parte del proceso de la ingeniería del software. El objetivo de la ingeniería del

software es proporcionar un marco de trabajo para construir software con mayor calidad.

Las economías de los países desarrollados dependen en gran parte del software, más y más sistemas son actualmente controlados por software. La ingeniería de software concierne a teorías, métodos y herramientas para el desarrollo profesional de software.

1.2 Conceptos básicos

La ingeniería de software es una disciplina de la Ingeniería que concierne a todos los aspectos de la producción de software, los Ingenieros de Software adoptan un enfoque sistemático para llevar a cabo su trabajo y utilizan las herramientas y técnicas necesarias para resolver el problema planteado, de acuerdo a las restricciones de desarrollo y recursos disponibles.

Hay que dejar claro cuál es la diferencia entre ingeniería de software y computación, la primera concierne sólo al desarrollo de sistemas o productos de software y la segunda concierne a la teoría, así como a los fundamentos de cualquier sistema de cómputo, sea de hardware o de software.

La ingeniería de sistemas concierne a todos los aspectos del desarrollo de sistemas basados en cómputo, que incluyen hardware, software y el proceso de ingeniería, mientras que la ingeniería de software es solo parte de este proceso.

¿Qué es el Software? Es un producto que consiste de programas de cómputo y su documentación asociada. Los productos de software genéricos son producidos por una organización para ser vendidos al mercado, mientras que los productos hechos a la medida son sistemas desarrollados bajo pedido a un desarrollador específico.

Un buen producto de software debe tener las siguientes características:

Confiable: El software no debe causar daños físicos o económicos en el caso de fallos. Es un término necesario que sea separado en varios elementos, los cuales permiten darle al software el fiabilidad, éstos son:

- Consistencia y precisión.
- Solidez.
- Simplicidad.
- Calidad en los procesos de desarrollo.

Eficiente: Debe realizar las funciones establecidas con resultados confiables, ejecutar todas las operaciones que se requieren en un tiempo aceptado y que sea fácilmente usado por el grupo de usuarios a quien está dirigido.

El software debe contar con una interfaz de usuario adecuada y su documentación. La importancia de estas características depende del tipo de producto y en el ambiente en el que será utilizado. Algunos de estos atributos pueden predominar dependiendo del tipo de sistema que se requiera. Los costos tienden a crecer exponencialmente si son requeridos altos niveles de alguna de estas características.

Facilidad para dar mantenimiento: Debe ser posible que el software evolucione y que siga cumpliendo con sus especificaciones. Involucra los elementos que simplifican la labor de prevención, corrección o ampliación del código del programa. Retomar un código escrito meses antes es un trabajo dispendioso y agobiante, en especial cuando las aplicaciones no cuentan con la característica a la cual aquí se hace referencia. Las siguientes características se pueden considerar como atributos de este aspecto:

- Exactitud y claridad en la documentación
- Modularidad acoplamiento
- Facilidad de lectura
- Simplicidad

El software contiene instrucciones de computadora, descripción de las estructuras de datos, algoritmos, componentes de software, procedimientos y funciones escritas en algún lenguaje de programación. El software puede dividirse de la siguiente forma:

- Por su estructura:
    - Funcionales.
    - Orientados a objetos.
    - Orientados a listas.
    - Orientados a componentes.
- Por su función:
    - Programas o Sistemas de Usuario
    - Interfaces Hombre-Máquina.
    - Herramientas de Software.
    - Librerías.
    - Sistemas de Uso Genérico: Compiladores, Procesadores de Texto, etc.
    - Bases de Datos.
    - Sistemas Basados en Web.

- Por su plataforma de cómputo:
    - Sistemas Embebidos.
    - Sistemas de Cómputo Distribuido.
    - Sistemas de Cómputo Paralelo.
    - Sistemas de Tiempo Real.
    - Sistemas Basados en Chips.

El proceso de software es un conjunto de actividades requeridas para desarrollar un sistema de software como son:

Especificación - Establecer los requerimientos y restricciones del sistema, qué debe hacer el software y cuáles son sus especificaciones de desarrollo.

Diseño - Producir un modelo en papel del sistema.

Manufactura - Se comienza con construcción del sistema.

Prueba - Verificar que el sistema cumpla con las especificaciones requeridas.

Instalación - Entregar el sistema al usuario y asegurar su funcionamiento.

Mantenimiento - Reparar los errores en el sistema cuando sean descubiertos.

Se debe tomar en cuenta que el proceso de software debe ser confiable (los errores son localizados antes de la entrega), robusto (no interrumpe las operaciones ante problemas inesperados), de fácil mantenimiento y rápido (rapidez en la construcción del sistema).

Todos los tipos de software requieren que los analistas, diseñadores y desarrolladores apliquen una metodología de desarrollo para que se logren productos a las necesidades del usuario.

El éxito de un proyecto involucra elementos como la planeación, la administración y la utilización de metodologías de desarrollo de software. A través de la planeación se determinan los recursos necesarios para el desarrollo del proyecto, la factibilidad del mismo y el tiempo estimado de desarrollo; unido a ello con la administración se controla, evalúa y corrige la dirección de acuerdo a las contingencias y demás elementos que se vayan presentando durante el desarrollo; finalmente, a través del uso de una metodología de desarrollo de software constituyen uno de los mecanismos que actualmente se utilizan para lograr que los participantes trabajen en conjunto y garanticen un buen producto de software.

1.3 Clasificación de las metodologías de análisis de sistemas

La metodología para el desarrollo de software es un método sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito.
La finalidad del uso de la metodología de análisis de sistemas es disminuir el tiempo, corregir y controlar cada etapa de desarrollo de un programa. Lo que nos permitirá de forma sistemática obtener un producto correcto y libre de errores.

Actualmente podemos clasificar las metodologías de análisis, con base en el paradigma de programación (Figura 1.1), estas son:

Metodologías estructuradas

La orientación de esta metodología se dirige hacia los procesos que intervienen en el sistema a desarrollar, es decir, cada función que realice el sistema se fragmenta en pequeños módulos individuales para resolver de una manera más sencilla el problema y posteriormente unirlos en una solución.

Metodologías orientadas a objetos

Estas metodologías no comprenden los procesos como funciones sino que arma módulos basados en componentes, es decir, cada componente es independiente del otro, lo que nos permite obtener un código reutilizable.

Metodologías Ágiles

Estas metodologías ofrecen una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

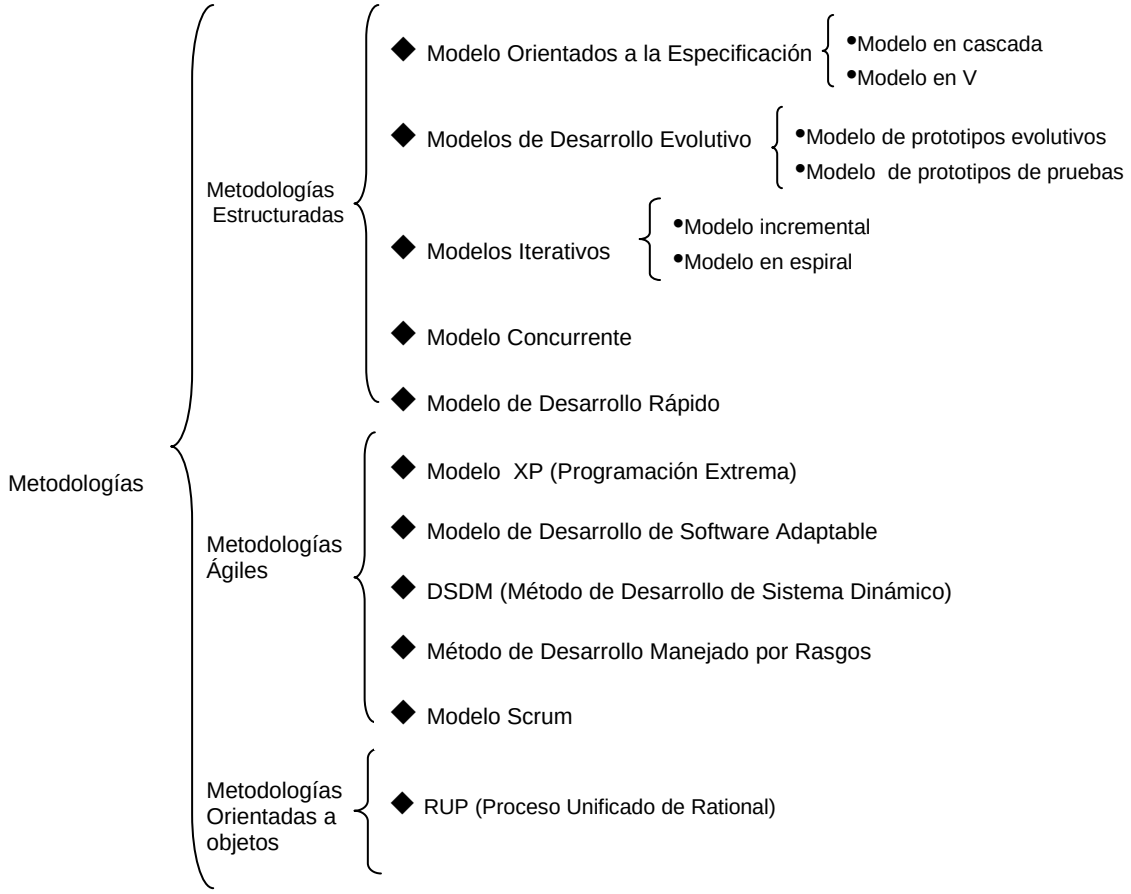

Figura 1.1 Clasificación de las metodologías de análisis de sistemas

1.4 Metodologías de análisis de sistemas

Las metodologías nos indican las actividades a realizar para lograr el producto de software deseado, mostrando además qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y los procedimientos necesarios para comenzar.

1.4.1 Proceso y características de las metodologías

Ciclo de Vida

Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de sistemas, intenta determinar el orden de las etapas involucradas y los criterios de transición asociadas entre estas etapas.

El ciclo de vida del software se encarga de:

- Describir las fases principales de desarrollo de software.
- Definir las fases primarias esperadas de ser ejecutadas durante esas fases.
- Administra el progreso del desarrollo.

Elementos del Ciclo de Vida

Se compone de fases sucesivas compuestas por tareas planificables. Según el modelo de ciclo de vida, la sucesión de fases puede ampliarse con ciclos de realimentación, de manera que conceptualmente se considere una misma fase y se pueda ejecutar más de una vez a lo largo de un proyecto, recibiendo en cada iteración aportaciones de resultados que se van produciendo (realimentación).

Los elementos que integran un ciclo de vida son:

Las fases. Una fase es un conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Se construye agrupando tareas que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación de tareas impone requisitos temporales correspondientes a la asignación de recursos. Cada fase viene definida por un conjunto de elementos observables externamente como son las actividades con las que se relaciona, los datos de entrada (resultados de la fase anterior, documentos o productos requeridos para la fase, experiencias de proyectos anteriores), los datos de salida (resultados a utilizar por la fase posterior, experiencia acumulada, pruebas o resultados efectuados) y la estructura interna de la fase.

Los entregables. Son los productos intermedios que se generan en cada fase. Pueden ser materiales (componentes, equipos) ó inmateriales (documentos, software). Los entregables permiten evaluar la marcha del proyecto mediante comprobaciones de posibles modificaciones a los requisitos funcionales y de las condiciones de realización previamente establecidos. Cada una de estas evaluaciones puede servir, además, para la toma de decisiones a lo largo del desarrollo del proyecto.

Metodologías estructuradas

a) Modelos orientados a la especificación

Se adoptan procesos de desarrollo de software estructurado definiendo actividades a realizar delimitadas por especificaciones, diseño y desarrollo con tiempos intermedios de separación entre ellas, en las cuales se suponían acabadas las actividades de las etapas anteriores.

Modelo en cascada. Este es el modelo básico que sirve como bloque de construcción para los demás de ciclo de vida tradicionales, consta en realizar el desarrollo de software a través de una secuencia simple de fases (Figura 1.2). Cada fase tiene un conjunto de metas bien definidas, y las actividades dentro de la fase contribuyen a la satisfacción de metas de esa fase o quizás a una subsecuencia de metas de la fase.

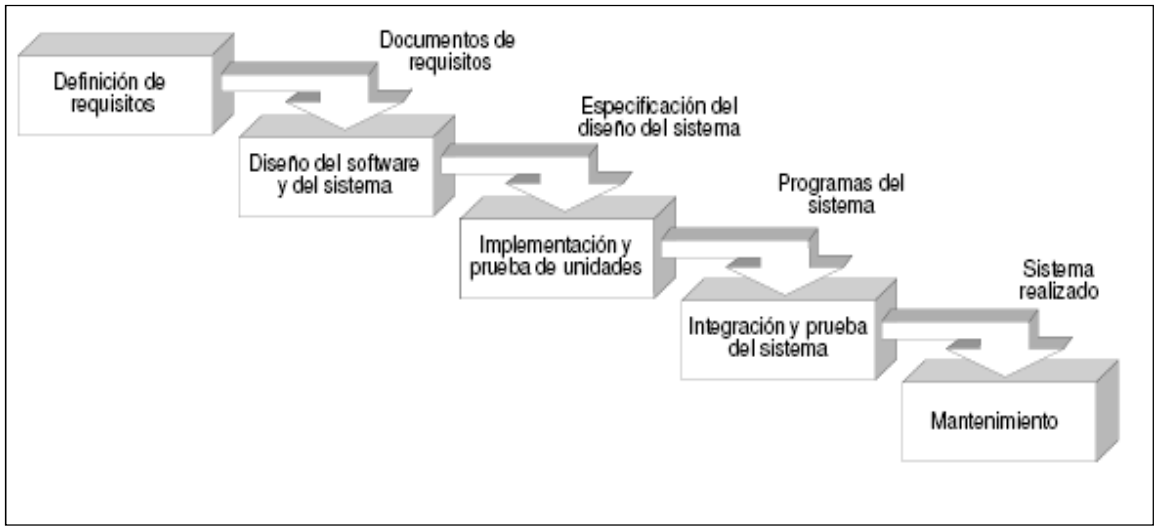

Figura 1.2. Modelo en cascada

Ventajas

- La ventaja de este modelo radica en su sencillez ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.

Desventajas

- Los proyectos reales raramente siguen el flujo secuencial que propone el modelo, siempre hay iteraciones y se crean problemas en la aplicación del paradigma.
- Normalmente, es difícil para el cliente establecer explícitamente al principio todos los requisitos del sistema, los cuales son necesarios para este modelo, además tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos productos.
- El cliente debe tener paciencia, ya que solo se tendrá una versión operativa del sistema hasta llegar a las etapas finales del proyecto.
- El riesgo es mayor que el de otros modelos, pues en lugar de hacer pruebas de aceptación al final de cada etapa, las pruebas comienzan a efectuarse luego de haber terminado la implementación, lo que puede traer como consecuencia un retroceso de todo el proceso.
- El modelo no contempla la posibilidad de retornar a etapas, cosa que en la realidad puede ocurrir.

Modelo en V. Entre las muchas variantes surgidas del modelo en cascada está el modelo en V (Figura 1.3), que resuelve en parte el problema de la simplicidad del modelo de cascada que sólo muestra una salida por proceso y el hecho de que esta salida no puede ser reinsertada como parte de la entrada de etapas anteriores.

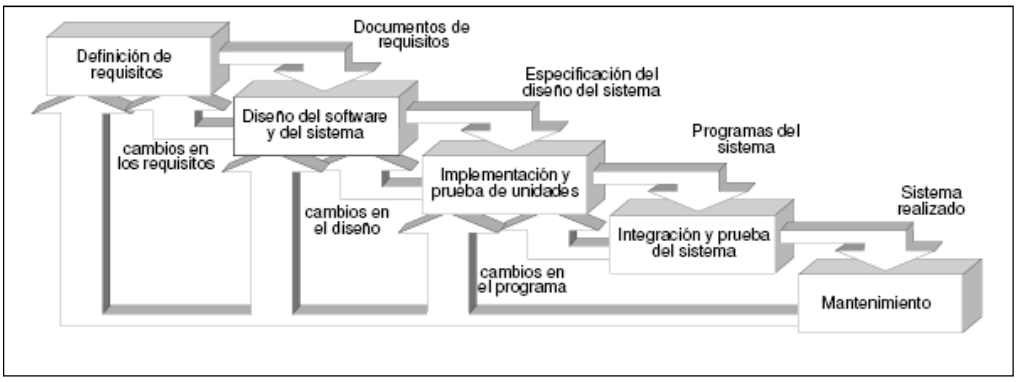

Figura 1.3: Modelo en V

Ventajas

- El modelo involucra validaciones de cada una de las etapas del modelo de cascada.

Desventajas

- Se toma toda la complejidad del problema de una vez y no en iteraciones o ciclos de desarrollo, lo que disminuye el riesgo.

b) Modelos de desarrollo evolutivo

Los documentos producidos al final de cada etapa pueden convertirse en obstáculos ya que las sucesivas etapas no podrán empezar hasta que éstos no se acaben. En los modelos evolutivos se produce un sistema rudimentario inicial que evoluciona según las necesidades del cliente hasta cumplir con los requisitos últimos de éste. Al no requerir pre-especificaciones detalladas tampoco exige una costosa producción de documentación por etapas.

Las etapas de los modelos evolutivos son:

1) Formulación de un esquema de los requisitos del sistema como guía para los programadores (aunque no necesariamente incompleta ni consistente).
2) Desarrollo de un sistema, tan rápido como sea posible, basado en las especificaciones anteriores.
3) Evaluación y modificación del sistema según vayan así especificándolo los propios usuarios.

Modelo de prototipos evolutivos. El modelo de desarrollo evolutivo construye una serie de grandes versiones sucesivas de un producto. Sin embargo, mientras que la aproximación incremental presupone que el conjunto completo de requerimientos es conocido al comenzar, el modelo evolutivo asume que los requerimientos no son completamente conocidos al inicio del proyecto.

En el modelo evolutivo, los requerimientos son cuidadosamente examinados, y sólo los que son bien comprendidos son seleccionados para el primer incremento. Los desarrolladores construyen una implementación parcial del sistema que recibe sólo estos requerimientos. El sistema es entonces desarrollado, los usuarios lo usan, y proveen retroalimentación a los desarrolladores. Basada en esta retroalimentación, la especificación de requerimientos es actualizada, y una segunda versión del producto es desarrollada y desplegada. El proceso se repite indefinidamente (Figura 1.4).

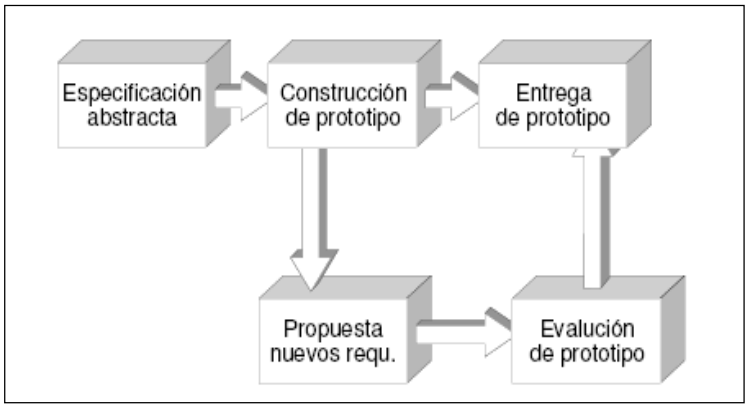

Figura 1.4: Modelo de desarrollo de prototipos evolutivos de software

Ventajas

- Todo lo que se tiene que hacer es construir un subconjunto de requerimientos conocidos (incremental), y comprender al principio que muchos nuevos requerimientos es probable que aparezcan cuando el sistema sea desplegado o desarrollado.

Desventajas

- El desarrollo de software en forma de prototipos evolutivos requiere un especial cuidado en la manipulación de documentos, programas, datos de pruebas, etc. desarrollados para distintas versiones del software.
- Cada paso debe ser registrado, la documentación debe ser recuperada con facilidad y los cambios deben ser efectuados de una manera controlada.

Modelo de prototipos de pruebas. En el modelo de prototipos de pruebas se crea una implementación parcial de un sistema, para el propósito explícito de aprender sobre los requerimientos del sistema (Figura 1.5). Un prototipo es construido de una manera rápida; esto es dado a los usuarios, clientes o representantes de ellos, posibilitando que ellos experimenten con el prototipo. Estos individuos luego proveen la retroalimentación sobre el prototipo proporcionado, quienes capturan en la documentación actual de la especificación de requerimientos la información entregada por los usuarios para el desarrollo del sistema real. Cuando los requerimientos son comprendidos se desecha del prototipo y se empieza a desarrollar el sistema real.

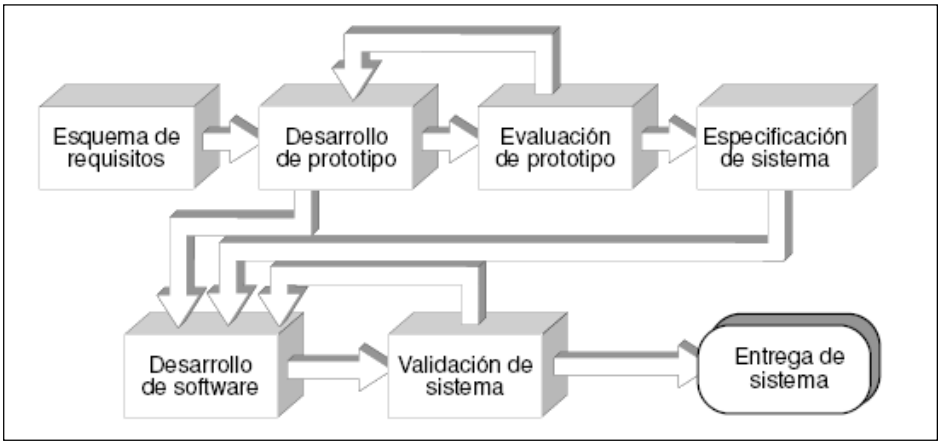

Figura 1.5: Modelo de desarrollo de prototipos de pruebas desechables

Ventajas

- El prototipo puede ser usado como parte de la fase de requerimientos (determinar requerimientos) o puede servir para algún o todo el desarrollo incremental en modelos incrementales o evolutivos.
- El prototipo es construido con los requerimientos generales y más pobremente entendidos.

Desventajas

- El desarrollo de software requiere un especial cuidado en la manipulación de documentos, programas, datos de pruebas, etc.

c) Modelos iterativos

La idea principal detrás de mejoramiento iterativo es desarrollar un sistema de programas de manera incremental, permitiéndole al desarrollador sacar ventaja de lo que se ha aprendido a lo largo del desarrollo anterior, incrementando versiones entregables del sistema. El aprendizaje viene de dos vertientes: el desarrollo del sistema, y su uso. Los pasos claves en el proceso eran comenzar con una implementación simple de los requerimientos del sistema, e iterativamente mejorar la secuencia evolutiva de versiones hasta que el sistema completo esté implementado. En cada iteración, se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema.

Modelo incremental. Sugerido por Mills, en 1980, divide el proceso en conjuntos de procesos menores, que se identifican y resuelven por separado. En el modelo incremental (Figura 1.6) el cliente identifica y esquematiza los servicios esperados por el sistema priorizándolos. Esta priorización es tenida en cuenta a la hora del desarrollo, posteriormente de la esquematización, y división surgen fragmentos a ser resueltos por distintas partes del sistema, una vez que uno de los incrementos se ha completado y puesto en funcionamiento, el cliente puede usarlo teniéndose así en cuenta sus sugerencias tanto para mejorar esa parte como para reestructurar y mejorar las demás.

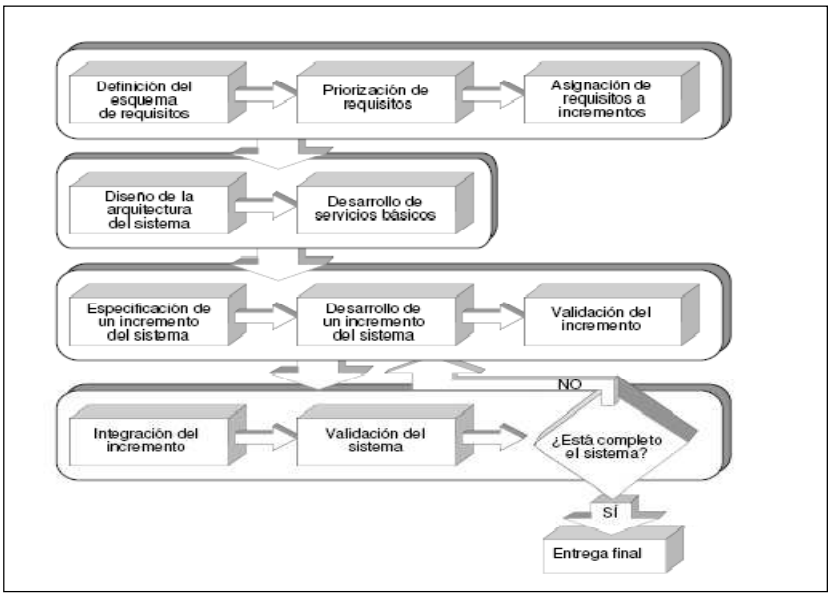

Figura 1.6: Modelo de desarrollo incremental

Ventajas

- El sistema es dividido en procesos menores.
- El desarrollo del sistema es creado tomando encuesta la prioridad de los requerimientos, es decir, el sistema primero cumple con los requerimientos más importantes para el usuario, permitiéndole satisfacer dichas necesidades aunque el sistema no esté terminado.

Desventajas

- La identificación de los incrementos son relativamente pequeños y deben cubrir alguna funcionalidad, siendo difícil, en muchos casos asociar requisitos del cliente con funcionalidades del sistema. Los requisitos, normalmente no son independientes y la consecución de uno implica el desarrollo de varias funcionalidades.
- Exige la implantación de una infraestructura común a varias partes del sistema. Esto dificulta aún más la independencia de las partes del sistema.
- Problemas de la gestión de los contratos, ya que partes del sistema no se saben cuándo van a empezar, es difícil llevar un sistema de contrataciones y organizar el trabajo del personal involucrado, que usualmente debe fijarse con horarios y fechas desde el principio.

Modelo en espiral. En el modelo en espiral cada parte se resuelve independientemente teniendo en cuenta los costos y riesgos. Propuesto inicialmente por Boehm en 1988. Ve el desarrollo del proyecto como una espiral, desde una concepción inicial, en el centro, hasta el desarrollo final del sistema (Figura 1.7), está basado en una constante evolución de los riesgos involucrados ya que antes de entrar a la fase siguiente se debe tener una completa evaluación de riesgos, estos riesgos son considerados falta de información de manera que para resolverlos es necesario recolectar información y analizarla.

Entre sus principales características se encuentran la de permitir revisiones regulares del progreso logrado frente a objetivos bien definidos, permitiendo eliminar los problemas usuales en el modelo de prototipos evolutivos.

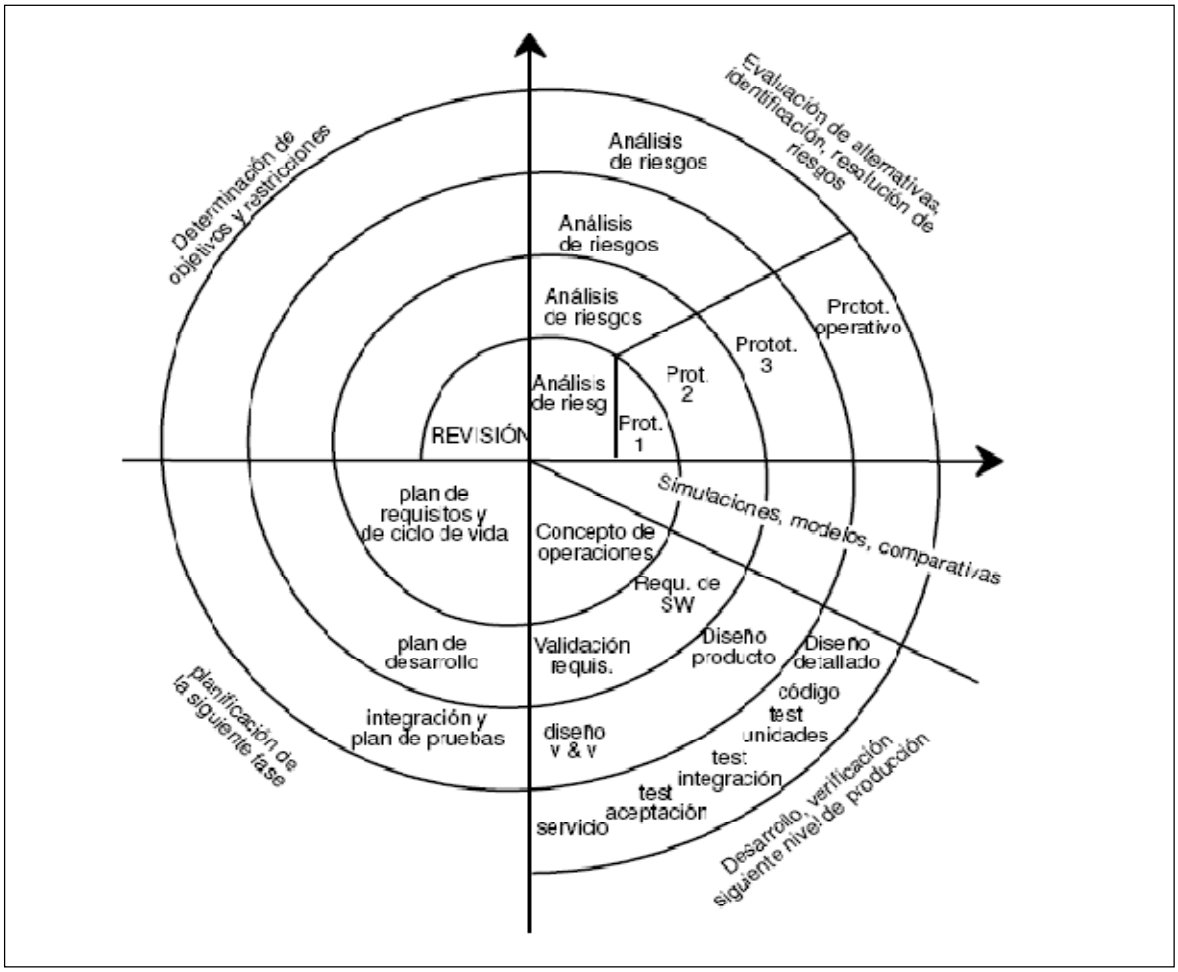

Figura 1.7 Modelo en espiral de Boehm original de 1988

Ventajas

- En sistemas pequeños donde no se presenten un gran riesgo en la obtención de los requisitos se convierte en un sencillo modelo en cascada
- Como el software evoluciona, a medida que progresa el proceso, el desarrollador y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos.
- Permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.
- Demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto.
- Reduce los riesgos.

Desventajas

- La dificultad del análisis de riesgos, por lo que se requiere de personas especializadas y con experiencia en el análisis de riesgos.
- Los problemas de contratación, ya que este modelo trata de evitar decisiones prematuras, que son, por otro lado esenciales a la hora de prever fechas de contratación.

d) Modelo concurrente

Llamado algunas veces ingeniería concurrente, es un modelo de tipo de red donde todas las personas actúan simultáneamente o al mismo tiempo.

El término ingeniería concurrente se ha usando desde 1986, cuando el Instituto para el Análisis de la Defensa de Estados Unidos lo describió en su reporte R-388. Mejora el enfoque secuencial de la producción tradicional mediante tres elementos principales:

- Una arquitectura computacional distribuida que permite la sincronización, la programación óptima de tareas y el manejo adecuado de flujos de información.
- Una representación unificada de toda la información de diseño y manufactura, de forma que pueda visualizarse e interpretarse desde diversas perspectivas.
- Un conjunto de herramientas computacionales que permiten desarrollar prototipos a bajo costo, de forma óptima e inteligente.

Sus principales características son: ser un modelo que esta dirigido por las necesidades del usuario, las decisiones de la gestión de los procesos de desarrollo y los resultados de las revisiones de los prototipos.

Ventajas

- El enfoque concurrente permite la realización simultánea de todas las tareas de desarrollo hasta la fabricación del prototipo.

- Funciona utilizando equipos de trabajo multidisciplinarios teniendo un gran énfasis en el manejo de rutas de información más que de jerarquías organizacionales.
- El modelo de desarrollo concurrente es aplicable a todo tipo de desarrollo de software y proporciona una imagen exacta del estado actual de un proyecto.

Desventajas

- Define una red de actividades, donde todas las actividades de la red existen simultáneamente con otras.

e) Modelo de desarrollo rápido

El Desarrollo Rápido de Aplicaciones (DRA-Rapid Application Development) es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto. DRA es una adaptación a alta velocidad en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un sistema completamente funcional dentro de periodos cortos de tiempo.

La limitación de tiempo impuesto en un proyecto DRA demanda un ámbito en escalas. Si una aplicación de gestión puede modularse de forma que permita completarse cada una de las funciones principales en menos de tres meses, es un sistema candidato para ser desarrollado usando DRA. Cada una de las funciones pueden ser enfrentadas por un equipo DRA diferente y ser integradas en un solo conjunto.

Ventajas

- Su principal ventaja es el desarrollar sistemas funcionales en poco tiempo.

Desventajas

- Para proyectos grandes, el DRA requiere de recursos humanos suficientes como para crear el número correcto de equipos DRA.
- DRA requiere clientes y desarrolladores comprometidos en las rápidas actividades necesarias para completar un sistema en un marco de tiempo abreviado. Si no hay compromiso, por ninguna de las partes constituyentes, los proyectos DRA fracasarán.

Metodologías ágiles

En febrero de 2001 nace el término ágil aplicado al desarrollo de software, cuyo objetivo es esbozar los valores y principios que deberán permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Estas metodologías se basan en 12 principios básicos para el desarrollo de software:

- La prioridad es satisfacer al cliente mediante oportunas y continuas entregas de software que le aporte un valor.
- Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
- Entregar frecuentemente, software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
- La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
- Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
- El software que funciona es la medida principal de progreso.

- Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.
- Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
- En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Ventajas

- Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo, ya que se incorpora una planificación al proceso de desarrollo.
- Entrega continua y en tiempos cortos del software funcional.
- Trabajo continuo entre el cliente y el equipo de desarrollo.
- Minimizan los costos de los cambios de requerimientos y alcances.
- Permite al cliente no tomar decisiones prematuras, mediante técnicas establecidas por la metodología especifica, permitiendo no realizar inversiones innecesarias en tecnología.

Desventajas

- Falta de documentación del diseño, lo cual hace difícil el mantenimiento y uso del sistema.
- Presenta una fuerte dependencia de las personas, ya que se evita en lo mas posible la documentación y el diseño convencional, los proyectos ágiles dependen en gran medida de las personal del equipo de desarrollo

a) Metodología XP (Programación Extrema)

Las raíces de la XP yacen en la comunidad de Smalltalk, y en particular la colaboración cercana de Kent Beck y Ward Cunningham a finales de los 80´s. Ambos refinaron sus prácticas en numerosos proyectos a principios de los 90´s, extendiendo sus ideas de un desarrollo de software adaptable y orientado a la gente.

La XP empieza con cuatro valores: Comunicación, Retroalimentación, Simplicidad y Coraje. Construye sobre ellos una docena de prácticas que los proyectos XP deben seguir. Muchas de estas prácticas son técnicas antiguas, tratadas y probadas, incluyendo la mayoría de los procesos planeados.

En esta plataforma XP construye un proceso de diseño evolutivo que se basa en refactorar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras. El resultado es un proceso de diseño disciplinado, lo que es más, combina la disciplina con la adaptabilidad de una manera que indiscutiblemente la hace la más desarrollada entre todas las metodologías adaptables.

b) Modelo de desarrollo de software adaptable

Creado por Highsmith, la cual ve a la planificación como una paradoja en un ambiente adaptable, ya que los resultados son naturalmente imprevisibles. En la planificación tradicional, las desviaciones del plan son errores que deben corregirse. En el ambiente adaptable, sin embargo, las desviaciones nos guían hacia la solución correcta.

En este ambiente imprevisible se necesita que las personas colaboren de la mejor manera para tratar con la incertidumbre. La atención de la gerencia es menor en lo que tiene que hacer la gente, y mayor sobre la comunicación alentadora para que las personas puedan proponer las respuestas creativas ellos mismos.

En ambientes predictivos, el aprendizaje se desalienta a menudo. Las cosas se ponen de antemano y entonces se sigue ese diseño. Highsmith se enfoca directamente en fomentar las partes difíciles del desarrollo adaptable, en particular cómo fomentar la colaboración y el aprendizaje dentro del proyecto.

c) DSDM (Método de Desarrollo de Sistema Dinámico)

El DSDM empezó en Gran Bretaña en 1994 como un consorcio de compañías del Reino Unido, el método empieza con un estudio de viabilidad y negocio. El estudio de viabilidad considera si DSDM es apropiado para el proyecto. El estudio de

negocio es una serie corta de talleres para entender el área de negocio dónde tiene lugar el desarrollo. También propone arquitecturas de esbozos del sistema y un plan del proyecto.

El resto del proceso forma tres ciclos entretejidos:

- **El ciclo del modelo funcional** produce documentación de análisis y prototipos.
- **El ciclo de diseño del modelo** diseña el sistema para uso operacional.
- **El ciclo de implantación** se ocupa del despliegue al uso operacional.

DSDM tiene principios subyacentes que incluyen una interacción activa del usuario, entregas frecuentes, equipos autorizados, pruebas a lo largo del ciclo. Como otros métodos ágiles usan ciclos de plazos cortos de entre dos y seis semanas. Hay un énfasis en la alta calidad y adaptabilidad hacia requisitos cambiantes.

d) Método de desarrollo manejado por rasgos

El Desarrollo Manejado por Rasgos (FDD siglas en inglés) fue desarrollado por Jeff De Luca y el viejo gurú de la OO Peter Coad. Se enfoca en iteraciones cortas que entregan funcionalidad tangible. En el caso del FDD las iteraciones duran dos semanas.

El FDD tiene cinco procesos. Los primeros tres se desarrollan al principio del proyecto.

- Desarrollar un Modelo Global
- Construir una Lista de los Rasgos
- Planear por Rasgo
- Diseñar por Rasgo
- Construir por Rasgo

Los últimos dos se hacen en cada iteración. Cada proceso se divide en tareas y se da un criterio de comprobación. Los desarrolladores entran en dos tipos: dueños de clases y programadores jefe.

Los programadores jefe son los desarrolladores más experimentados. A ellos se les asignan rasgos a construir. Sin embargo ellos no los construyen solos. Sólo identifican qué clases se involucran en la implantación de un rasgo y juntan a los dueños de dichas clases para que formen un equipo para desarrollar ese rasgo. El programador jefe actúa como el coordinador, diseñador líder y mentor mientras los dueños de clases hacen gran parte de la codificación del rasgo.

e) Metodología Scrum

Scrum divide un proyecto en iteraciones (llamada carreras cortas) de 30 días. Antes de que comience una carrera se define su funcionalidad requerida y entonces se deja al equipo para que la entregue. El punto es estabilizar los requisitos durante la carrera.

Sin embargo la gerencia no se desentiende durante la carrera corta. Todos los días el equipo sostiene una junta corta (quince minutos), llamada scrum, dónde el equipo discute lo que hará al día siguiente. En particular muestran a los bloques de la gerencia: los impedimentos para progresar que se atraviesan y que la gerencia debe resolver. También informan lo que se ha hecho para que la gerencia tenga una actualización diaria de dónde va el proyecto.

Scrum se enfoca principalmente en la planeación iterativa y el seguimiento del proceso. Es muy cercana a las otras metodologías ágiles en muchos aspectos y debe funcionar bien con las prácticas de código de la XP.

Metodología Orientada a objetos

En general el desarrollo orientado a objetos se realiza de forma iterativa e incremental. Es iterativo por que las tareas de cada fase se llevan a cabo de forma iterativa, a la vez que existe un ciclo de desarrollo de análisis, diseño e implementación que permite hacer evolucionar al sistema.

La parte incremental se lleva a cabo al dividir el sistema en un conjunto de particiones, cada una de las cuales se desarrolla de manera completa, hasta, que se finaliza el sistema.

Al implementar estas dos formas de desarrollo permite que las actividades de validación, verificación y aseguramiento de la calidad se puedan realizar. Para cada iteración de cada fase de cada incremento en el desarrollo del sistema, es decir, de una forma continua.

El modelo orientado a objetos se caracteriza por:

- La eliminación de fronteras entre fases, ya que debido a la naturaleza iterativa del desarrollo orientado a objetos, estas fronteras se difuminan cada vez más.
- Una nueva forma de concebir los lenguajes de programación y su uso, ya que se incorporan bibliotecas de clases y otros componentes reutilizables.
- Un alto grado de iteración y solapamiento, lo que lleva a una forma de trabajo muy dinámica.

Ventajas

- Permite realizar un modelo de sistemas casi independientemente de los requisitos del proyecto.
- El modelo se establece de una forma similar al razonamiento humano, y por lo tanto son más fáciles de entender.
- Nos permite crear sistemas complejos, ya que se apoya en el desarrollo, de un lenguaje de programación orientado a objetos.
- Nos permite dar mantenimiento más fácil al sistema.
- Los modelos orientados a objetos representan el mundo real de una forma lo más fiel posible.

Desventajas

- Representa mayor dificultad el realizarlo, ya que los modelos orientados a objetos exigen una mayor abstracción de los requisitos del sistema.
- Requieren gran compresión en cuestión de los requerimientos del sistema.

RUP (Proceso Unificado de Rational)

RUP es una metodología sólida, con documentación, que apoya el ciclo de vida evolutivo incremental, además de basarse en el desarrollo de componentes y al de orientado a objetos. Según Kruchten, RUP es un proceso de ingeniería de software que provee un enfoque disciplinado para la asignación de tareas y responsabilidades dentro de una organización desarrolladora de software.

Los autores de RUP destacan que el proceso de software tiene tres características esenciales:

- Dirigido por los Casos de Uso, los cuales no sólo especifican el diseño, si no también guían el diseño, implementación y pruebas.
- Centrado en la arquitectura, es decir se centra en la organización o estructura de las partes más relevantes del sistema, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo.
- Iterativo e incremental, en donde el trabajo se divide en partes más pequeñas o mini proyectos. Permitiendo que el equilibrio entre Casos de Uso y arquitectura se vaya logrando durante cada mini proyecto, así durante todo el proceso de desarrollo.

La ventaja principal de la metodología RUP es que cada una de las fases del desarrollo puede ser repetida, ordenadamente, cuando se desee para implementar nuevos cambios. Tiene una forma disciplinada de asignar tareas, administrar los requisitos y verifica la calidad continuamente.

Ventajas

- Tiene con conjunto de elementos de planificación (plan de desarrollo, plan de iteración, plan de calidad, etc.) que permiten controlar el desarrollo de software.

- Determina un esquema de escalabilidad y gestión de riesgos con el cual detecta problema y fallos de una forma temprana, con la finalidad de prevenirlos y corregirlos.
- Define que artefactos son necesarios para realizar alguna actividad, así como, que artefactos deben ser creados en cada actividad.

Desventajas

RUP representa un proceso basado en la documentación de todo el sistema, por lo cual su uso no es muy aconsejable para sistemas a desarrollar en un corto plazo.