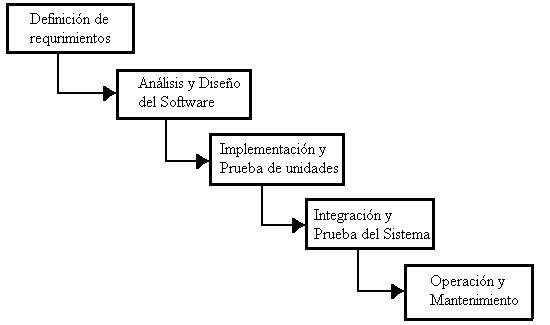
**MODELO LINEAL SECUENCIAL (CASCADA)**

También llamado "Ciclo de vida básico" o "Modelo de cascada" tiene su origen en el "Modelo de cascada" ingeniado por Winston Royce, aunque omite los muchos bucles de este último. El Modelo Lineal Secuencial sugiere un enfoque sistemático o más bien secuencial del desarrollo de software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento. El Modelo Lineal Secuencial acompaña las siguientes actividades:

**GRÁFICO DEL MODELO LINEAL SECUENCIA "CASCADA"**



**Análisis de los requerimientos del software:**

Es la fase en la cual se reúnen todos los requisitos que debe cumplir el software. En esta etapa es fundamental la presencia del cliente que documenta y repasa dichos requisitos.   
   
   
   
 **Diseño:**

Es una etapa dirigida hacia la estructura de datos, la arquitectura del software, las representaciones de la interfaz y el detalle procedimental (algoritmo). En forma general se hace un esbozo de lo solicitado y se documenta haciéndose parte del software.   
   
  

**Generación del código:**

 Es la etapa en la cual se traduce el diseño para que sea comprensible por la máquina. Esta etapa va a depender estrechamente de lo detallado del diseño. 

**Pruebas:**

Esta etapa se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en la detección de errores.   
   
 

**Mantenimiento:**

Debido a que el programa puede tener errores, puede no ser del completo agrado del cliente o puede necesitar, eventualmente acoplarse a los cambios en su entorno. Esto quiere decir que no se rehace el programa, sino que sobre la base de uno ya existente se realizan algunos cambios.

 El Modelo Lineal Secuencial es el paradigma de desarrollo de software más antiguo que existe, sin embargo esto no ha impedido que se haya creado una desconfianza alrededor de él basada en los siguientes errores reales:

* Los proyectos raramente siguen el paradigma secuencial que propone el proyecto.
* A menudo es difícil que el cliente exponga exactamente todos los requisitos.
* El cliente debe tener paciencia.

Los responsables del desarrollo de software siempre se retrasan innecesariamente. Todo lo anteriormente expuesto es cierto pero este paradigma tiene un lugar bien definido e importante en el trabajo de la Ingeniería de Software aparte de proporcionar una plantilla en la que se encuentran métodos para análisis, diseño, codificación, pruebas y mantenimiento. Con todo y sus errores, sigue siendo el paradigma más utilizado en el desarrollo del software, siendo mucho mejor que un enfoque al azar.

**Características del modelo**

* Primer modelo empleado (Royce, 1970), también denominado ciclo de vida clásico y modelo lineal secuencial.
* Consiste en la ejecución secuencial de una serie de fases que se suceden, lo que da nombre al modelo.
* Cada fase genera documentación para la siguiente. Esta documentación debe ser aprobada.
* Una fase no comienza hasta que la anterior ha terminado.
* Requiere disponer de unos requisitos completos y precisos al principio del desarrollo.
* Se disponga de unos requisitos completos y consistentes al principio del desarrollo.
* Sea un proyecto pequeño, en el que el período de congelación de los requisitos es corto, o un proyecto con unos requisitos bastante estables.

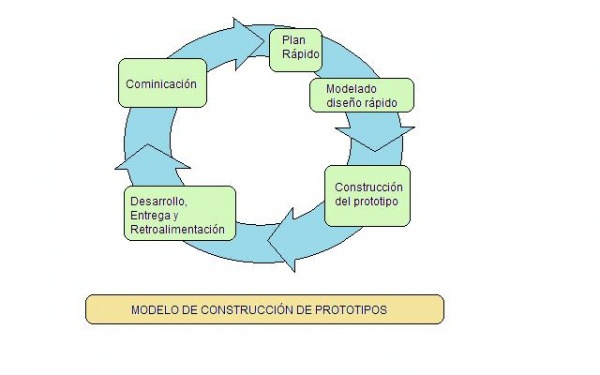
**Modelo de construcción de prototipos**   
 

Este modelo no secuencial, basado en la construcción de simulaciones o modelos ejecutables de aplicaciones más extensos, persigue un objetivo principal: la participación directa del cliente en la construcción del software requerido. Las fases son similares a las del modelo en cascada: es necesario un análisis previo de los requisitos tanto del sistema como del cliente, se concibe la arquitectura del sistema y se realiza el diseño del software. Sin embargo, se incluye un elemento hasta ahora no utilizado, que consiste en el diseño rápido de un prototipo que se mostrará al cliente para que evalúe el trabajo realizado. 

El prototipo es una versión reducida del programa completo; es una “fachada virtual” que mostramos al cliente (que carece de la posibilidad de ser utilizada de la forma en que lo haríamos con el software final. Tras recoger los requisitos tanto del cliente como del sistema, se comienza con el diseño rápido del prototipo; el diseño completo obedece al previo diseño de pequeños prototipos específicos para funciones individuales. Más tarde, estos diseños serán unidos en uno sólo.

Después, se procede a la construcción del mismo. Éste prototipo es el que mostraremos al cliente para que lo evalúe y considere cambios en él, aunque no se trate de una versión definitiva.   
   
 

**Gráfico modelo de construcción de prototipos**

**Comunicación:**   
   
**tener una interacción con el cliente para evaluar la petición del software y determinar si el programa a desarrollar es un buen candidato para construir un prototipo.Debido a que el cliente debe interaccionar con el prototipo para determinar el refinamiento del proyecto**   
   
 

**Plan rápido:**   
   
**cuando se tienen que los resultados de un proyecto son aceptables, se procede a desarrollar una representación abreviada de los requerimientos.Antes de que pueda comenzar la construcción de un prototipo, en este se debe representar los dominios funcionales y de información del programa. La aplicación de estos principios de análisis fundamentales, pueden realizarse mediante los métodos de análisis de requerimientos.**   
   
   
   
   
Modelado diseño rápido:   
   
Después de que se haya revisado la representación de los requerimientos, se crea un conjunto de especificaciones de diseño abreviadas para el prototipo.El diseño debe ocurrir antes de que comience la construcción del prototipo. Sin embargo, el diseño de un prototipo se enfoca normalmente hacia la arquitectura a nivel superior y a los aspectos de diseño de datos.   
   
 

   
Construcción de prototipo:   
   
El software del prototipo se crea, prueba y se corrigen Idealmente todos los posibles errores, los bloques de construcción de software preexisten se utilizan para crear el prototipo de una forma rápida y se determina si un prototipo es funcional o no. Para las aplicaciones interactivas con el hombre,es posible frecuentemente crear un prototipo en papel que describa la interacción hombre-maquina.   
 

   
Desarrollo y entrega:   
   
Una vez que el prototipo ha sido probado, se presenta al cliente, el cual "conduce la prueba" de la aplicación y sugiere modificaciones. Este paso es el núcleo del método de construcción de prototipo. Es aquí donde el cliente puede examinar una representación implementada de los requerimientos del programa, sugerir modificaciones que harán al programa cumplir mejor las necesidades reales.   
   
   
   
Los pasos 4 y 5 se repiten iterativamente hasta que todos los requerimientos estén formalizados o hasta que el prototipo haya evolucionado hacia un sistema de producción.

**Construcción de prototipos:**   
   
**este modelo básicamente se comienza elaborando un prototipo del producto final: qué aspecto tendrá, cómo funcionará a**demás de los aspectos ya mencionados el prototipo debe ser construido en poco tiempo, usando los programas adecuados y no se debe utilizar mucho dinero pues a partir de que este sea aprobado nosotros podemos iniciar el verdadero desarrollo del software. Pero eso si al construir el prototipo nos asegura que nuestro software sea de mejor calidad, además de que su interfaz sea de agrado para el usuario.    
 

**Tipos de modelos de prototipos**

**Modelo de Prototipos rápido:**Metodología de diseño que desarrolla rápidamente nuevos diseños, los evalúa y prescinde del prototipo cuando el próximo diseño es desarrollado mediante un nuevo prototipo.

**Modelo de Prototipos reutilizable:** También conocido como "Evolutionary Prototyping"; no se pierde el esfuerzo efectuado en la construcción del prototipo pues sus partes o el conjunto pueden ser utilizados para construir el producto real.

**Modelo de Prototipos Modular:** También conocido como Prototipo Incremental (Incremental prototyping); se añaden nuevos elementos sobre el prototipo a medida que el ciclo de diseño progresa.

**Modelo de Prototipos Horizontal:**El prototipo cubre un amplio número de aspectos y funciones pero la mayoría no son operativas. Resulta muy útil para evaluar el alcance del producto, pero no su uso real.   
 

**Modelo de Prototipos Vertical:** El prototipo cubre sólo un pequeño número de funciones operativas. Resulta muy útil para evaluar el uso real sobre una pequeña parte del producto.

**Modelo de Prototipos de Baja-fidelidad:** El prototipo se implementa con papel y lápiz, emulando la función del producto real sin mostrar el aspecto real del mismo. Resulta muy útil para realizar tests baratos.

**Modelo de Prototipos de Alta-fidelidad:** El prototipo se implementa de la forma más cercana posible al diseño real en términos de aspecto, impresiones, interacción y tiempo.

**conclusiones**

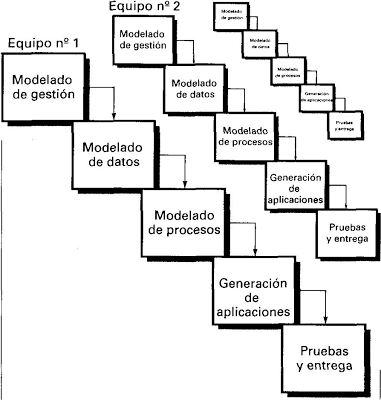
Este modelo es utilizado básicamente para facilitar el desarrollo de un producto de software mediante la definición de  requisitos que permitan satisfacer las necesidades del cliente y el usuario además de facilitarle el trabajo al desarrollador, la clave esta en definir las reglas con las que se iniciara el proyecto esto básicamente es para que halla una buena relación entre el cliente y desarrollador  para que estén de acuerdo con el prototipo, el tiempo, la tecnología que se implementara en el proyecto.

este modelo como lo pudimos ver consiste en la representación de un diseño rápido el mismo que me permite establecer incluso el nivel de aceptación que tendrá el software a desarrollarse. Así mismo es muy fácil de utilizar y de modificar es utilizado para establecer aspectos del sistema que no son contemplados bien mediante la retroalimentación , que consiste en analizar las alternativas y en caso de no cubrir las expectativas del cliente se procede a repetir las etapas para que el sistema a desarrollar sea de calidad

**MODELO DE DESARROLLO RAPIDO DE APLICACIONES(DRA)**

Este es un modelo de proceso de desarrollo del software lineal, secuencias que enfatiza un ciclo de desarrollo extremadamente corto. El modelo DRA es una adaptación a del modelo lineal secuencial 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

**GRÁFICO DEL MODELO DRA**

**Modelado de Gestión:**

El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión?, ¿Qué información se genera?, ¿Quién la genera?, ¿A dónde va la información?, ¿Quién la procesa?.

**Modelado de Datos:**

El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.

**Modelado de Proceso:**

Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir o recuperar un objeto de datos.

**Generación de Aplicaciones:**

El modelo asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.

**Pruebas y Entrega:**

Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

No todos los tipos de aplicaciones son apropiados para DRA. No es adecuado cuando los riesgos técnicos son altos. Esto ocurre cuando una nueva aplicación hace uso de tecnologías nuevas, o cuando el nuevo software requiere un alto grado de interoperatividad con programas de computadora ya existentes. DRA enfatiza el desarrollo de componentes de programas reutilizables.

**Conclusiones**

El modelo DRA básicamente lleva a cavo varios equipos de trabajo que siguen ciertas etapas, módulos, y objetivos del proyecto de manera simultanea que se aplican a la construcción de sistemas de información mas fáciles y modularizables para satisfacer las necesidades de los clientes, pudimos ver que nos es muy útil para aplicaciones que requieren adopción de nuevas tecnologías porque la curva de aprendizaje puede afectar el cronograma del proyecto. 

**Modelos Evolutivos** 

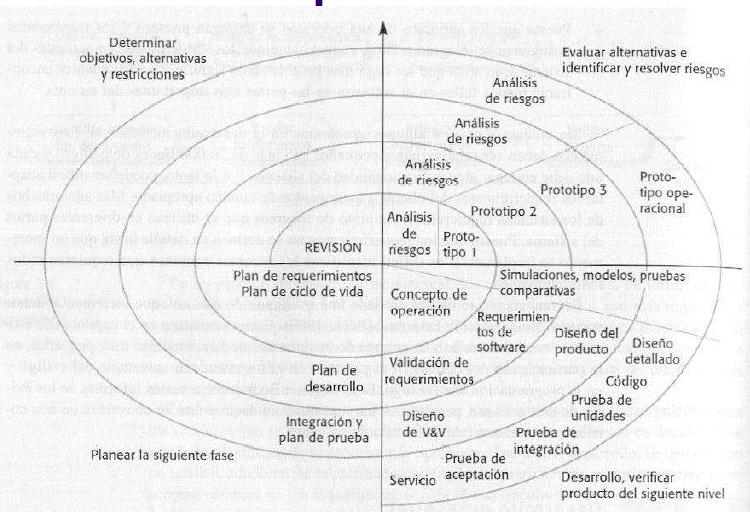
Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros del software desarrollar versiones cada vez mas completas del software.

El software evoluciona con el tiempo. Los requisitos del usuario y del producto suelen cambiar conforme se desarrolla el mismo. Las fechas de mercado y la competencia hacen que no sea posible esperar a poner en el mercado un producto absolutamente completo, por lo que se aconsejable introducir una versión funcional limitada de alguna forma para aliviar las presiones competitivas.

**Modelo Espiral**

Es un ciclo de vida de software definido por  Barry Boehm en 1988, utilizado mayormente en la ingeniería de software. Fue descrito por Boehm de la siguiente manera: ¨El modelo de desarrollo en espiral es un generador de modelo de proceso guiado por el riesgo que se emplea para conducir sistemas intensivos de ingeniería de software concurrente y a la vez con muchos usuarios¨. Las actividades que conforman este modelo forman una espiral, en la que cada bucle o interacciona representa un conjunto de actividades. Se tiene en cuenta fuertemente el riesgo que aparece a la hora de desarrollar software.

**Gráfico modelo Espiral**   
 

**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.   
   
   
Durante la primera vuelta alrededor de la espiral se definen los objetivos, las alternativas y las restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay una incertidumbre en los requisitos, se puede usar la creación de prototipos en el cuadrante de ingeniería para dar asistencia tanto al encargado de desarrollo como al cliente.   
   
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 bucle 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, cada vez más completa y, al final, al propio sistema operacional.   
   
El paradigma del modelo en espiral para la ingeniería de software es 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 reaccionar a los riesgos en cada nivel evolutivo. Utiliza la creación de prototipos como un mecanismo de reducción de riesgo, pero, lo que es más importante permite a quien lo desarrolla aplicar el enfoque de creación de prototipos en cualquier etapa de la evolución de prototipos.   
   
Características:

* Tiene y esta conformado en un enfoque cíclico para el crecimiento del grado de definición e implementación de un sistema, mientras que disminuye su grado de riesgo.
* Utiliza un conjunto de puntos de fijación para asegurar el compromiso que asume el usuario con las soluciones de sistema que sean factibles y totalmente satisfactorias.

**conclusiones**

Básicamente consiste en una serie de ciclos que comienzan desde el centro  que se repiten en forma de espiral, El Espiral puede verse como un modelo evolutivo que conjuga la naturaleza iterativa del modelo MCP con los aspectos controlados y sistemáticos del Modelo Cascada, con el agregado de gestión de riegos.   
   
este modelo no es tan usado por lo que no se tiene clara la medida de eficiencia de este modelo en un sistema de información, pero este modelo podemos ver que apto para el desarollo de sistemas operativos complejos, este modelo evolutivo tiene aspectos buenos ya que todo lo que hace este modelo es controlar y sistematizar las actividades

**Concurrente**

La gran mayoría de los modelos de desarrollo de software están ligados al tiempo, cuanto más se dure sería mejor o peor para el modelo. Este modelo de de desarrollo de software concurrente está ligado y dirigido primordialmente por las necesidades del usuario, las decisiones que se tomen  acabo y las tareas que realicen dichos usuarios.

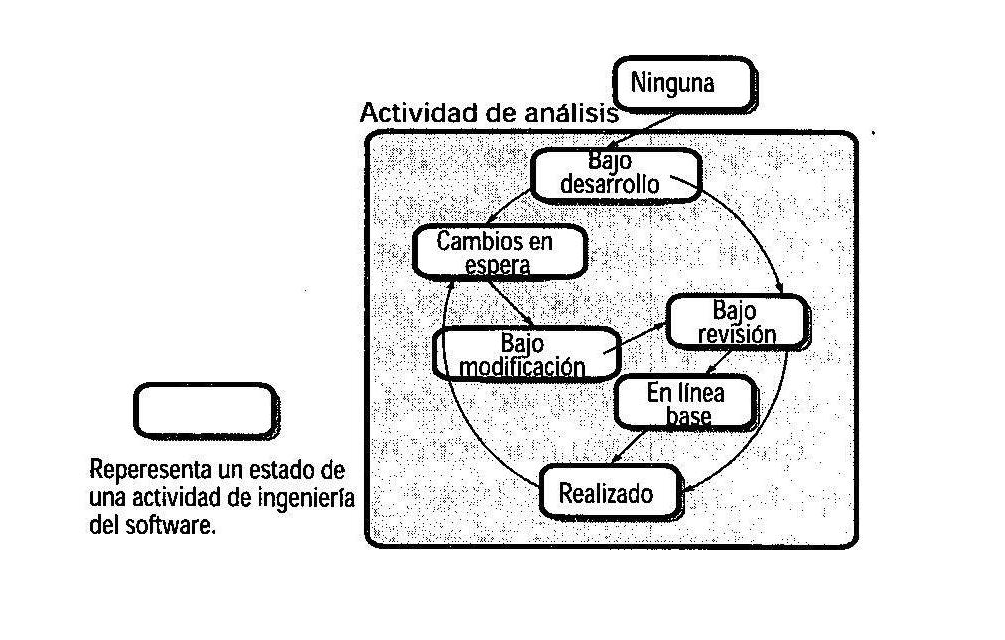
También define una serie de acontecimientos que se disparan a los estados de cada uno de las actividades realizadas.   
   
 

Fue creado por  Davis Sitaram, se puede representar en forma de esquema de una serie de actividades, técnicas tareas y estados asociados a ellas como ya lo habíamos mencionado. Tiene la capacidad de describir las múltiples actividades del software que están ocurriendo simultáneamente.

**Características:**

* Se expresa de manera esquematizada y organizada.
* Cada actividad lleva procesos concurrentes.
* Se aplica a la mayoría de tipos de desarrollo de software
* Es un modulo aplicable  para el cliente soñador.
* Esta dirigido básicamente y esencialmente a las necesidades del usuario.
* Es aplicable al cliente servidor.

**Gráfico del modelo concurrente**

**conclusiones**

  El modelo de proceso concurrente se puede sentar en forma de esquema como una serie de actividades técnicas importantes, tareas y estados asociados a ellas. Por ejemplo, la actividad de ingeniería definida para el modelo en espiral, se lleva acabo invocando las tareas siguientes: modelado de construcción de prototipos y/o análisis, especificación de requisitos y diseño.

      Este modelo define una serie de acontecimientos que disparan transiciones de estado a estado para cada una de las actividades del proyecto, utilizando para ellos el paradigma de desarrollo de aplicaciones cliente/servidor.   
 

**Incremental**

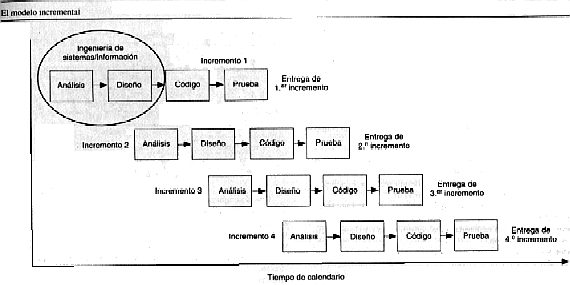
Es un modelo basado en varios ciclos en Cascada retroalimentados aplicados consecutivamente. Combina los elementos del MLS con la filosofía con la filosofía interactiva de construcción de prototipos.

Fue propuesto por Mills en 1980. En una visión genérica, el proceso se divide en 4 partes: Análisis, Diseño, Código y Prueba. Se usa el principio de trabajo en cadena o “Pipeline”, utilizado en muchas otras formas de programación.   
   
 

**Características:**

* Se evitan proyectos largos y se entrega “algo de valor” a los usuarios con cierta       frecuencia.
* El usuario se involucre más.
* Difícil de evaluar el costo total.
* Difícil de aplicar a los sistemas transaccionales que tienden a ser integrados y a operar como un todo.
* Requiere gestores experimentados.
* Los errores en los requisitos se detectan tarde.
* El resultado puede ser muy positivo.

**Gráfico del modelo incremental**



**conclusiones**

     El modelo incremental es una unión de las mejores funcionalidades del modelo de cascada y del modelo de prototipos. A medida que se presenta un prototipo se produce un “incremento”, que es una iteración del proceso anterior pero aplicando las experiencias aprendidas del proceso anterior. A diferencia del modelo de prototipos, los prototipos de este modelo están orientados a ser operacionales en cada incremento y no ser solo una “previa” de cómo sería el sistema en su versión final.

**Modelos de métodos formales**   
 

El modelo de métodos formales acompaña a un conjunto de actividades que conducen a la especificación matemática del software de computadora. Los métodos formales permiten que un ingeniero del software especifique, desarrolle y verifique un sistema basado en computadora aplicando una notación rigurosa y matemática.

La ambigüedad, lo incompleto y la inconsistencia se descubren y se corrigen más fácilmente, no mediante una revisión a propósito para el caso, sino mediante la aplicación del análisis matemático. Cuando se utilizan métodos formales durante el diseño, sirven como base para la verificación de programas y por consiguiente permiten que el ingeniero del software descubra y corrija errores que no se pudieron detectar de otra manera.

Aunque todavía no hay un enfoque establecido, los modelos de métodos formales ofrecen la promesa de un software libre de defectos. Sin embargo, se ha hablado de una gran preocupación sobre su aplicabilidad en un entorno de gestión

**Gráfico de los modelos de métodos formales**

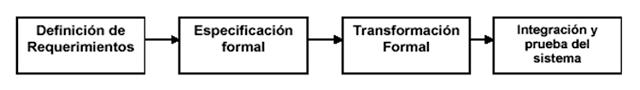
**Modelos de métodos formales**   
 

El modelo de métodos formales acompaña a un conjunto de actividades que conducen a la especificación matemática del software de computadora. Los métodos formales permiten que un ingeniero del software especifique, desarrolle y verifique un sistema basado en computadora aplicando una notación rigurosa y matemática.

La ambigüedad, lo incompleto y la inconsistencia se descubren y se corrigen más fácilmente, no mediante una revisión a propósito para el caso, sino mediante la aplicación del análisis matemático. Cuando se utilizan métodos formales durante el diseño, sirven como base para la verificación de programas y por consiguiente permiten que el ingeniero del software descubra y corrija errores que no se pudieron detectar de otra manera.

Aunque todavía no hay un enfoque establecido, los modelos de métodos formales ofrecen la promesa de un software libre de defectos. Sin embargo, se ha hablado de una gran preocupación sobre su aplicabilidad en un entorno de gestión

**Gráfico de los modelos de métodos formales**



**Ventajas e Inconvenientes**

* Si bien los métodos formales constituyen un acercamiento alternativo a las metodologías de desarrollo de software, existen un numero de diferencias signiﬁcativas que deben de ser consideradas
* Ofrece un fundamento para entornos de especificación que dan lugar a modelos de análisis más completos, consistentes y carentes de ambigüedad, que aquellos que se producen empleando métodos convencionales u orientados a objetos.
* al pensar en instaurar los métodos formales en un equipo de desarrollo de software

**Conclusiones**

Los métodos formales se centra fundamentalmente en las funciones y los datos. La especificación mediante métodos formales es más difícil de aprender que otros métodos de análisis. Las especificaciones formales se pueden estudiar matemáticamente, mientras que las informales no pueden estudiarse de esta manera. 

|  |  |  |
| --- | --- | --- |
| **CUADRO COMPARATIVO DE LOS MODELOS DE PROCESO DEL SOFTWARE** | | |
| MODELOS DE PROCESO DEL SOFTWARE | Ventajas | Desventajas |
| [**Modelo Lineal Secuencia "Cascada"**](http://primermodelo.blogspot.com/) | * Se debe tener en cuenta que fue el primer modelo empleado, y por lo tanto es mejor que ninguno. * Facilita la gestión del desarrollo. | * En general, establecer todos los requisitos al principio del proceso de desarrollo es un mito inalcanzable, Los usuarios no pueden imaginarse lo que quieren hasta que no ven un sistema funcionando. * Los requisitos no se pueden congelar mientras dura el desarrollo. El mercado cambia, todo cambia. * El usuario debe esperar mucho tiempo hasta ver los resultados * Los errores de análisis y diseño son costosos de eliminar, y se propagan a las fases siguientes con un efecto conocido como bola de nieve. * Se genera mucho mantenimiento inicial debido al período de congelación de requisitos y éste recae, en su mayor parte |
| [**Modelo De Construcción De Prototipos**](http://segundomodelo.blogspot.com/) | * Permite la retroalimentación por parte del usuario. * Desarrollo rápido. * El usuario se siente parte del grupo * También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina. * No modifica el flujo del ciclo de vida. * Reduce el riesgo de construir productos que no satisfagan las necesidades de los usuarios. * Reduce costos y aumenta la probabilidad de éxito. * Exige disponer de las herramientas adecuadas. * No presenta calidad ni robustez. | * El desarrollador debe dar forma prematuramente a un sistema, incluso antes de comprender de manera básica el problema y su funcionamiento. * El usuario puede creer que un prototipo es un software final. * Debe ser un sistema con el que se pueda experimentar * Debe desarrollarse rápidamente * Énfasis en la interfaz de usuario * Equipo de desarrollo reducido * Herramientas y lenguajes adecuadas |
| [**Modelo De Desarrollo Rápido De Aplicaciones (DRA)**](http://tercermodelo.blogspot.com/) | * Es muy rápido. * Permite trabajar en él a varias personas a la vez. * Enfatiza ciclos de desarrollo extremadamente cortos. * Se asegura de que el producto entregado cumple las necesidades del cliente. | * El enfoque DRA tiene inconvenientes para proyectos grandes aunque por escalas, necesita suficientes recursos humanos para crear el numero correcto de equipos. * Si los desarrolladores y clientes no se comprenden con las actividades necesarias para completar el sistema, los proyectos fallarán. * El DRA sería inapropiado cuando los riesgos técnicos son altos. * No todos los tipos de aplicaciones son apropiados para DRA. Si un sistema no se puede modulizar adecuadamente. La construcción de los componentes necesarios para DRA será problemático. Si está en juego el alto rendimiento, y se va a conseguir el rendimiento convirtiendo interfaces en componentes de sistema, el enfoque DRA puede que no funcione. |
| [**Modelos Evolutivos**](http://cuartomodelo.blogspot.com/)   |  | | --- | | Modelo espiral | | Modelo concurrente | | Modelo incremental | | |  | | --- | | Modelo espiral  El análisis del riesgo se hace de forma explícita y clara. Une los mejores elementos de los restantes modelos.   * Reduce riesgos del proyecto * Incorpora objetivos de calidad * Integra el desarrollo con el mantenimiento, etc.   Además es posible tener en cuenta mejoras y nuevos requerimientos sin romper con la metodología, ya que este ciclo de vida no es rígido ni estático. | | Modelo concurrente   * Es excelente para proyectos en los que se conforman grupos de trabajo. * Proporciona una imagen exacta del estado actual de un proyecto * No restringe el proyecto a una secuencia de sucesos. | | Modelo incremental   * Con un paradigma incremental se reduce el tiempo de desarrollo inicial, ya que se implementa la funcionalidad parcial. * También provee un impacto ventajoso frente al cliente, que es la entrega temprana de partes operativas del Software. * El modelo proporciona todas las ventajas del modelo en cascada realimentado, reduciendo sus desventajas sólo al ámbito de cada incremento. * Permite entregar al cliente un producto más rápido en comparación del modelo de cascada. * Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos. * Por su versatilidad requiere de una planeación cuidadosa tanto a nivel administrativo como técnico. | | |  | | --- | | Modelo espiral   * Genera mucho tiempo en el desarrollo del sistema * Modelo costoso * Requiere experiencia en la identificación de riesgos | | Modelo concurrente   * Si no se dan las condiciones específicas no se puede aplicar. * Si no existe grupo de trabajo  no se puede  trabajar  en este método. * Todas las actividades de red  existen simultáneamente con otras. * Los sucesos generados dentro de una actividad, o en algún otro lado de la red de         actividad, inician las transiciones entre los estados de otra actividad. | | Modelo incremental   * El modelo Incremental no es recomendable para casos de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos.       Requiere de mucha planeación, tanto administrativa como técnica. * Requiere de metas claras para conocer el estado de los proyectos. | |
| [**Modelos De Métodos Formales**](http://quintomodelos.blogspot.com/2013/05/modelos-de-el-modelo-de-metodos.html) | * Se comprende mejor el sistema. * La comunicación con el cliente mejora ya que se dispone de una descripción clara y no ambigua de los requisitos del usuario. * El sistema se describe de manera más precisa. * El sistema se asegura matemáticamente que es correcto según las especificaciones. * Mayor calidad software respecto al cumplimiento de las especificaciones. * Mayor productividad | * El desarrollo de herramientas que apoyen la aplicación de métodos formales es complicado y los programas resultantes son incómodos para los usuarios. * Los investigadores por lo general no conocen la realidad industrial. * Es escasa la colaboración entre la industria y el mundo académico, que en ocasiones se muestra demasiado dogmático. * Se considera que la aplicación de métodos formales encarece los productos y ralentiza su desarrollo. |

**REFERENCIAS WEB**

1. Nelson Medinilla Martínez, Facultad de Informática, Universidad Politécnica de Madrid, “Análisis y selección de estrategias de desarrollo de software” [En línea], Madrid España [Consulta: Mayo de 2006], <http://is.ls.fi.upm.es/udis/docencia/proyecto/docs/  
   estrategias.pdf>
2. Carolina Zibert, “Ciclos de vida de Ingeniería de Software” [En línea], Caracas Venezuela [Consulta: Abril de 2006],<carolina.terna.net/ingsw2/Datos/Cascada-ModeloV.doc>
3. Tesis doctorales en Zarza, “Ingeniería de Software” [En línea], España [Consulta: Mayo de 2006], <http://www.tdx.cesca.es/TESIS\_UPC/AVAILABLE/TDX-0716102-102210//05Capitulo05.pdf>
4. Mundo Geek, “Ciclos de vida del software” [En línea], [Consulta: Mayo de 2006], <<http://mundogeek.net/archivos/2004/05/20>>
5. Wikipedia Foundation Inc, EUA Desarrollo en Cascada [En línea], St. Petersburg [Consulta: Mayo de 2006], <<http://es.wikipedia.org/wiki/Modelo_en_cascada>>
6. Instituto Tecnológico de la paz, “Análisis y diseño de sistemas Modelo de Espiral” [En línea], México [Consulta: Mayo de 2006], <http://www.itlp.edu.mx/publica/tutoriales/  
   analisis/index.htm>
7. Copyright **©** Programación en Castellano, [En línea], España [Consulta: Mayo de 2006], <http://www.programacion.com/blogs/46\_aprendiendostruts>
8. Zavala R. 2000. Diseño de un Sistema de Información Geográfica sobre Internet. Tesis de Maestría en Ciencias de la Computación. Universidad Autónoma Metropolitana-Azcapotzalco., “La Ingeniería del Software” [En línea], [Consulta: Abril de 2006], México, D.F. <http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html#fig2>
9. Juan Pavón Maestras, Universidad Complutense de Madrid, “El proceso unificado” [En línea], Madrid España [Consulta: Abril de 2006], <http://www.fdi.ucm.es/profesor/  
   jpavon/is2/03ProcesoUnificado.pdf>
10. Carlos Reynoso, Universidad de Buenos Aires, Arquitectura de Software [En línea] Buenos Aires Argentina [Consulta: Mayo de 2006], <http://www.microsoft.com/spanish/  
    msdn/arquitectura/roadmap\_arq/heterodox.asp>
11. Grupo de Investigación Kybele, Universidad Rey Juan Carlos, “Fases del proceso unificado” [En linea], Madrid España [Consulta: Abril de 2006], <http://kybele.escet.urjc.es/Documentos/ISI/Fases%20del%20Proceso%20Unificado.pdf>
12. Wikipedia Foundation Inc, EUA “Proceso Unificado de Rational” [En línea], St. Petersburg [Consulta: Abril de 2006],http://es.wikipedia.org/wiki/RUP
13. Mike Grasso, University of Maryland, “The Personal Software Process” [En línea], Maryland EUA [Consulta: Mayo de 2006] <http://www.csee.umbc.edu/~mikeg/  
    cmsc645/se\_psp.pdf>
14. Gustavo Adolfo Ramírez González, Universidad del Caucan Popayán, “Proceso Unificado para desarrollo de Software”, Colombia [Consulta: Junio de 2006], <http://atenea.ucauca.edu.co/~gramirez/archivos/AnotacionesRUP.pdf>
15. A.U.S. GustavoTorossi, “El Proceso Unificado de Desarrollo de Software” [En línea], Provincia de Chaco Republica de Argentina [Consulta: Junio de 2006] <http://www.chaco.gov.ar/UTN/disenodesistemas/apuntes/oo/ApunteRUP.pdf>
16. Ivar Jacobson, “Applying UML in The Unified Process” [En linea], [Consulta:Enero de 2006], <http://www.jeckle.de/files/uniproc.pdf>