TALLER DISEÑO DE SISTEMA DE INFORMACIÒN

HERMAN COLLAZOS CASTAÑEDA

DISEÑO DE SISTEMAS DE INFORMACION

CARLOS ALBERTO LONDOÑO

INGENIERO DE SISTEMAS

CORPORACIÒN DE ESTUDIOS TECNOLOGICOS DEL NORTE DEL VALLE

CIENCIAS INFORMÁTICAS, TECNOLÓGICAS E INGENIERÍA

[TECNOLOGIA EN SISTEMAS DE INFORMACION](http://www.cnotas.cotecnova.edu.co/servicios/servicios_est2.php)

CARTAGO VALLE

2017

¿QUE ES UN PROCESOS DE SOFTWARE?

un proceso no es una prescripción rígida de cómo elaborar software de cómputo. Por el contrario, es un enfoque adaptable que permite que las personas que hacen el trabajo (el equipo de software) busquen y elijan el conjunto apropiado de acciones y tareas para el trabajo. Se busca siempre entregar el software en forma oportuna y con calidad suficiente para satisfacer a quiénes patrocinaron su creación y a aquellos que lo usarán.

La estructura del proceso establece el fundamento para el proceso completo de la ingeniería de software por medio de la identificación de un número pequeño de actividades estructurales que sean aplicables a todos los proyectos de software, sin importar su tamaño o complejidad. Además, la estructura del proceso incluye un conjunto de actividades sombrilla que son aplicables a través de todo el proceso de software.

CONSULTAR LOS DIFERENTES MODELOS DE PROCESO DE SOFTWARE CON CADA UNO DE SUS DIAGRAMAS, CARACTERISTICAS, CLASIFICACION, HERRAMIENTAS, FASES Y DEFINICION DE COMPONENTES

MODELO LINEAL SECUENCIAL

También llamado "Ciclo de vida básico " tiene su origen en el "Modelo de cascada" basado en el modelo en cascada de Winston Royce, 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 MLS tiene las siguientes actividades:

Mantenimiento

Ingeniería y Análisis del Sistema

Análisis de los Requisitos

Diseño

Codificación

Prueba

Ingeniería y Análisis del Sistema: Debido a que el software es siempre parte de un Sistema mayor el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software.

Análisis de los requisitos del software: El proceso de recopilación de los requisitos se centra e intensifica especialmente en el software. El ingeniero de software (Analistas) debe comprender el ámbito de la información del software, así como la función, el rendimiento y las interfaces requeridas.

Diseño: El diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación.Se dividen en:

Diseño de Alto Nivel o Arquitectónico

Diseño Detallado

Codificación: El diseño debe traducirse en una forma legible para la máquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada la codificación puede realizarse mecánicamente.

Prueba: Una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren. Existen varios tipos de pruebas:

Pruebas de unidad

Pruebas de integración

Prueba de sistema

Prueba de aceptación

Mantenimiento: El software sufrirá cambios después de que se entrega al cliente. Los cambios ocurrirán debido a que hayan encontrado errores, a que el software deba adaptarse a cambios del entorno externo (sistema operativo o dispositivos periféricos), o debido a que el cliente requiera ampliaciones funcionales o del rendimiento. Los tipos de mantenimiento son:

Mantenimiento Preventivo o Perfectivo

Mantenimiento Correctivo

Mantenimiento Evolutivo

CARACTERISTICAS:

Es una visión del proceso de desarrollo de software como una sucesión de etapas que producen productos intermedios.

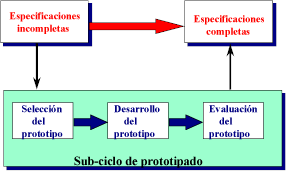
Para que el proyecto tenga éxito deben desarrollarse todas las fases.

Las fases continúan hasta que los objetivos se han cumplido.

Si se cambia el orden de las fases, el producto final será de inferior calidad

MODELO DE PROPOTIPOS

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



¿Qué es la construcción de prototipos?

Es un proceso que facilita al programador la creación de un modelo de software a construir.

A menudo un cliente define un conjunto de objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida. 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 humana – máquina, entonces es en este caso cuando utilizamos la construcción de prototipos.

Al usar prototipos, las etapas de ciclo de vida clásico pueden modificarlas de la siguiente manera:

Análisis de requisito del sistema

Análisis de requisito del software

Diseño, desarrollo e implementación del prototipo.

Prueba del prototipo

Refinamiento interactivo del prototipo

Refinamiento de las especificaciones del prototipo

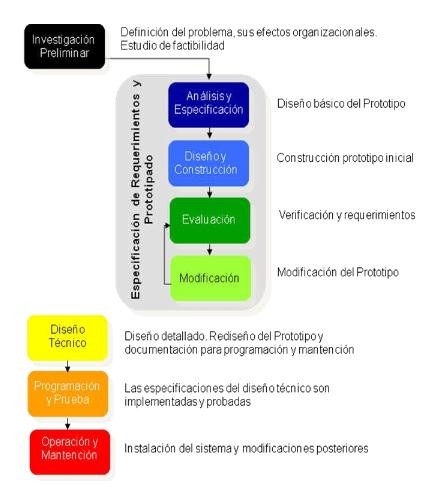
Diseño e implementación del sistema final

Explotación y mantenimiento.

Si bien este modelo de prototipos es fácilmente modificable y ampliable también es muy usado, en muchos casos pueden usarse prototipos descartables para esclarecer aquellos aspectos del sistema que no se comprendan bien.

SELECCIÓN DEL MODELO DE PROTOTIPO

Este modelo es adecuado cuando se desea desarrollar programas didácticos computarizados de una manera más abierta de modo que el cliente en este caso el profesor realice los refinamientos o las aportaciones necesarias. Todos los requerimientos no son conocidos al principio. Coloca énfasis en la etapa de Especificación de Requerimientos a través de la construcción de Prototipos que aproximan al usuario a la idea final del sistema con el propósito de poder clarificar los requerimientos. Los usuarios lo prueban y añaden requerimientos. Se hace una implantación parcial del sistema y se prueba. Se utiliza en sistemas complejos.

ETAPAS DEL METODO DE PROTOTIPOS:

MODELO DRA

El desarrollo rápido de aplicaciones o RAD (Rapid Application Development) es un proceso de desarrollo de software, desarrollado inicialmente por James Martin en 1980. El método comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

El Desarrollo Rápido de Aplicaciones (DRA) 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.

Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque DRA comprende las siguientes fases:

Modelado de Gestión: Se diseña en base a al flujo de información y con respecto a este flujo se deben responder las siguientes preguntas:

¿Que información se recibe?

¿Que información se genera?

¿De dónde viene?

¿Hacia dónde va?

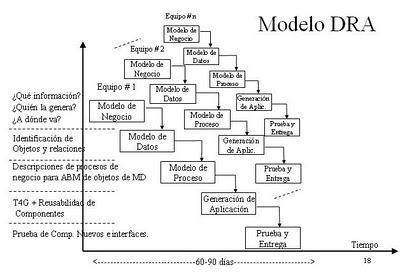
¿Quién procesara luego la información?

Modelado de Datos: Se diseña la estructura de datos con sus objetos y sus relaciones para que contenga la información del modelado de gestión.

Modelado de Proceso: Se aplican funciones sobre los datos del punto anterior, básicamente se diseñan procesos que crean, modifican, eliminan o recuperar objetos.

Generación de Aplicaciones: Se basa en el uso de técnicas de cuarta generación en donde lo que se utiliza para programar no son lenguajes propiamente dichos si no componentes anteriores que son reutilizables, se da el caso de que deben desarrollarse componentes nuevos y entonces se diseñan para que también puedan ser utilizados por futuros desarrollos basados en este modelo.

Pruebas y entrega: Como muchos componentes son reutilizados ya se han probado se reduce gran parte del proceso de prueba. Queda probar los componentes nuevos y asegurarse de que la comunicación entre ellos se adecuada.



CARACTERISTICAS:

Modelo lineal secuencial orientado a un ciclo rápido de desarrollo.

Basado en el empleo de componentes para poder entregar un modelo totalmente operativo en un corto periodo de tiempo.

Es fundamental poder modular la aplicación pata que cada equipo pueda trabajar en diferentes modelos.

RAD TIENDE A FUNCIONAR CUANDO:

La aplicación funcionará de manera independiente.

Se pueden usar mayormente bibliotecas existentes.

Desempeño no crítico.

Distribución limitada, interna o vertical.

Alcance del proyecto limitado.

Confiabilidad no crítica.

El sistema puede dividirse en muchos módulos independientes.

El producto está dirigido a un mercado altamente especializado.

El proyecto cuenta con fuertes limitantes de tiempos parciales

RAD TIENDE A FALLAR CUANDO:

La aplicación debe inter operar con sistemas existentes.

Existen pocos componentes reutilizables.

Alto desempeño crítico.

El desarrollo no puede aprovechar herramientas de alto nivel.

Distribución amplia, horizontal o masiva.

Métodos RAD para desarrollar sistemas operativos (confiabilidad demasiado alta) o juegos (desempeño demasiado alto).

Riesgos técnicos de tecnología de punta.

El producto pone en riesgo la misión o la vida.

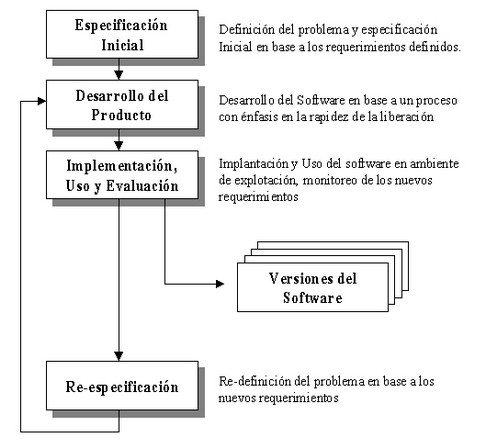
El producto no puede ser modularizado.

MODELOS EVOLUTIVOS

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 debe introducir una versión funcional limitada de alguna forma para aliviar las presiones competitivas. En esas u otras situaciones similares los desarrolladores necesitan modelos de progreso que estén diseñados para acomodarse a una evolución temporal o progresiva, donde los requisitos centrales son conocidos de antemano, aunque no estén bien definidos a nivel detalle.

Los evolutivos son modelos iterativos, permiten desarrollar versiones cada vez más completas y complejas, hasta llegar al objetivo final deseado; incluso evolucionar más allá, durante la fase de operación. Los modelos “Iterativo Incremental” y “Espiral” (entre otros) son dos de los más conocidos y utilizados del tipo evolutivo.

Modelos de desarrollos que a diferencia del de Prototipos busca reemplazar el viejo sistema con un nuevo que tendría la propiedad de satisfacer los nuevos requerimientos lo más rápido posible. Los modelos evolutivos asumen que los requerimientos están sujetos a cambios continuos y que la estrategia para enfrentar aquello pasa por un reflejo, también continuo, de aquellos cambios.



MODELO ITERATIVO INCREMENTAL

El modelo incremental combina elementos del modelo cascado (aplicado repetidamente) así como la filosofía iterativa del prototipado. La parte inicial es el núcleo del producto (es la parte más importante).Una nueva versión del producto surge cuando nuevas características han sido implementadas a medida que han sido sugeridas por el usuario.

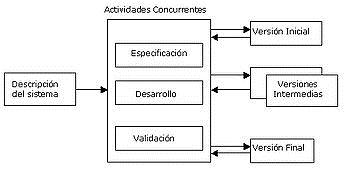
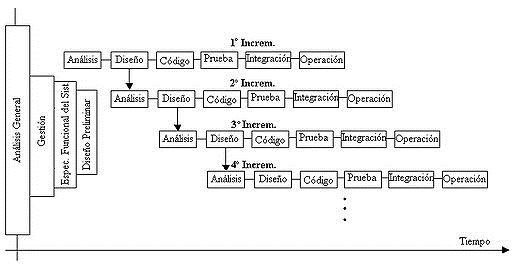
La Descripción del Sistema es esencial para especificar y confeccionar los distintos incrementos hasta llegar al Producto global y final. Las actividades concurrentes (Especificación, Desarrollo y Validación) sintetizan el desarrollo pormenorizado de los incrementos.

Diagrama genérico del desarrollo evolutivo incremental

El desarrollo evolutivo incremental permite la entrega de versiones parciales a medida que se va construyendo el producto final. Es decir, a medida que cada incremento definido llega a su etapa de operación y mantenimiento. Cada versión emitida incorpora a los anteriores incrementos las funcionalidades y requisitos que fueron analizados como necesarios.

En la siguiente imagen se muestra un refino del diagrama previo, bajo un esquema temporal, para obtener finalmente el esquema del Modelo de ciclo de vida Iterativo Incremental, con sus actividades genéricas asociadas. Aquí se observa claramente cada ciclo cascada que es aplicado para la obtención de un incremento; estos últimos se van integrando para obtener el producto final completo. Cada incremento es un ciclo Cascada Realimentado, aunque, por simplicidad, en la imagen se muestra como secuencial puro.

Modelo iterativo incremental para el ciclo de vida del software

Se observa que existen actividades de desarrollo (para cada incremento) que son realizadas en paralelo o concurrentemente así, por ejemplo, en la figura, mientras se realiza el diseño detalle del primer incremento ya se está realizando en análisis del segundo.

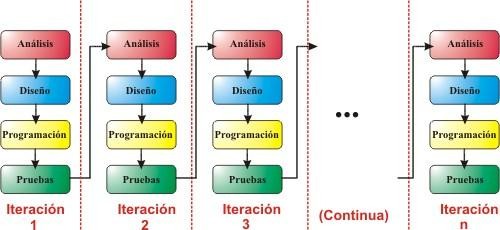
El modelo iterativo incremental es sólo esquemático, un incremento no necesariamente se iniciará durante la fase de diseño del anterior, puede ser posterior (incluso antes), en cualquier tiempo de la etapa previa. Cada incremento concluye con la actividad de “Operación y Mantenimiento” (indicada "Operación" en la figura), que es donde se produce la entrega del producto parcial al cliente. El momento de inicio de cada incremento es dependiente de varios factores: tipo de sistema; independencia o

dependencia entre incrementos (dos de ellos totalmente independientes pueden ser fácilmente iniciados al mismo tiempo si se dispone de personal suficiente); capacidad y cantidad de profesionales involucrados en el desarrollo; etc.

Como se muestra en la anterior imagen, se aplican secuencias Cascada en forma escalonada, mientras progresa el tiempo calendario. Cada secuencia lineal o Cascada produce un incremento y a menudo el primer incremento es un sistema básico, con muchas funciones suplementarias (conocidas o no) sin entregar.

El cliente utiliza inicialmente ese sistema básico intertanto, el resultado de su uso y evaluación puede aportar al plan para el desarrollo del/los siguientes incrementos (o versiones). Además, también aportan a ese plan otros factores, como lo es la priorización (mayor o menor urgencia en la necesidad de cada incremento) y la dependencia entre incrementos (o independencia).

Luego de cada integración se entrega un producto con mayor funcionalidad que el previo. El proceso se repite hasta alcanzar el software final completo.

Siendo iterativo, con el modelo incremental se entrega un producto parcial pero completamente operacional en cada incremento, y no una parte que sea usada para reajustar los requerimientos (como si ocurre en el [modelo de construcción de](http://es.wikipedia.org/wiki/Modelo_de_prototipos) [prototipos](http://es.wikipedia.org/wiki/Modelo_de_prototipos)).

La principal característica de estos modelos es que permite crear cada vez versiones más completas de software, para esto construimos versiones sucesivas de un producto. Se crea una primera versión que es utilizada por el usuario donde se provee retroalimentación al desarrollador, y según los requerimientos especificados de este usuario se crea una segunda versión.

MODELO ESPIRAL

Es un modelo evolutivo inicialmente por [Barry Boehm](http://es.wikipedia.org/wiki/Barry_Boehm) que conjuga la naturaleza iterativa del modelo [MCP](http://es.wikipedia.org/wiki/Modelo_de_prototipos) con los aspectos controlados y sistemáticos del Modelo Cascada. Proporciona potencial para desarrollo rápido de versiones incrementales más completas del sistema diseñado.

En el modelo espiral, el software se desarrolla en una serie de versiones increméntales. Durante las primeras iteraciones. La versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas de ingeniería del sistema.

El modelo en espiral se divide en un número de actividades estructurales, también llamadas regiones de tareas. Generalmente, existen entre tres y seis regiones de tareas.

 Comunicación con el cliente: las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.

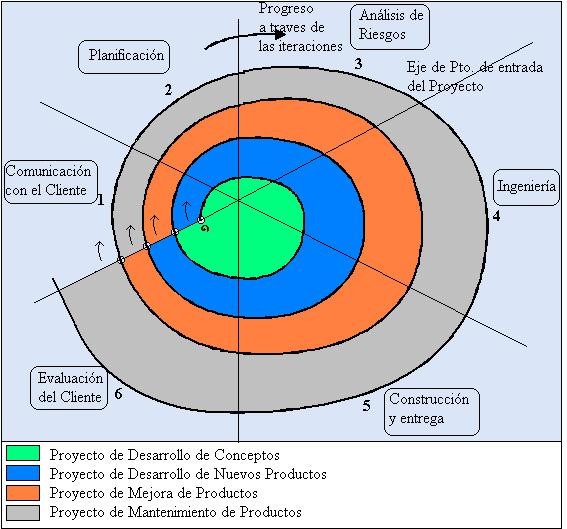
 Planificación: las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto. Son todos los requerimientos.

 Análisis de riesgos: las tareas requeridas para evaluar riesgos técnicos y otras informaciones relacionadas con el proyecto.

 Ingeniería: las tareas requeridas para construir una o más representaciones de la aplicación.

 Construcción y adaptación: las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario.

 Evaluación el cliente: las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementación durante la etapa de instalación.



Modelo espiral para el ciclo de vida del software

Cuando empieza el proceso evolutivo, el equipo de ingeniería del software gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. El primer circuito de la espiral produce el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones más sofisticadas del software. Cada paso de la región de planificación produce ajustes en el plan del proyecto. El coste y la planificación se ajustan según la reacción ante la evolución del cliente.

CARACTERISTICAS:

Es también al igual que el anterior un modelo evolutivo que combina el modelo clásico con el diseño de prototipos.

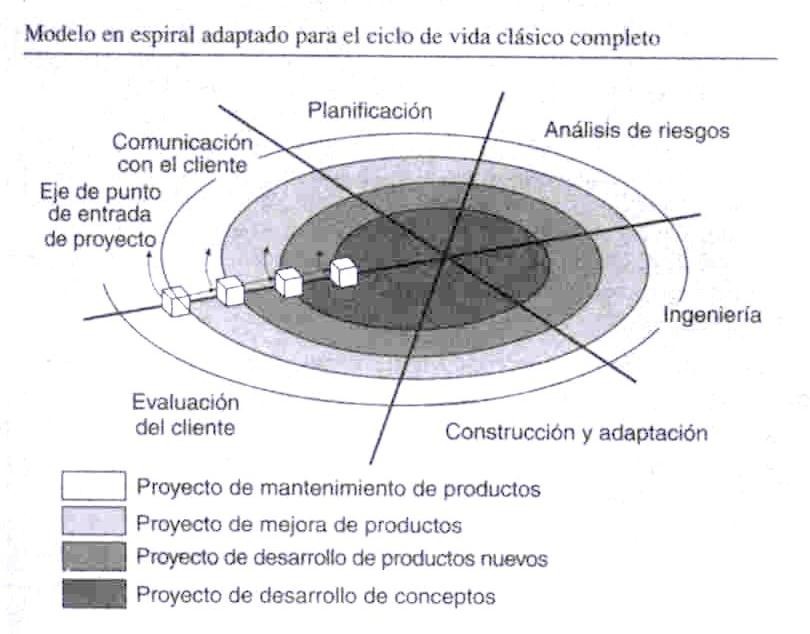
Incluye la etapa de análisis de riesgos, no incluida anteriormente.

Es ideal para crear productos con diferentes versiones mejoradas como se hace con los softwares modernos de microcomputadoras.

La ingeniería puede desarrollarse a través del ciclo de vida clásico o el de construcción de prototipos.

Este es el enfoque más realista actualmente.

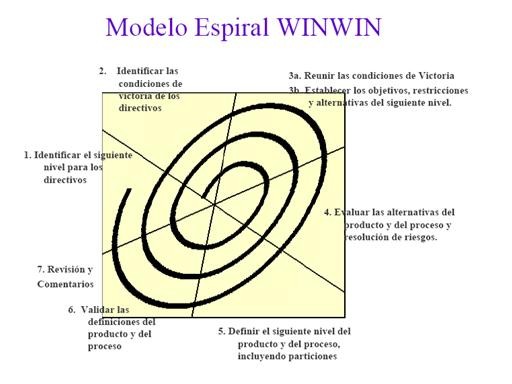
La siguiente figura define un eje de punto de entrada en el proyecto. Cada uno de los cubos situados a lo largo del eje representa el punto de arranque para un tipo diferente de proyecto. Un proyecto de desarrollo de conceptos comienza en el centro de la espiral y continuara hasta que se completa el desarrollo del concepto. Si el concepto se va a desarrollar dentro de un producto real, el proceso procede a través del cubo siguiente y se inicia un nuevo proyecto de desarrollo.



MODELO ESPIRAL WIN & WIN

Una variante interesante del Modelo Espiral. El Modelo Espiral previo (clásico) sugiere la comunicación con el cliente para fijar los requisitos, en que simplemente se pregunta al cliente qué necesita y él proporciona la información para continuar; pero esto es en un contexto ideal que rara vez ocurre. Normalmente cliente y desarrollador entran en una negociación, se negocia coste frente a funcionalidad, rendimiento, calidad, etc.

"Es así que la obtención de requisitos requiere una negociación, que tiene éxito cuando ambas partes ganan".



Las mejores negociaciones se fuerzan en obtener "Victoria & Victoria" (Win & Win), es decir que el cliente gane obteniendo el producto que lo satisfaga, y el desarrollador también gane consiguiendo presupuesto y fecha de entrega realista. Evidentemente, este modelo requiere fuertes habilidades de negociación.

El modelo Win-Win define un conjunto de actividades de negociación al principio de cada paso alrededor de la espiral; se definen las siguientes actividades:

Identificación del sistema o subsistemas clave de los directivos (\*) (saber qué quieren).

Determinación de "condiciones de victoria" de los directivos (saber qué necesitan y los satisface)

Negociación de las condiciones "victoria" de los directivos para obtener condiciones "Victoria & Victoria" (negociar para que ambos ganen).

(\*) Directivo: Cliente escogido con interés directo en el producto, que puede ser premiado por la organización si tiene éxito o criticado si no.

El modelo Win & Win hace énfasis en la negociación inicial, también introduce 3 hitos en el proceso llamados "puntos de fijación", que ayudan a establecer la completitud de un ciclo de la espiral, y proporcionan hitos de decisión antes de continuar el proyecto de desarrollo del software.

MODELO BASADO EN COMPONENTES

El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo espiral. Es evolutivo por naturaleza y exige un enfoque interactivo para la creación del software. Sin embargo, el modelo de desarrollo basado en componentes configura aplicaciones desde componentes preparados de software (clases).

El modelo de desarrollo basado en componentes conduce ala reutilización del software, y la reutilización proporciona beneficios a los ingenieros de software. Según estudios de reutilización, QSM Associates, Inc. Informa que el ensamblaje de componentes lleva a una reducción del 70 % del ciclo de desarrollo un 84% del coste del proyecto y un índice de productividad del 26.2. No hay duda que el ensamblaje de componentes proporciona ventajas significativas para los ingenieros del software.

El proceso unificado de desarrollo de software representa un número de modelos de desarrollo basado en componentes que han sido propuestos en la industria. El lenguaje de modelado unificado define los componentes. Utilizando una combinación del desarrollo incremental e interactivo, el proceso unificado define la función del sistema aplicando un enfoque basado en escenarios.

El desarrollo de software basado en componentes se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software.

Un modelo basado en componentes define la arquitectura básica de un componente, especificando la estructura de sus interfaces y los mecanismos por los cuales interactúan con su contenedor y los demás componentes. El modelo basado en componentes proporciona los lineamientos para crear e implementar componentes que puedan funcionar conjuntamente para constituir

una aplicación mayor. Los diseñadores de aplicaciones pueden combinar componentes de diferentes programadores o proveedores para construir una aplicación. [Thomas 98]

Una vez que la mayor parte de los aspectos funcionales de esta disciplina comienzan a estar bien definidos, la atención de la comunidad científica comienza a centrarse en los aspectos extra funcionales y de calidad, como un paso hacia una verdadera ingeniería. En este artículo se discuten precisamente los aspectos de calidad relativos a los componentes software y a las aplicaciones que con ellos se construyen, con especial énfasis en los estándares internacionales que los definen y regulan, y en los problemas que se plantean en este tipo de entornos.



CARACTERISTICAS:

Evolutivo por naturaleza

Exige un enfoque iterativo

Notación de componentes

Diagrama de componentes

Interfaces

Componentes y nodos

Restricciones

REALIZAR UN CUADRO COMPARATIVO CON CADA UNO DE LOS MODELOS DE PROCESOS DE SOFTWARE DONDE SE EVIDENCIA: VENTAJAS Y DESVENTAJAS

|  |  |  |
| --- | --- | --- |
| MODELOS DE PROCESOS DE SOFTWARE | Ventajas | Desventajas |
| MODELO LINEAL SECUENCIAL | Se tiene todo bien organizado y no se mezclan las fases.  Modelo y planificación fácil y sencillos.  Este método radica en su sencillez ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.  Es perfecto para proyectos que son rígidos, y además donde se especifiquen muy bien los requerimientos y se conozca muy bien la herramienta a utilizar.  Los usuarios lo pueden comprender fácilmente | 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. El ciclo de vida clásico lo requiere y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos productos.  El cliente debe tener paciencia. Hasta llegar a las etapas finales del proyecto, no estará disponible una versión operativa del programa. Un error importante no detectado hasta que el programa esté funcionando puede ser desastroso.  Alto riesgo en sistemas nuevos debido a problemas en las especificaciones y en el diseño.  Los responsables del desarrollo de software siempre se retrasan innecesariamente |
| MODELO DE PROPOTIPOS | Reducción de incertidumbre y del riesgo, reducción de tiempo y de costos.  Incrementos en la aceptación del nuevo sistema.  Mejoras en la administración de proyectos.  Mejoras en la comunicación entre desarrolladores y clientes.  Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.  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. | Que el prototipo se construya y sirva como un mecanismo para la definición de requisitos.  Que el prototipo se descarte, al menos en parte.  Que después se desarrolle el software real con un enfoque hacia la calidad.  El cliente ve funcionando lo que para él es la primera versión del prototipo que ha sido construido con “chicle y cable para embalaje”, y puede decepcionarse al indicarle que el sistema aún no ha sido construido.  El desarrollador puede caer en la tentación de aumentar el prototipo para construir el sistema final sin tener en cuenta las obligaciones de calidad y de mantenimiento que tiene con el cliente. |
| MODELO DRA | Los entregables pueden ser fácilmente trasladados a otra plataforma.  Visibilidad temprana.  Menor codificación manual.  Mayor involucramiento de los usuarios.  Ciclos de desarrollo más pequeños.  Interfaz gráfica estándar. | El DRA se basa en componentes y por ende en el trabajo en paralelo de distintos equipos DRA (esto se hace para ganar tiempo) cuando se trata de un proyecto grande puede ser imposible mantener una gran cantidad de equipos funcionando al mismo tiempo.  Se requiere que todos los implicados en el desarrollo estén comprometidos con la rapidez (DRA) si faltara el compromiso de alguna parte el desarrollo acelerado perdería su esencia  Además, es importante aclarar que no todos los proyectos se pueden modularizar y entonces no se pueden distribuir entre equipos de trabajo |
| MODELOS EVOLUTIVOS | ES INTERACTIVO  Con cada incremento se entrega al cliente un producto operacional, que puede evaluarlo  PERSONAL  Permite variar el personal asignado a cada interacción  GESTION RIESGOS TECNICOS  Por ejemplo, disponibilidad de hardware especifico | La primera interacción puede plantear los mismos problemas que un modelo lineal secuencial |
| MODELO ITERATIVO INCREMENTAL | Resolución de problemas de alto riesgo en tiempos tempranos del proyecto. Visión de avance en el desarrollo desde las etapas iníciales del desarrollo.  Menor tasa de fallo del proyecto, mejor productividad del equipo, y menor cantidad de defectos, según demuestran estudios realizados sobre proyectos que han aplicado esta técnica.  Permite manejar la complejidad del proyecto, apuntando a la resolución de los problemas por partes, y no caer en la inanición del “súper análisis” del producto.  El aprendizaje y experiencia del equipo iteración tras iteración, mejora exponencialmente el trabajo, aumenta la productividad y permite optimizar el proceso en el corto plazo.  El trabajo iterativo deja una experiencia en el equipo que permite ir ajustando y mejorando las planificaciones, logrando menores desvíos en la duración total del proyecto.  Su adopción, con ciertos recaudos, no presenta grandes inversiones. | Hasta el momento se podría decir que no existen grandes desventajas, pero sí hay puntos a manejar con sumo cuidado:  El uso de un desarrollo iterativo e incremental no garantiza por sí solo el éxito de su uso.  Hay costos ocultos en su implementación, ya que se incorporan varias actividades a realizar por el equipo, y hay que saber medir ese impacto para no fracasar en el intento. |
| MODELO ESPIRAL | El modelado en espiral puede adaptarse y aplicarse a lo largo de la vida del software de computadora, no terminal cuando se entrega el software.  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 antes de que se conviertan en problemáticos.  años antes de que se determine con absoluta certeza la eficacia de este nuevo e importante paradigma. | Requiere mucha experiencia y habilidad para la evaluación de los riesgos, lo cual es requisito para el éxito del proyecto.  Es difícil convencer a los grandes clientes que se podrá controlar este enfoque evolutivo.  Este modelo no se ha usado tanto, como el Cascada (Incremental) o [MCP](http://es.wikipedia.org/wiki/Modelo_de_prototipos), por lo que no se tiene bien medida su eficacia, es un paradigma relativamente nuevo y difícil de implementar y controlar. |
| MODELO BASADO EN COMPONENTES | Reutilización del software. Nos lleva a alcanzar un mayor nivel de reutilización de software.  Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.  Reducirá el tiempo de programación. Los costos de programación y mantenimiento del software se reducen al distribuir componentes que pueden compartirse y volverse a usar entre proyectos múltiples.  Ciclos de desarrollo más cortos. La adición de una pieza dada de funcionalidad tomará días en lugar de meses ó años.  Mejor ROI. Usando correctamente esta estrategia, el retorno sobre la inversión puede ser más favorable que desarrollando los componentes uno mismo. | Genera mucho tiempo en el desarrollo del sistema - Modelo costoso –Requiere experiencia en la identificación de riesgos  Genera mucho trabajo adicional. Cuando un sistema falla se pierde tiempo y coste dentro de la empresa. Exige una cierta habilidad en los analistas (es bastante difícil). |
| MODELO ESPIRAL WIN & WIN | El modelo en espiral puede adaptarse y aplicarse a lo largo de la vida del software de computadora.  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.  El modelo en espiral permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.  El modelo en espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto y si se aplica adecuadamente debe reducir los riesgos antes de que se conviertan en problemas | Resulta difícil convencer a grandes clientes de que el enfoque evolutivo es controlable.  Debido a su elevada complejidad no se aconseja utilizarlo en pequeños sistemas.  Genera mucho tiempo en el desarrollo de sistemas. |

BIBLIOGRAFIA

<https://www.codejobs.biz/es/blog/2013/05/25/el-proceso-del-software>

<http://es.wikipedia.org/wiki/Software#Modelos_evolutivos>

[http://www.buenastareas.com/ensayos/Ingenieria-De-Software-Basada-En- Componentes/232694.html](http://www.buenastareas.com/ensayos/Ingenieria-De-Software-Basada-En-Componentes/232694.html)

<http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software>

<http://scruz334.blogspot.es/1193793960/>

<http://148.202.148.5/cursos/cc321/fundamentos/unidad1/espiral.htm>

<http://www.slideshare.net/JoanFernandoChipia/modelos-basados-en-prototipos>

<http://html.rincondelvago.com/modelos-de-procesos-de-software.html>

<http://www.mitecnologico.com/Main/ModelosDeProcesoDeSoftware>

[www.dsic.upv.es/asignaturas/.../lsi/.../IntroduccionProcesoSW.doc](http://www.dsic.upv.es/asignaturas/.../lsi/.../IntroduccionProcesoSW.doc)

[www.e-market.cl/dir/umayor/ingsw/06-01\_vesp/espiral.ppt](http://www.e-market.cl/dir/umayor/ingsw/06-01_vesp/espiral.ppt)

<http://www.slideshare.net/chiki.carito/procesos-del-software>

[http://www.mitecnologico.com/Main/ModeloBasadoEnComponentesDise%F1o De Sistema](http://www.mitecnologico.com/Main/ModeloBasadoEnComponentesDise%F1oDeSistemas)s