PROCESOS DE SOFTWARE, MODELOS Y COMPARATIVA

CRISTHIAN URREGO SALAZAR

CENTRO DE ESTUDIOS TECNOLÓGICOS DEL NORTE DEL VALLE COTECNOVA

TECNOLOGÍA EN SISTEMAS DE INFORMACIÓN

CARTAGO VALLE

2017

PROCESOS DE SOFTWARE, MODELOS Y COMPARATIVA

CRISTHIAN URREGO SALAZAR

NOMBRE PROFEROS

DOCENTE DISEÑO DE SISTEMAS DE INFORMACIÓN

CENTRO DE ESTUDIOS TECNOLÓGICOS DEL NORTE DEL VALLE COTECNOVA

TECNOLOGÍA EN SISTEMAS DE INFORMACIÓN

CARTAGO VALLE

2017

**QUE ES UN PROCESO DE SOFTWARE**

La meta de la ingeniería de software es construir productos de software, o mejorar los existentes; en ingeniería de procesos, la meta es desarrollar o mejorar procesos.

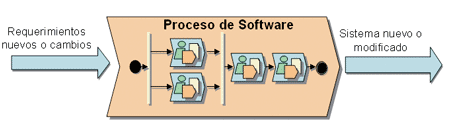
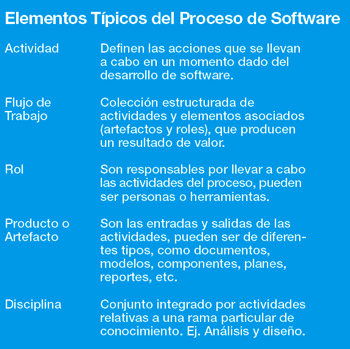
Un proceso de desarrollo de software es un conjunto de personas, estructuras de organización, reglas, políticas, actividades y sus procedimientos, componentes de software, metodologías, y herramientas utilizadas o creadas específicamente para definir, desarrollar, ofrecer un servicio, innovar y extender un producto de software.

Un proceso de software efectivo habilita a la organización a incrementar su productividad al desarrollar software:

* Permite estandarizar esfuerzos, promover reusó, repetición y consistencia entre proyectos.
* Provee la oportunidad de introducir mejores prácticas de la industria.
* Permite entender que las herramientas deben ser utilizadas para soportar un proceso.
* Establece la base para una mayor consistencia y mejoras futuras.

Un proceso de software mejora los esfuerzos de mantenimiento y soporte:

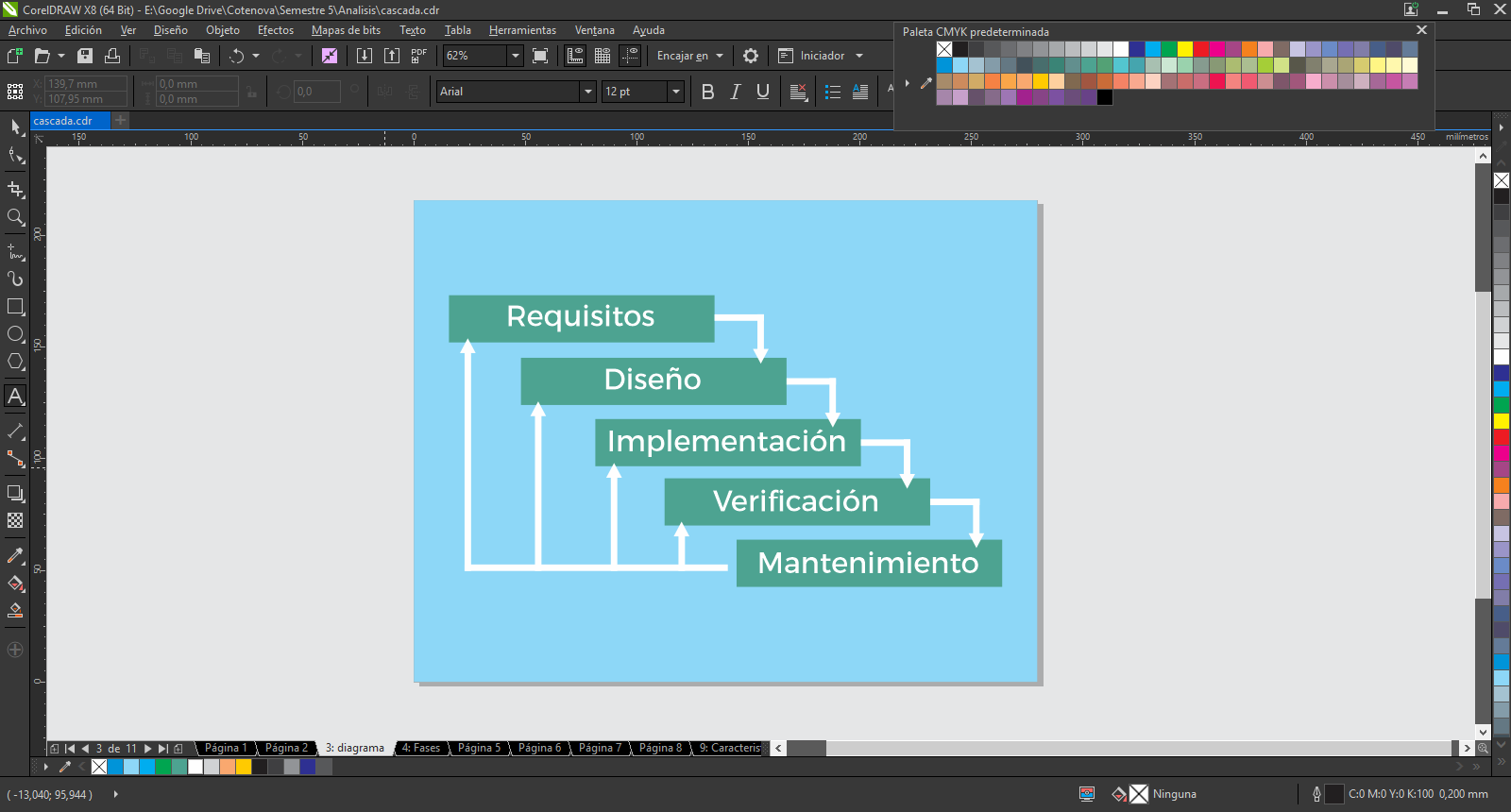
* Define cómo manejar los cambios y liberaciones a sistemas de software existentes.
* Define cómo lograr la transición del software a la operación, y cómo ejecutar los esfuerzos de operación y soporte.

#### MODELOS DE PROCESOS

**MODELO CASCADA**

Su visión es muy simple: el desarrollo de software se debe realizar siguiendo una secuencia de fases. Cada etapa tiene un conjunto de metas bien definidas y las actividades dentro de cada una contribuyen a la satisfacción de metas de esa fase o quizás a una subsecuencia de metas de la misma.



**FASES**

**Requisitos**

**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 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.

**Implementación**

El diseño debe traducirse en una forma legible para la máquina. Si el diseño se realiza de una manera detallada, la codificación puede realizarse mecánicamente.

**Verificación**

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.

**Mantenimiento**

El software sufrirá cambios después de que se entrega al cliente. Los cambios ocurrirán debidos a que se haya encontrado errores, a que el software deba adaptarse a cambios del entorno externo (sistema operativo o dispositivos periféricos) o a que el cliente requiera ampliaciones funcionales o del rendimiento.

**Características**

•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.

•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.

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

**Ventajas**

•Se tiene todo bien organizado y no se mezclan las fases.

•Ayuda a localizar errores en las primeras etapas del proyecto a un bajo costo.

•Ayuda a minimizar los gastos de la planificación porque permite realizarla sin problemas.

**Desventajas**

•Gran dependencia en los requerimientos iníciales

•Difícilmente un cliente va a establecer al principio todos los requerimientos necesarios, por lo que provoca un gran atraso trabajando en este modelo, ya que este es muy restrictivo y no permite movilizarse entre fases.

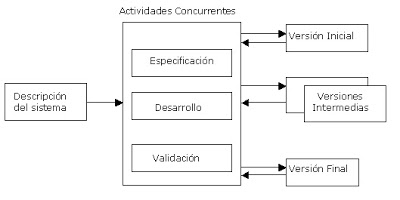
•El modelo genera pocos signos visibles de progreso hasta el final. Esto puede dar la impresión de un desarrollo lento, existe la incertidumbre de los clientes si sus proyectos serán entregados a tiempo.

•Inicio de la codificación muy tarde en el ciclo de vida del proyecto.

**MODELO EVOLUTIVO**

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.

La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado.Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.



**Existen dos tipos de desarrollo evolutivo:**

·       Desarrollo Exploratorio: El objetivo de este enfoque es explorar con el usuario los requisitos hasta llegar a un sistema final. El desarrollo comienza con las partes que se tiene más claras. El sistema evoluciona conforme se añaden nuevas características propuestas por el usuario.

·       Enfoque utilizando prototipos: El objetivo es entender los requisitos del usuario y trabajar para mejorar la calidad de los requisitos. A diferencia del desarrollo exploratorio, se comienza por definir los requisitos que no están claros para el usuario y se utiliza un prototipo para experimentar con ellos. El prototipo ayuda a terminar de definir estos requisitos.

**VENTAJAS**

·       La especificación puede desarrollarse de forma creciente.

·       Los usuarios y desarrolladores logran un mejor entendimiento del sistema. Esto se refleja en una mejora de la calidad del software.

·       Es más efectivo que el modelo de cascada, ya que cumple con las necesidades inmediatas del cliente.

**DESVENTAJAS**

·       Proceso no Visible: Los administradores necesitan entregas para medir el progreso. Si el sistema se necesita desarrollar rápido, no es efectivo producir documentos que reflejen cada versión del sistema.

·       Sistemas pobremente estructurados: Los cambios continuos pueden ser perjudiciales para la estructura del software haciendo costoso el mantenimiento.

·       Se requieren técnicas y herramientas: Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

**METODOS FORMALES**

La denominación métodos formalesse usa para referirse a cualquier actividad relacionada con representaciones matemáticas del software, incluyendo la especificación formal de sistemas, análisis y demostración de la especificación, el desarrollo transformacional y la verificación de programas. Todas estas actividades dependen de una especificación formal del software.

Una especificación formal del software es una especificación expresada en un lenguaje cuyo vocabulario, sintaxis y semántica están formalmente definidos. Esta necesidad de unadefinición formal significa que los lenguajes de especificación deben basarse en conceptos matemáticos cuyas propiedades se comprendan bien. La rama de las matemáticas usada es la de matemática discreta, y los conceptos matemáticos provienen de la teoría de conjuntos, la lógica y el álgebra.

En la década de los 80, muchos investigadores de ingeniería del software propusieron que el uso de métodos formales de desarrollo era la mejor forma de mejorar la calidad del software. Argumentaban que el rigor y el análisis detallado, que son una parte esencial de los métodos formales, podrían dar lugar a programas con menos errores y más adecuados a las necesidades de los usuarios. Predijeron que, en el siglo XXI, una gran proporción del software estaría desarrollado usando métodos formales.

Claramente, esta predicción no se ha hecho realidad. Existen cuatro razones principales para esto:

*1. Una ingeniería del software exitosa.*El uso de otros métodos de ingeniería del software como los métodos estructurados, gestión de configuraciones y ocultación de la información en el diseño del software y procesos de desarrollo ha conducido a mejoras en la calidad del software. La gente que sugirió que la única forma de mejorar la calidad del software era usando métodos formales estaba claramente equivocada.

*2. Cambios en el mercado.*En la década de los 80, la calidad del software fue vista como un problema clave de la ingeniería del software. Sin embargo, desde entonces, la cuestión crítica para muchas clases de desarrollo del software no es la calidad, sino la oportunidad de mercado. El software debe desarrollarse rápidamente, y los clientes están dispuestos a aceptar software con algunos defectos si se les entrega rápidamente. Las técnicas para el desarrollo rápido del software no funcionan de forma efectiva con las especificaciones formales. Por supuesto, la calidad todavía es un factor importante, pero debe lograrse en el contexto de entrega rápida.

*3. Ámbito limitado de los métodos formales.*Los métodos formales no son muy apropiados para la especificación de interfaces de usuario e interacciones del usuario. El componente de interfaz de usuario se ha convertido en una parte cada vez mayor de la mayoría de los sistemas, de manera que realmente s6lo pueden usarse métodos formales cuando se desarrollan las otras partes del sistema.

*4. Escalabilidad limitada de los métodos formales.*Los métodos formales todavía no son muy escalables. La mayoría de los proyectos con éxito que han usado estas técnicas han estado relacionados con núcleos de sistemas críticos relativamente pequeños. A medida que los sistemas incrementan su tamaño, el tiempo y esfuerzo requerido para desarrollar una especificación formal crece de forma desproporcionada.

**VENTAJAS**

•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

**DESVENTAJAS**

•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.

**DESARROLLO BASADO EN COMPONENTES (REUTILIZACIÓN)**

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.

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.

**Beneficios del Desarrollo de Software Basado en Componentes**

**El uso de este paradigma posee algunas ventajas:**

1. Reutilización del software. Nos lleva a alcanzar un mayor nivel de reutilización de software.

2. Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.

3. Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desabollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

4. Mayor calidad. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo

**La Notación de Componentes**

Un componente puede ser algo como un control Actives; tanto un componente de la Interfaz de usuario como un servidor de reglas de negocio.

**El Diagrama de Componentes**

El diagrama de componentes muestra la relación entre componentes de software, sus dependencias, su comunicación su ubicación y otras condiciones.

**Interfaces**

Los componentes también pueden exponer las interfaces. Estas son los puntos visibles de entrada o los servicios que un componente está ofreciendo y dejando disponibles a otros componentes de software y clases. Típicamente, un componente está compuesto por numerosas clases y paquetes de clases internos. También se puede crear a partir de una colección de componentes más pequeños.

**Los componentes y los Nodos**

Un diagrama de despliegue muestra el despliegue físico del sistema en un ambiente de producción (o de prueba). Muestra dónde se ubican los componentes, en qué servidores, máquinas o hardware. Puede representar los enlaces de redes.

**Restricciones**

Los componentes pueden restricciones asignadas que indican el entorno en el que operan.

Las pre-condiciones especifican lo que debe ser verdadero antes de que un componente pueda realizar alguna función; las post-condiciones indican lo que debe ser verdadero después de que un componente haya realizado algún trabajo y los invariantes especifican lo que debe permanecer verdadero durante la vida del componente.

Tenemos la fortuna de presenciar el nacimiento de una nueva forma de hacer software, que traerá beneficios inmensos para todos. El desarrollo de software basado en componentes desde siempre fue la idea revolucionaria que nos llevó a pensar que sí era posible el construir software de calidad en corto tiempo y con la misma calidad que la mayoría de las industrias de nuestro tiempo. Al mirar hacia atrás, vemos los increíbles avances que hemos logrado en la comprensión de la forma correcta de reutilizar el software y el conocimiento existente, y nos asombramos cada vez más al darnos cuenta de que este solo es el inicio.

El desarrollo de software basado de componentes se convirtió en el pilar de la Revolución Industrial del Software y se proyecta hoy en día en diversas nuevas formas de hacer software de calidad con los costos más bajos del mercado y en tiempos que antes eran impensables. Empresas como Microsoft entendieron el potencial de esta metodología hace años y hoy nos ofrecen nuevas iniciativas y herramientas que buscan llevar al proceso de construcción de software hacia el sitial privilegiado en el que debió colocarse desde un principio.

**Análisis del riesgo**

Se estudian todos los riesgos potenciales y se seleccionan una o varias alternativas propuestas para reducir o eliminar los riesgos.

Planificar Revisamos todo lo hecho, evaluándolo, y con ello decidimos si continuamos con las fases siguientes y planificamos la próxima actividad.

**Ventajas**

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

**Desventajas:**

• Genera mucho tiempo en el desarrollo del sistema - Modelo costoso –Requiere experiencia en la identificación de riesgos

• Inconvenientes

• 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 INCREMENTAL**

El modelo incremental fue propuesto por Harlan Mills en el año 1980. Surgió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema. Este modelo se conoce también bajo las siguientes denominaciones:

* Método de las comparaciones limitadas sucesivas.
* Ciencia de salir del paso.
* Método de atacar el problema por ramas.

El Modelo Incremental combina elementos del Modelo Lineal Secuencial con la filosofía interactiva de Construcción de Prototipos. Como se muestra en la Figura 1, el modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software. El primer incremento generalmente es un producto esencial denominado núcleo.

En una visión genérica, el proceso se divide en 4 partes:

* Análisis
* Diseño
* Código
* Prueba

  
  
Sin embargo, para la producción del Software, se usa el principio de trabajo en cadena o Pipeline. Con esto se mantiene al cliente en constante contacto con los resultados obtenidos en cada incremento. Es el mismo cliente el que incluye o desecha elementos al final de cada incremento a fin de que el software se adapte mejor a sus necesidades reales. El proceso se repite hasta que se elabora el producto completo. De esta forma el tiempo de entrega se reduce considerablemente.  
  
El Modelo Incremental es de naturaleza interactiva brindando al final de cada incremento la entrega de un producto completamente operacional. Este modelo es particularmente útil cuando no se cuenta con una dotación de personal suficiente. Los primeros pasos los pueden realizar un grupo reducido de personas y en cada incremento se añadirá personal, de ser necesario. Por otro lado los incrementos se pueden planear para gestionar riesgos técnicos.

##### **Durante el proceso se trata de llevar a cabo al proyecto en diferentes partes que al final terminará siendo la solución completa requerida por el cliente, pero éstas partes no se pueden realizar en cualquier orden, sino que dependen de lo que el cliente este necesitando con más urgencia, de los puntos más importantes del proyecto, los requerimientos más básicos, difíciles y con mayor grado de riesgo, ya que estos se deben hacer al comienzo, de manera que se disminuya la dificultad y el riesgo en cada versión.**

De este modo podemos terminar una aplicación ejecutable (primera versión) que podrá ser entregada al cliente para que éste pueda trabajar en ella y el programador pueda considerar las recomendaciones que el cliente efectúe para hacer mejoras en el producto. Estas nuevas mejoras deberán esperar a ser integradas en la siguiente versión junto con los demás requerimientos que no fueron tomados en cuenta en la versión anterior.

El modelo incremental consiste en un desarrollo inicial de la arquitectura completa del sistema, seguido de sucesivos incrementos funcionales. Cada incremento tiene su propio ciclo de vida y se basa en el anterior, sin cambiar su funcionalidad ni sus interfaces. Una vez entregado un incremento, no se realizan cambios sobre el mismo, sino únicamente corrección de errores. Dado que la arquitectura completa se desarrolla en la etapa inicial, es necesario conocer los requerimientos completos al comienzo del desarrollo.

Al iniciar del desarrollo, los clientes o los usuarios, identifican a grandes rasgos, las funcionalidades que proporcionará el sistema. Se confecciona un bosquejo de requisitos funcionales y será el cliente quien se encarga de priorizar que funcionalidades son mas importantes. Con las funcionalidades priorizadas, se puede confeccionar un plan de incrementos, donde en cada incremento se indica un subconjunto de funcionalidades que el sistema entregará. La asignación de funcionalidades a los incrementos depende de la prioridad dada a los requisitos. Finalizado el plan de incrementos, se puede comenzar con el primer incremento.  
  
**CARACTERÍSTICAS**

* Se evitan proyectos largos y se entrega "algo de valor" a los usuarios con cierta frecuencia.
* El usuario se involucra mas.
* Dificil de evaluar el costo total.
* Dificil 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 positivo.

**VENTAJAS**

* 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.
* Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos.

### DESVENTAJAS

* 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 del proyecto.

**MODELO EN ESPIRAL**

Modelo Espiral, propuesto originalmente por Boehm, es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software.

En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteracciones, la version incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.

**Regiones de Tareas del Modelo**

El modelo en espiral se divide en un número de actividades de marco de trabajo, 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 yel cliente.

**Planificación**

Las tareas requeridas para definir recursos, el tiempo y otra información relacionadas con el proyecto.

**Análisis de riesgos**

Las tareas requeridas para evaluar riesgos técnicos y de gestión.

**Ingeniería**

Las tareas requeridas para construir una o más representaciones de la aplicación.

**Construcción y acción**

Las tareas requeridas para construir, probar, instalar yproporcionar soporte al usuario (por ejemplo: documentación y práctica)

**Evaluación del 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 implementada durante la etapa de instalación. Cada una de las regiones están compuestas por un conjunto de tareas del trabajo, llamado conjunto de tareas, que se adaptan a las características del proyecto que va a emprenderse. Para proyectos pequeños, el número de tareas de trabajo y suformalidad es bajo. Para proyectos mayores ymás críticos cada región de tareas contiene tareas de trabajo que se definen para lograr un nivel más alto de formalidad. En todos los casos, se aplican las actividades de protección.

**VENTAJAS**

**•**Puede adaptarse y aplicarse a lo largo de la vida del software de computadora.

**•**Es un enfoque realista del desarrollo de sistemas y de software a gran escala.

**•**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.

**•**Utiliza la construcción de prototipos como mecanismo de reducción de riesgos.

**•**Permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.

**•**Mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo iterativo que refleja de forma más realista el mundo real.

**•**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 problemáticos.

**DESVENTAJAS**

**•**Puede resultar difícil convencer a grandes clientes (particularmente en situaciones bajo contrato) de que el enfoque evolutivo es controlable.

**•**Requiere una considerable habilidad para la evaluación del riesgo.

**•**No se ha utilizado tanto como los paradigmas lineales secuenciales o de construcción de prototipos.

**BIBLIOGRAFIA**

Calero, W. (25 de Agosto de 2017). Obtenido de http://ingenieraupoliana.blogspot.com.co/2010/10/modelo-de-desarrollo-basado-en.html

EcuRed. (25 de Agosto de 2017). Obtenido de https://www.ecured.cu/Modelo\_Espiral

EduRed. (25 de Agosto de 2017). *EduRed*. Obtenido de https://www.ecured.cu/Modelo\_en\_cascada

Jortiz. (25 de Agosto de 2017). Obtenido de http://tema3isoftware.blogspot.com.co/p/modelo-de-metodos-formales.html

Ruvalcaba, M. (s.f.). *Procesos de Software*. Recuperado el 23 de 08 de 2017, de SG Buzz: https://sg.com.mx/revista/1/procesos-software#.WZ3alVFYGUk

Soloriio, M. (25 de Agosto de 2017). *metodologiaencascada* . Obtenido de blogspot: http://metodologiaencascada.blogspot.com.co/

Trejos, J. (25 de Agosto de 2017). Obtenido de http://jorgetrejos.blogspot.com.co/2010/08/modelo-evolutivo.html

Zulay, M. (25 de Agosto de 2017). Obtenido de http://modelosprocesosdesoftware.blogspot.com.co/p/modelo-en-cascada-o-lineal-secuencial.html