# ENTORNOS DE DESARROLLO

2013

Egibide

# Entornos de Desarrollo

2015

Departamento de Informática

# Ingeniería del software

## 1. Definición

Ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y el estudio de estos enfoques.

## 2. Objetivo

La ingeniería de software aplica diferentes normas y métodos que permiten obtener mejores resultados, en cuanto al desarrollo y uso del software, mediante la aplicación correcta de estos procedimientos se puede llegar a cumplir de manera satisfactoria con los objetivos fundamentales de la ingeniería de software.
Entre los objetivos de la ingeniería de software están:

- Mejorar el diseño de aplicaciones o software de tal modo que se adapten de mejor manera a las necesidades de las organizaciones o finalidades para las cuales fueron creadas.
- Promover mayor calidad al desarrollar aplicaciones complejas.
- Brindar mayor exactitud en los costos de proyectos y tiempo de desarrollo de los mismos.
- Aumentar la eficiencia de los sistemas al introducir procesos que permitan medir mediante normas específicas, la calidad del software desarrollado, buscando siempre la mejor calidad posible según las necesidades y resultados que se quieren generar.
- Una mejor organización de equipos de trabajo, en el área de desarrollo y mantenimiento de software.
- Detectar a través de pruebas, posibles mejoras para un mejor funcionamiento del software desarrollado.

## 3. Recursos

### 3.1. Recurso humano

Son todas aquellas personas que interviene en la planificación de cualquier instancias de software (por ejemplo: gestor, ingeniero de software experimentado, etc.), El número de personas requerido para un proyecto de software sólo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo.

### 3.2. Recursos de software reutilizables

Son aquellos componentes de un software que son usados en otras aplicaciones. De la misma índole, ya sea para reducir costos o tiempo.

### 3.3. Recursos de entorno

Es el entorno de las aplicaciones (software y hardware) el hardware proporciona el medio físico para desarrollar las aplicaciones (software), este recurso es indispensable.

# 4. Notaciones

## 4.1. LUM (lenguaje unificado de modelado) o UML

Es un lenguaje de modelado muy reconocido y utilizado actualmente que se utiliza para describir o especificar métodos. También es aplicable en el desarrollo de software.

Las siglas UML significan lenguaje unificado de modelado esto quiere decir que no pretende definir un modelo estándar de desarrollo, sino únicamente un lenguaje de modelado.

Un lenguaje de modelado consiste de vistas, elementos de modelo y un conjunto de reglas: sintácticas, semánticas y pragmáticas que indican cómo utilizar los elementos.

## 4.2. BPMN (notación para el modelado de procesos de negocios)

El objetivo de la notación para el modelado de procesos de negocios es proporcionar de una manera fácil de definir y analizar los procesos de negocios públicos y privados simulando un diagrama de flujo. La notación ha sido diseñada específicamente para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes del mismo, con un conjunto de actividades relacionadas.

## 4.3. Diagrama de flujo de datos (DFD)

Un diagrama de flujo de datos permite el movimiento de datos a través de un sistema por medio de modelos que describen los flujos de datos, los procesos que tranforman o cambian los datos, los destinos de datos y los almacenamientos de datos a la cual tiene acceso el sistema.

# 5. Metodología

Un objetivo de décadas ha sido el encontrar procesos y metodologías, que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del producto software, en pocas palabras, determina los pasos a seguir y como realizarlos para finalizar una tarea.

## 5.1. Etapas del proceso

La ingeniería de software requiere llevar a cabo numerosas tareas agrupadas en etapas, al conjunto de estas etapas se le denomina ciclo de vida. Las etapas comunes a casi todos los modelos de ciclo de vida son las siguientes:

### 5.1.1. Obtención de los requerimientos

Se debe identificar sobre que se está trabajando es decir, el tema principal que motiva el inicio del estudio y creación del nuevo software o modificación de uno ya existente. A su vez identificar los recursos que se tienen, en esto entra el conocer los recursos humanos y materiales que participan en el desarrollo de las actividades.

Se tienen que tener dominio de información de un problema lo cual incluye los datos fuera del software(usuarios finales, otros sistemas o dispositivos externos), los datos salen del sistema (por la interfaz de usuario, interfaces de red, reportes, gráficas y otros medios) y los almacenamientos de datos que recaban y organizan objetos persistentes de datos (por ejemplo, aquellos que se conservan de manera permanente).

También hay que ver los puntos críticos lo que significa tener de una manera clara los aspectos que entorpecen y limitan el buen funcionamiento de los procedimientos actuales, los problemas más comunes y relevantes que se presentan, los motivos que crean insatisfacción y aquellos que deben ser cubiertos a plenitud. Por ejemplo: ¿El contenido de los reportes generados, satisface realmente las necesidades del usuario? ¿Los tiempos de respuesta ofrecidos, son oportunos?, etc.

Hay que definir las funciones que realizara el software ya que estas ayudan al usuario final y al funcionamiento del mismo programa.

Se tiene que tener en cuenta como será el comportamiento del software antes situaciones inesperadas como lo son por ejemplo una cantidad de usuarios enormes usando el software o una gran cantidad de datos entre otros.

### 5.1.2. Análisis de requisitos

Extraer los requisitos de un producto software es la primera etapa para crearlo. Durante la fase de análisis, el cliente plantea las necesidades que se presenta e intenta explicar lo que debería hacer el software o producto final para satisfacer dicha necesidad mientras que el desarrollador actúa como interrogador, como la persona que resuelve problemas. Con este análisis, el ingeniero de sistemas puede elegir la función que debe realizar el software y establecer o indicar cual es la interfaz más adecuada para el mismo.

El análisis de requisitos puede parecer una tarea sencilla, pero no lo es debido a que muchas veces los clientes piensan que saben todo lo que el software necesita para su buen funcionamiento, sin embargo se requiere la habilidad y experiencia de algún especialista para reconocer requisitos incompletos, ambiguos o contradictorios. Estos requisitos se determinan tomando en cuenta las necesidades del usuario final, introduciendo técnicas que nos permitan mejorar la calidad de los sistemas sobre los que se trabaja.

El resultado del análisis de requisitos con el cliente se plasma en el documento ERS (especificación de requisitos del sistema), cuya estructura puede venir definida por varios estándares, tales como CMMI. Asimismo, se define un diagrama de entidad/relación, en el que se plasman las principales entidades que participarán en el desarrollo del software.
La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de

esta etapa depende en gran medida el logro de los objetivos finales. Se han ideado modelos y diversos procesos metódicos de trabajo para estos fines. Aunque aún no está formalizada, ya se habla de la ingeniería de requisitos.

La IEEE Std. 830-1998 normaliza la creación de las especificaciones de requisitos de software (Software Requirements Specification).

Finalidades del análisis de requisitos:

- Brindar al usuario todo lo necesario para que pueda trabajar en conjunto con el software desarrollado obteniendo los mejores resultados posibles.
- Tener un control más completo en la etapa creación del software, en cuanto a tiempo de desarrollo y costos.
- Utilización de métodos más eficientes que permitan el mejor aprovechamiento del software según sea la finalidad de uso del mismo.
- Aumentar la calidad del software desarrollado al disminuir los riesgos de mal funcionamiento.

### 5.1.3. Limitaciones

Los software tienen la capacidad de emular inteligencia creando un modelo de ciertas características de la inteligencia humana pero sólo posee funciones predefinidas que abarcan un conjunto de soluciones que en algunos campos llega a ser limitado. Aun cuando tiene la capacidad de imitar ciertos comportamientos humanos no es capaz de emular el pensamiento humano porque actúa bajo condiciones.

Otro aspecto limitante de los software proviene del proceso totalmente mecánico que requiere de un mayor esfuerzo y tiempos elevados de ejecución lo que lleva a tener que implementar el software en una máquina de mayor capacidad.

### 5.1.4. Especificación

La especificación de requisitos describe el comportamiento esperado en el software una vez desarrollado. Gran parte del éxito de un proyecto de software radicará en la identificación de las necesidades del negocio (definidas por la alta dirección), así como la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requisitos del software.

### 5.1.5. Arquitectura

El arquitecto de software es la persona que añade valor a los procesos de negocios gracias a su valioso aporte de soluciones tecnológicas.

La arquitectura de sistemas en general, es una actividad de planeación, ya sea a nivel de infraestructura de red y hardware, o de software.

Lo principal en este punto es poner en claro los aspectos lógicos y físicos de las salidas, modelos de organización y representación de datos, entradas y procesos que componen el sistema.

Hay que tener en consideración la arquitectura del sistema en la cual se va a trabajar, elaborar un plan de trabajo viendo la prioridad de tiempo y recursos disponibles. En los diseños de salidas entra los que es la interpretación de requerimientos lo cual es el dominio de información del problema, las funciones visibles para el usuario, el comportamiento del sistema y un conjunto de clases de requerimientos que agrupa los objetos del negocio con los métodos que les dan servicio.

La arquitectura de software consiste en el diseño de componentes de una aplicación (entidades del negocio), generalmente utilizando patrones de arquitectura. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado, por ejemplo por medio de diagramas de secuencia. Un diseño arquitectónico describe en general el cómo se construirá una aplicación de software. Para ello se documenta utilizando diagramas, por ejemplo:

- Diagramas de clases
- Diagramas de base de datos
- Diagrama de despliegue
- Diagrama de secuencia

Siendo los dos primeros los mínimos necesarios para describir la arquitectura de un proyecto que iniciará a ser codificado. Dependiendo del alcance del proyecto, complejidad y necesidades, el arquitecto elegirá cuales de los diagramas se requiere elaborar.

Las herramientas para el diseño y modelado de software se denominan CASE (Computer Aided Software Engineering) entre las cuales se encuentran:

- Enterprise Architect
- Microsoft Visio for Enterprise Architects

### 5.1.6. Programación

Implementar un diseño en código puede ser la parte más obvia del trabajo de ingeniería de software, pero no necesariamente es la que demanda mayor trabajo y ni la más complicada. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los lenguajes de programación utilizados, así como al diseño previamente realizado.

### 5.1.7. Desarrollo de la aplicación

### 5.1.8. Pruebas de software

Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación del problema. Una técnica es probar por separado cada módulo del software, y luego

probarlo de manera integral, para así llegar al objetivo. Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó, idealmente un área de pruebas; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. En general hay dos grandes maneras de organizar un área de pruebas, la primera es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta manera se evalúa que la documentación entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el software hace las cosas tal y como están descritas. El segundo enfoque es tener un área de pruebas conformada por programadores con experiencia, personas que saben sin mayores indicaciones en qué condiciones puede fallar una aplicación y que pueden poner atención en detalles que personal inexperto no consideraría.

#### 5.1.9. Implementación

Una Implementación es la realización de una especificación técnica o algoritmos con un programa, componente software, u otro sistema de cómputo. Muchas especificaciones son dadas según a su especificación o un estándar. Las especificaciones recomendadas según el ´World Wide Web Consortium, y las herramientas de desarrollo del software contienen implementaciones de lenguajes de programación. El modelo de implementación es una colección de componentes y los subsitemas que contienen. Componentes tales como: ficheros ejecutables, ficheros de código fuente y totro tipo de ficheros que sean necesarios para la implementación y despliegue del sistema.

#### 5.1.10. Documentación

Es todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelaciones (UML), diagramas de casos de uso, pruebas, manuales de usuario, manuales técnicos, etc; todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.

#### 5.1.11. Mantenimiento

Fase dedicada a mantener y mejorar el software para corregir errores descubiertos e incorporar nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo del software inicial. Alrededor de 2/3 del tiempo de ciclo de vida de un proyecto está dedicado a su mantenimiento. Una pequeña parte de este trabajo consiste eliminar errores (bugs); siendo que la mayor parte reside en extender el sistema para incorporarle nuevas funcionalidades y hacer frente a su evolución.

## 5.2. Ventajas

#### 5.2.1. Desde el punto de vista de gestión

- Facilitar la tarea de seguimiento del proyecto
- Optimizar el uso de recursos
- Facilitar la comunicación entre usuarios y desarrolladores

- Facilitar la evaluación de resultados y cumplimiento de objetivos

#### 5.2.2. Desde el punto de vista de los ingenieros de Software

- Ayudar a comprender el problema
- Permitir la reutilización
- Facilitar el mantenimiento del producto final
- Optimizar el conjunto y cada una de las fases del proceso de desarrollo

#### 5.2.3. Desde el punto de vista de cliente o usuario final

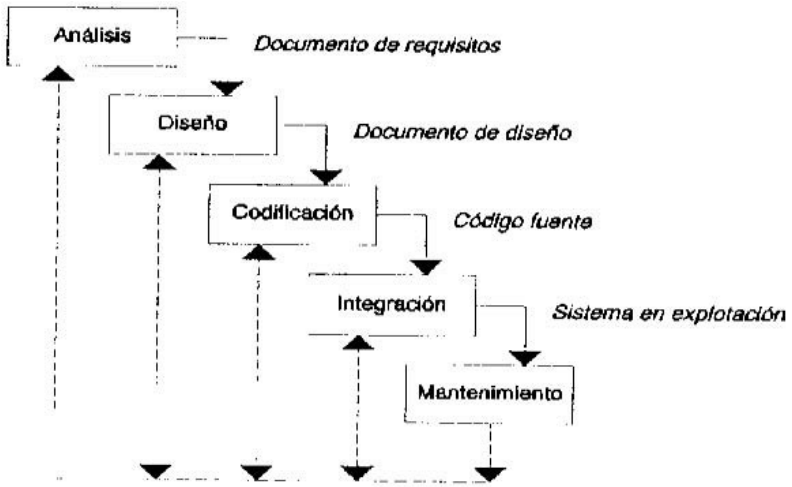
- Garantizar el nivel de calidad del producto final
- Obtener el ciclo de vida adecuado para el proyecto
- Confianza en los plazos del tiempo mostrados en la definición del proyecto

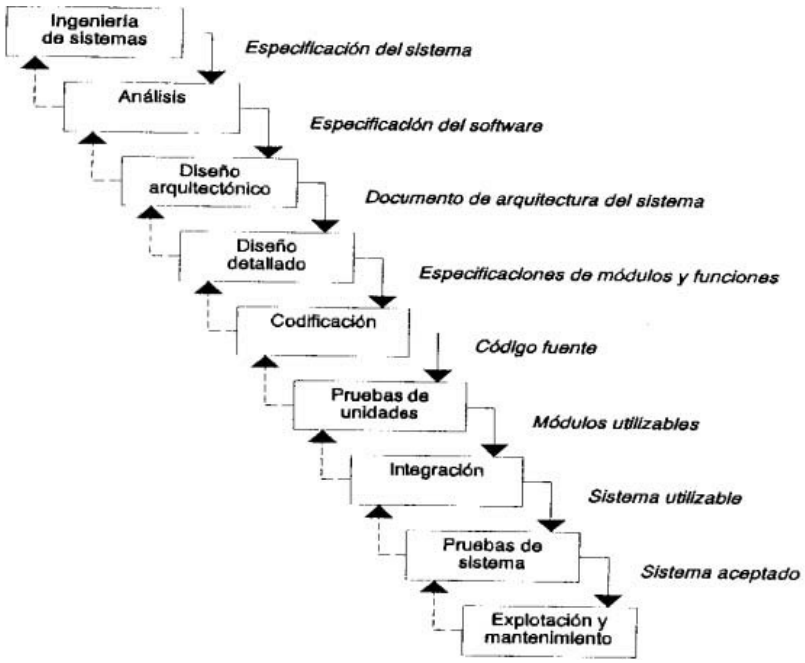
# 6. Modelos y Ciclos de Vida del Desarrollo de Software

La ingeniería de software, con el fin de ordenar el caos que era anteriormente el desarrollo de software, dispone de varios modelos, paradigmas y filosofías de desarrollo, estos los conocemos principalmente como modelos o ciclos de vida del desarrollo de software, esto incluye el proceso que se sigue para construir, entregar y hacer evolucionar el software, desde la concepción de una idea hasta la entrega y el retiro del sistema y representa todas las actividades y artefactos (productos intermedios) necesarios para desarrollar una aplicación, entre ellos se puede citar:

## 6.1. Modelo en cascada o clásico

En ingeniería de software el modelo en cascada —también llamado desarrollo en cascada o ciclo de vida clásico— se basa en un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, esto sugiere una aproximación sistemática secuencial hacia el proceso de desarrollo del software, que se inicia con la especificación de requerimientos del cliente y continúa con la planeación, el modelado, la construcción y el despliegue para culminar en el soporte del software terminado.




FASES

- ANÁLISIS. Analizar las necesidades de los usuarios potenciales del software. Determinar qué debe hacer el sistema a desarrollar. Escribir una especificación precisa de dicho sistema.
- DISEÑO. Descomponer y organizar el sistema en elementos componentes que puedan ser desarrollados por separado, aprovechando las ventajas de la división del trabajo. El resultado es la colección de especificaciones de cada elemento componente.
- CODIFICACIÓN. Se programa cada elemento componente por separado. Se harán pruebas para verificar que los componentes funcionan aisladamente.
- INTEGRACIÓN. Se combinan todos los componentes del sistema y se hacen pruebas del

sistema completo.
- MANTENIMIENTO. Realizar cambios ocasionales, corregir errores no detectados e introducir mejoras.

Se trata de aislar cada fase de la siguiente. En cada fase se genera una información de salida precisa y suficiente para que otras personas puedan acometer la fase siguiente.

Se suelen exigir los siguientes documentos:

- DOCUMENTO DE REQUESITOS DEL SOFTWARE (SRD: Software Requirements Document). Fase de análisis.
- DOCUMENTO DE DISEÑO DEL SOFTWARE (SDD: Software Design Document). Fase de diseño.
- CÓDIGO FUENTE. Fase de codificación.
- EL SISTEMA SOFTWARE. Fase de integración.
- DOCUMENTO DE CAMBIOS. Fase de mantenimiento.

Las variantes de este modelo se diferencian en el reconocimiento o no como fases separadas de ciertas actividades. Es necesario terminar correctamente cada fase antes de comenzar la siguiente.

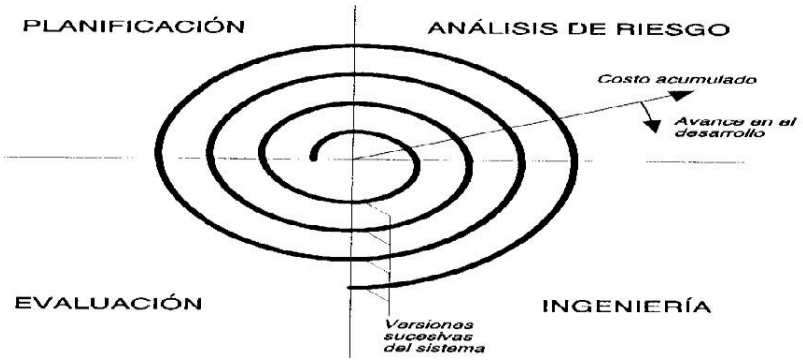
## 6.2. Modelo de prototipos

En ingeniería de software, el modelo de prototipos pertenece a los modelos de desarrollo evolutivo. Este permite que todo el sistema, o algunos de sus partes, se construyan rápidamente para comprender con facilidad y aclarar ciertos aspectos en los que se aseguren que el desarrollador, el usuario, el cliente estén de acuerdo en lo que se necesita así como también la solución que se propone para dicha necesidad y de esta manera minimizar el riesgo y la incertidumbre en el desarrollo, este modelo se encarga del desarrollo de diseños para que estos sean analizados y prescindir de ellos a medida que se adhieran nuevas especificaciones, es ideal para medir el alcance del producto, pero no se asegura su uso real.

Este modelo principalmente se aplica cuando un cliente define un conjunto de objetivos generales para el software a desarrollarse sin delimitar detalladamente los requisitos de entrada procesamiento y salida, es decir cuando el responsable no está seguro de la eficacia de un algoritmo, de la adaptabilidad del sistema o de la manera en que interactúa el hombre y la máquina.

Este modelo se encarga principalmente de ayudar al ingeniero de sistemas y al cliente a entender de mejor manera cuál será el resultado de la construcción cuando los requisitos estén satisfechos.

## 6.3. Modelo en espiral




Las **actividades de planificación** sirven para establecer el contexto del desarrollo y decidir qué parte del mismo se abordará en ese ciclo de la espiral.

El **análisis del riesgo**, consiste en evaluar diferentes alternativas para la realización de la parte del desarrollo elegida, seleccionando la más ventajosa, tomando precauciones para evitar los inconvenientes previstos.

Las **actividades de ingeniería**, son las indicadas en los modelos clásicos: análisis, diseño, codificación, etc. El resultado es ir obteniendo en cada ciclo una versión más completa del sistema.

Las **actividades de evaluación**, analizan los resultados de la fase de ingeniería, con la colaboración del "cliente". El resultado se utiliza como información de entrada para la planificación del ciclo siguiente.

## 6.4. Modelo de desarrollo por etapas

Es un modelo en el que el software se muestra al cliente en etapas refinadas sucesivamente. Con esta metodología se desarrollan las capacidades más importantes reduciendo el tiempo necesario para la construcción de un producto; el modelo de entrega por etapas es útil para el desarrollo de la herramienta debido a que su uso se recomienda para problemas que pueden ser tratados descomponiéndolos en problemas más pequeños y se caracteriza principalmente en que las especificaciones no son conocidas en detalle al inicio del proyecto y por tanto se van desarrollando simultáneamente con las diferentes versiones del código.

En este modelo pueden distinguirse las siguientes fases:

- Especificación conceptual.
- Análisis de requisitos.
- Diseño inicial.
- Diseño detallado (codificación, depuración, prueba y liberación).

Cuando es por etapas, en el diseño global estas fases pueden repetirse según la cantidad de etapas

que sean requeridas.

Entre sus ventajas tenemos:

- Detección de problemas antes y no hasta la única entrega final del proyecto.
- Eliminación del tiempo en informes debido a que cada versión es un avance.
- Estimación de tiempo por versión, evitando errores en la estimación del proyecto general.
- Cumplimiento a la fecha por los desarrolladores.

## 6.5. Modelo Incremental o Iterativo

Desarrollo iterativo y creciente (o incremental) es un proceso de desarrollo de software, creado en respuesta a las debilidades del modelo tradicional de cascada, es decir, este modelo aplica secuencias lineales como el modelo en cascada, pero de una manera iterativa o escalada según como avance el proceso de desarrollo y con cada una de estas secuencias lineales se producen incrementos (mejoras) del software.

Se debe tener en cuenta que el flujo del proceso de cualquier incremento puede incorporar el paradigma de construcción de prototipos, ya que como se mencionó anteriormente, este tipo de modelo es iterativo por naturaleza, sin embargo se diferencia en que este busca la entrega de un producto operacional con cada incremento que se le realice al software.

Este desarrollo incremental es útil principalmente cuando el personal necesario para una implementación completa no está disponible.

### 6.5.1 Modelo estructurado

Este modelo —como su nombre lo indica— utiliza las técnicas del diseño estructurado o de la programación estructuradapara su desarrollo, también se utiliza en la creación de los algoritmos del programa. Este formato facilita la comprensión de la estructura de datos y su control. Entre las principales características de este modelo se encuentan las siguientes:

- Generalmente se puede diferenciar de una manera más clara los procesos y las estructuras de datos.
- Existen métodos que se enfocan principalmente en ciertos datos.
- La abstracción del programa es de un nivel mucho mayor.
- Los procesos y estructuras de datos son representados jerárquicamente.

Este modelo también presenta sus desventajas entre las cuales podemos mencionar algunas:

- Se podía encontrar datos repetidos en diferentes partes del programa.
- Cuando el código se hace muy extenso o grande su manejo se complica demasiado.

En el modelo estructurado las técnicas que comúnmente se utilizan son:

- El modelo entidad-relación, esta técnica se relaciona principalmente con los datos.
- El diagrama de flujo de datos, esta es utilizada principalmente para los procesos.

### 6.5.2. Modelo orientado a objetos

Estos modelos tienen sus raíces en la programación orientada a objetos y como consecuencia de ella gira entorno al concepto de clase, también lo hacen el análisis de requisitos y el diseño. Esto además de introducir nuevas técnicas, también aprovecha las técnicas y conceptos del desarrollo estructurado, como diagramas de estado y transiciones. El modelo orientado a objetos tiene dos características principales, las cuales ha favorecido su expansión:

- Permite la reutilización de software en un grado significativo.
- Su simplicidad facilita el desarrollo de herramientas informáticas de ayuda al desarrollo, el cual es fácilmente implementada en una notación orientada a objetos llamado UML.

## 6.6. Modelo RAD (rapid application development)

El RAD (rapid application development: 'desarrollo rápido de aplicaciones'), es un modelo de proceso de software incremental, desarrollado inicialmente por James Maslow en 1980, que resalta principalmente un ciclo corto de desarrollo.

Esta es una metodología que posibilita la construcción de sistemas computacionales que combinen técnicas y utilidades CASE (Computer Aided Software Engineering), la construcción de prototipos centrados en el usuario y el seguimiento lineal y sistemático de objetivos, incrementando la rapidez con la que se producen los sistemas mediante la utilización de un enfoque de desarrollo basado en componentes.

Si se entienden bien los requisitos y se limita el ámbito del proyecto, el proceso RAD permite que un equipo de desarrollo cree un producto completamente funcional dentro de un periodo muy limitado de tiempo sin reducir en lo más mínimo la calidad del mismo.

## 6.7. Modelo de desarrollo concurrente

El modelo de desarrollo concurrente es un modelo de tipo de red donde todas las personas actúan simultáneamente o al mismo tiempo. Este tipo de modelo se puede representar a manera de esquema como una serie de actividades técnicas importantes, tareas y estados asociados a ellas.
En realidad, el modelo de desarrollo concurrente es aplicable a todo tipo de desarrollo de software y proporciona una imagen exacta del estado actual de un proyecto. En vez de confinar actividades de ingeniería de software a una secuencia de sucesos, define una red de actividades, todas las actividades de la red existen simultáneamente con otras. Los sucesos generados dentro de una actividad dada o algún otro lado de la red de actividad inicia las transiciones entre los estados de una actividad.

# 7. Producto

El software se ha convertido en algo muy necesario en nuestra sociedad actual, es la maquina que conduce a la toma de decisiones comerciales, sirve para la investigacion científica moderna, es un factor clave que diferencia productos y servicios modernos, etc. Esto se da porque el software está inmerso en sistemas de todo tipo alrededor de nosotros.

Los productos se pueden clasificar en:

- **Productos genéricos:** Son los producidos por una organización para ser vendidos al mercado.
- **Productos hechos a medida:** Sistemas que son desarrollados bajo pedido a un desarrollador específico.

Estos productos deben cumplir varias características al ser entregados, estas son:

- **Mantenibles:** El software debe poder evolucionar mientras cumple con sus funciones.
- **Confiabilidad:** No debe producir daños en caso de errores.
- **Eficiencia:** El software no debe desperdiciar los recursos.
- **Utilizacion adecuada:** Debe contar con una interfaz de usuario adecuada y su documentacion.

Lo que constituye el producto final es diferente para el ingeniero y los usuarios, para el ingeniero son los programas, datos y documentos que configuran el software pero para el usuario el producto final es la información que de cierto modo soluciona el problema planteado por el usuario.

# 8. Participantes y papeles

Para el desarrollo de un sistema de software es necesaria la colaboración de muchas personas con diversas competencias, capacidades e intereses. Al conjunto de personas involucradas en el proyecto se les conoce como participantes.

Al conjunto de funciones y responsabilidades que hay dentro del proyecto o sistema se le conoce como roles o papeles. Los roles están asociados a las tareas que son asignadas a los participantes, en consecuencia, una persona puede desempeñar uno o múltiples roles, así también un mismo rol puede ser representado por un equipo.

## 8.1. Cliente

Es frecuente el uso de "usuarios", "usuarios finales" y "clientes" como sinónimos, lo cual produce confusión. El cliente (persona, organización) es quién especifica los requerimientos del sistema.

## 8.2. Desarrolladores

Esta clase de participantes están relacionados con todas las facetas del proceso de desarrollo del software. Su trabajo incluye la investigacion, diseño, implementación, pruebas y depuracion del software.

## 8.3. Gestores

En el contexto de ingeniería de software, el gestor de desarrollo de software es un participante, que reporta al director ejecutivo de la empresa que presta el servicio de desarrollo. Es responsable del manejo y coordinación de los recursos y procesos para la correcta entrega de productos de software, mientras participa en la definición de la estrategia para el equipo de desarrolladores, dando iniciativas que promuevan la visión de la empresa.

## 8.4. Usuarios finales

El usuario final es quien interactúa con el producto de software una vez es entregado. Generalmente son los usuarios los que conocen el problema, ya que día a día operan los sistemas.

# 9. Fase del Diseño de Software

Las etapas de desarrollo se dedican a determinar CÓMO se debe resolver el proyecto.

El punto de partida es el documento de especificación de requisitos (SRD).

Es muy importante la experiencia previa, siempre que sea posible se tratará de reutilizar el mayor número posible de módulos o elementos ya desarrollados. Cuando no sea posible, por tratarse del diseño de un sistema completamente original, seguro que para comenzar el diseño, al menos se podrá aprovechar el enfoque dado a algún otro proyecto anterior. Se conoce como aprovechar el know-how (saber hacer). En sucesivas iteraciones se perfilará el enfoque más adecuado para el nuevo diseño.

La etapa de diseño es la más importante de la fase de desarrollo de software. En esta etapa se tiene que pasar de una forma gradual de QUÉ debe hacer el sistema, según se detalla en el documento de requisitos, al CÓMO lo debe hacer.

Durante el diseño se tiene que pasar de las ideas informales recogidas en el SRD a definiciones detalladas y precisas para la realización del software mediante refinamientos sucesivos.

## 9.1. Notaciones para el Diseño

El objetivo fundamental de cualquier notación es resultar precisa, clara y sencilla de interpretar en el aspecto concreto que describe. Evitar ambigüedades e interpretaciones erróneas del diseño.

Según el aspecto del sistema que se trata de describir es aconsejable emplear una u otra clase de notación.

Se pueden clasificar las notaciones en los siguientes grupos:

- Notaciones estructurales
- Notaciones estáticas o de organización
- Notaciones dinámicas o de comportamiento
- Notaciones híbridas

### 9.1.1 Notaciones estructurales

Sirven para cubrir un primer nivel del diseño arquitectónico. Se trata de desglosar y estructurar el sistema en sus partes fundamentales.

#### DIAGRAMAS DE ESTRUCTURA

#### DIAGRAMAS HIPO

#### DIAGRAMAS DE JACKSON

### 9.1.2 Notaciones estáticas

Sirven para describir caracteristicas estáticas del sistema, sin tener en cuenta su evolución durante el funcionamiento del sistema.

Como resultado del diseño se tendrá una organización de la información con un nivel de datalle mucho mayor. Las notaciones que se pueden emplear para describir el resultado de este trabajo son las mismas que se emplean para realizar la especificación.

#### DICCIONARIO DE DATOS

#### DIAGRAMAS ENTIDAD-RELACIÓN

### 9.1.3 Notaciones dinámicas

Permiten describir el comportamiento del sistema durante su funcionamiento. Al diseñar la dinámica del sistema se detallará su comportamiento externo y se añadirá la descripción de un comportamiento interno capaz de garantizar que se cumplen todos los requisitos especificados en el documento SRD.

#### DIAGRAMAS DE FLUJO DE DATOS

#### DIAGRAMAS DE TRANSICIÓN DE ESTADOS

#### LENGUAJE DE DESCRIPCIÓN DE PROGRAMAS (PDL)

### 9.1.4 Notaciones híbridas

Tratan de cubrir simultáneamente aspectos estructurales, estáticos y dinámicos. Metodologías de diseño basado en abastracciones y diseño orientado a objetos.

#### DIAGRAMA DE ABASTRACCIONES

#### DIAGRAMAS DE OBJETOS

## 9.2. Documentos de Diseño

El resultado principal de la labor realizada en la etapa de diseño se recoge en un documento que se utilizará como elemento de partida para las sucesivas etapas del proyecto, es el documento de diseño de software o Software Design Document (SDD)

Cuando la complejidad del sistema haga que el documento SDD resulte muy voluminoso y difícil de manejar es habitual utilizar dos o más documentos para describir de forma jerarquizada la estructura global del sistema.

Se establece el empleo de:

- Un **Documento de Diseño Arquitectónico o Architectural Design Document ADD**. Describe el sistema en su conjunto.

- Un **Documento de Diseño Detallado o Detailed Design Document DDD**. Describe por separado cada uno de los componentes del sistema.

# 10. Fase de Codificación

## 10.1. Programación

Una de las etapas que componen el ciclo de vida de una aplicación es la programación. Después de llevar a cabo un estudio detallado de toda la información, es decir, cuando ya se sabe qué es lo que se quiere hacer y cómo, se codifican las órdenes necesarias en un lenguaje de programación legible por el ordenador e inteligible por el programador.

Un lenguaje de programación es una notación para escribir programas a través de la cual podemos comunicarnos con el hardware y dar así las órdenes adecuadas para la realización de una determinada tarea.

Cuando se programa con un lenguaje de alto nivel, un software adicional tendrá que traducir esas instrucciones a lenguaje máquina.

Existen distintas formas de llevar a cabo esta traducción:

**A través de un compilador.**

Un compilador traduce un programa codificado en un lenguaje de alto nivel a instrucciones codificadas en lenguaje máquina.

**A través de un intérprete.**

Un intérprete detecta errores, traduce y ejecuta las instrucciones escritas en un lenguaje de alto nivel.

Estos programas de traducción también realizar el análisis lexicográficos, teniendo en cuenta el análisis sintácticos y semántico, detectando los errores que se hayan podido cometer al codificar el programa.

## 10.2. Documentación de los programas

El llevar a cabo la tarea de documentar un programa es muy importante puesto que una persona desconocedora de cualquier lenguaje de programación podría, a través de los comentarios, sobrentender el significado del programa, o incluso para el propio programador ya que al cabo de un tiempo puede tener que examinar el programa.

## 10.3. El Entorno de programación

Un entorno de programación está formado por el conjunto de instrumentos que facilitan o automatizan las actividades de desarrollo

Sería deseable automatizar todo el desarrollo, pero normalmente se automatiza sólo en parte. Da soporte a las actividades específicas de una fase del ciclo de vida: análisis de requisitos, diseño de la arquitectura, edición y compilación del código, etc. y a las actividades generales como documentación, planificación, gestión de configuración, etc.

### 10.3.1. Contenido

Un entorno de programación incluye un editor, un compilador, archivos de biblioteca, un enlazador y mucho más.

**Editor:** permite introducir y modificar el código fuente.

**Compilador:** Un programa que convierte el código fuente en un código que entiende el ordenador (código objeto).

**Archivos de biblioteca:** Son programas previamente compilados que realizan funciones específicas.

**Enlazador:** Esencialmente, el enlazador combina todas las partes necesarias, tales como archivos de biblioteca y el código compilado para producir el código ejecutable final.

### 10.3.2. Programa fuente, objeto, intermedio y ejecutable

#### Programa fuente, objeto y ejecutable

Nosotros escribimos un programa en un lenguaje de programación. Este programa se denomina programa fuente. A continuación, hay que traducir el programa fuente para generar un programa objeto. Esta traducción recibe el nombre de compilación.

Finalmente, para obtener un programa ejecutable se deben enlazar todos los archivos de código objeto con un programa llamado enlazador (linker).

Para facilitar el trabajo a los programadores existen programas llamados intérpretes que compilan, enlazan y ejecutan de manera automatizada.

# Ejercicios

- ¿Qué ciclo de vida elegiríamos para resolver un problema que se comprende bien desde el principio y está muy estructurado?
- Se supone que se va desarrollar una aplicación relativa a la gestión de pedidos de una empresa. En este caso el cliente no tiene todavía muy claro qué es lo que quiere. Además, el personal informático va a utilizar un tecnología que le resulta completamente nueva. Elige qué tipo de ciclo de vida es más apropiado y cuales serian los pasos a seguir.
- Indicar la(s) respuesta(s) correcta(s) y razonar la respuesta:

El ciclo de vida:

1. Comienza con una idea o necesidad que satisfacer y acaba con las pruebas satisfactorias del producto.
2. No existe ningún estándar que describa sus procesos y actividades.
3. No se trata sólo de realizar el análisis, diseño, codificación y pruebas; también incluye, entre otros, procesos de soporte.
4. El mantenimiento lo constituyen las actividades para mantener sin cambios el sistema.
5. En la actividad de análisis de los requisitos software los desarrolladores obtienen de los futuros usuarios los requisitos que piden al sistema.
6. La combinación de modelos puede ser ventajoso a la hora de trabajar con software para solucionar un problema.

- Busca en Internet las herramientas que se utilizan hoy en día en las empresas para el desarrollo de proyectos de software. Crear un documento con una breve descripción y debate con tus compañeros cúal puede ser la mejor opción para utilizar en una empresa.