TEMA 1. Desarrollo de un software

CICLOS DE VIDA DE UN SOFTWARE

1.- Modelo en Cascada

Es el modelo de vida clásico del software.

Es prácticamente imposible que se pueda utilizar, ya que requiere conocer de antemano todos los requisitos del sistema. Sólo es aplicable a pequeños desarrollos, ya que **las etapas pasan de una a otra sin retorno posible** (se presupone que no habrá errores ni variaciones del software).

2.- Modelo en Cascada con Realimentación

Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una **realimentación entre etapas**, de forma que podamos **volver atrás** en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.

Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

3.- Modelos Evolutivos

Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software. Distinguimos dos variantes:

3.1.- Modelo Iterativo Incremental

Está basado en el modelo en cascada con realimentación, donde **las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.** El proyecto se desarrolla en pequeñas porciones (incremental) en sucesivas iteraciones (sprints), al final de las cuales se puede ver lo que se ha desarrollado, y antes de comenzar la siguiente iteración (sprint) se pueden ver los requerimientos que no se conocían o estaban mal interpretados, o incluso introducir nuevos requerimientos (adaptativo). Cada sprint debe proporcionar un resultado completo (un incremento de producto final) preparado para entregárselo al cliente.

3.2.- Modelo en Espiral

Es una combinación del modelo anterior con el modelo en cascada. En él, el software se va construyendo repetidamente en forma de **versiones que son cada vez mejores**, debido a que incrementan la funcionalidad en cada versión. Es un modelo bastante complejo.

HERRAMIENTAS DE APOYO AL DESARROLLO SOFTWARE – CASE

Las herramientas **CASE** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.

¿En qué fases del proceso nos pueden ayudar? En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores…

El desarrollo rápido de aplicaciones o **RAD** es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario o entornos de desarrollo integrado completos.

La tecnología CASE trata de automatizar las fases del desarrollo de software para que mejore la calidad del proceso y del resultado final. En concreto, estas herramientas permiten:

- Mejorar la planificación del proyecto. Darle agilidad al proceso.
- Poder reutilizar partes del software en proyectos futuros. Hacer que las aplicaciones respondan a estándares.
- Mejorar la tarea del mantenimiento de los programas.
- Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

CLASIFICACIÓN

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- **U-CASE:** ofrece ayuda en las fases de planificación y análisis de requisitos.
- **M-CASE**: ofrece ayuda en análisis y diseño.
- **L-CASE**: ofrece ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas, y en la generación de la documentación del proyecto.

Por funcionalidad se pueden diferenciar algunas como:

- Herramientas de generación semiautomática de código. Editores UML.
- Herramientas de refactorización de código.
- Herramientas de mantenimiento como los sistemas de control de versiones.

Ejemplos de herramientas **CASE libres** son: ArgoUML, Use Case Maker, ObjectBuilder…

LENGUAJES DE PROGRAMACIÓN

Características de los Lenguajes de Programación Lenguaje máquina:

- Sus instrucciones son combinaciones de unos y ceros.
- Es el único lenguaje que entiende directamente el ordenador (no necesita traducción). Fue el primer lenguaje utilizado.
- Es único para cada procesador (no es portable de un equipo a otro). Hoy día nadie programa en este lenguaje.

Lenguaje ensamblador:

- Sustituyó al lenguaje máquina para facilitar la labor de programación.
- En lugar de unos y ceros se programa usando mnemotécnicos (instrucciones complejas). Necesita traducción al lenguaje máquina para poder ejecutarse.
- Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo. Es difícil de utilizar.

Lenguaje de alto nivel basados en código:

- Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación.
- En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del idioma inglés. Necesita traducción al lenguaje máquina.
- Son más cercanos al razonamiento humano.
- Son utilizados hoy día, aunque la tendencia es que cada vez menos.

Lenguajes visuales:

- Están sustituyendo a los lenguajes de alto nivel basados en código.
- En lugar de sentencias escritas, se programa gráficamente usando el ratón y diseñando directamente la apariencia del software.
- Su correspondiente código se genera automáticamente. Necesitan traducción al lenguaje máquina.
- Son portables de un equipo a otro.

LENGUAJES DE PROGRAMACIÓN ESTRUCTURADOS

La programación estructurada se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

- Sentencias secuenciales.
- Sentencias selectivas (condicionales).
- Sentencias repetitivas (iteraciones o bucles).

VENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA

Los programas son fáciles de leer, sencillos y rápidos. El mantenimiento de los programas es sencillo.

La estructura del programa es sencilla y clara.

INCONVENIENTES

Todo el programa se concentra en un único bloque (si se hace demasiado grande es difícil manejarlo).

No permite reutilización eficaz de código, ya que todo va "en uno". Es por esto que a la programación estructurada le sustituyó la programación modular, donde los programas se codifican por módulos y bloques, permitiendo mayor funcionalidad.

Ejemplos: Pascal, C, Fortran.

LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

Los lenguajes de programación orientados a objetos tratan a los programas no como un conjunto ordenado de instrucciones (tal como sucedía en la programación estructurada), sino como un conjunto de objetos que colaboran entre ellos para realizar acciones

Su primera desventaja es clara: no es una programación tan intuitiva como la estructurada. A pesar de eso, alrededor del 55% del software que producen las empresas se hace usando esta técnica.

Razones:

- El código es reutilizable.
- Si hay algún error, es más fácil de localizar y depurar en un objeto que en un programa entero.

Características:

- Los objetos del programa tendrán una serie de atributos que los diferencian unos de otros.
- Se define clase como una colección de objetos con características similares.
- Mediante los llamados métodos, los objetos se comunican con otros produciéndose un cambio de estado de los mismos.
- Los objetos son, pues, como unidades individuales e indivisibles que forman la base de este tipo de programación.

Principales lenguajes orientados a objetos: Ada, C++, VB.NET, Delphi, Java, PowerBuilder, etc.

FASES EN EL DESARROLLO Y EJECUCIÓN DEL SOFTWARE

1. ANÁLISIS DE REQUISITOS.

Se especifican los requisitos funcionales y no funcionales del sistema. Es la primera fase del proyecto, es la de mayor importancia y todo lo demás dependerá de lo bien detallada que esté. La culminación de esta fase es el **documento ERS** (Especificación de Requisitos Software)

Requisitos:

Funcionales: Qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.

No funcionales: Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

2. DISEÑO.

Se divide el sistema en partes y se determina la función de cada una.

Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas. Decidir qué hará exactamente cada parte. En definitiva, debemos crear un modelo funcional- estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.

En este punto, se deben tomar decisiones importantes, tales como:

- Entidades y relaciones de las bases de datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del SGBD.
- Definición de diagrama de clases. Definición de diagrama de colaboración.
- Definición de diagrama de paso de mensajes. Definición de diagrama de casos de uso

3. CODIFICACIÓN.

Se elige un Lenguajes de Programación y se codifican los programas.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

1. Modularidad: que esté dividido en trozos más pequeños.
2. Corrección: que haga lo que se le pide realmente.
3. Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
4. Eficiencia: que haga un buen uso de los recursos.
5. Portabilidad: que se pueda implementar en cualquier equipo.

Durante esta fase, el código pasa por diferentes estados:

- **Código Fuente:** es el escrito por los programadores en algún editor de texto. Se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesarias.

- **Código Objeto:** es el código binario (bytecode) resultado de compilar el código fuente. No puede ser ejecutado por la computadora.
  La compilación es la traducción de una sola vez del programa, y se realiza utilizando un compilador. La interpretación es la traducción y ejecución simultánea del programa línea a línea.
  El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora. Es un código intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso.

Código Ejecutable: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. También es conocido como código máquina y ya sí es directamente inteligible por la computadora. Para obtener un sólo archivo ejecutable, habrá que enlazar todos los archivos de código objeto, a través de un software llamado **linker** (enlazador) y obtener así un único archivo que ya sí es ejecutable directamente por la computadora.

MÁQUINAS VIRTUALES

Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados. Esta capa de software desempeña un papel muy importante en el funcionamiento de los lenguajes de programación, tanto compilados como interpretados.

Con el uso de máquinas virtuales podremos desarrollar y ejecutar una aplicación sobre cualquier equipo, independientemente de las características concretas de los componentes físicos instalados. Esto garantiza la portabilidad de las aplicaciones.

Las funciones principales de una máquina virtual son las siguientes:

- Conseguir que las aplicaciones sean **portables**.
- Reservar memoria para los objetos que se crean **y liberar la memoria** no utilizada.
- **Comunicarse con el sistema** donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos.
- Cumplimiento de las **normas de seguridad de las aplicaciones**.

Características de las máquinas virtuales:

- La máquina virtual **aísla la aplicación de los detalles físicos del equipo** en cuestión. Cuando el código fuente se compila se obtiene código objeto (bytecode, código intermedio). Para ejecutarlo en cualquier máquina se requiere tener independencia respecto al hardware concreto que se vaya a utilizar. La máquina virtual funciona como una capa de software de bajo nivel y actúa como puente entre el bytecode de la aplicación y los dispositivos físicos del sistema.
- La máquina virtual **verifica todo el bytecode** antes de ejecutarlo.
- La máquina virtual **protege direcciones de memoria.**

FRAMEWORKS

Un framework es una estructura de ayuda al programador, en base a la cual podemos

desarrollar proyectos sin partir desde cero. Se trata de una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.

Con el uso de framework podemos pasar más tiempo analizando los requerimientos del sistema y las especificaciones técnicas de nuestra aplicación, ya que la tarea laboriosa de los detalles de programación queda resuelta.

Ventajas de utilizar un framework:

- Desarrollo rápido de software.
- Reutilización de partes de código para otras aplicaciones.
- Diseño uniforme del software.
- Portabilidad de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.

Inconvenientes:

- Gran dependencia del código respecto al framework utilizado (sin cambiamos de framework, habrá que reescribir gran parte de la aplicación).
- La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.

Ejemplos: .Net, Spring de Java.

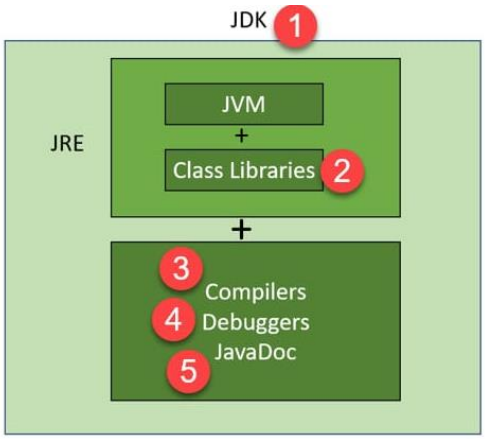
JAVA RUNTIME ENVIROMENT

Se denomina JRE al Java Runtime Environment (entorno en tiempo de ejecución Java).

El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma.

JRE está formado por:

- Una máquina virtual Java (JMV o JVM si consideramos las siglas en inglés), que es el programa que interpreta el código de la aplicación escrito en Java.
- Bibliotecas de clase estándar que implementan el API de Java.
- Las dos: JMV y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.



4. PRUEBAS.

Se prueban los programas para detectar errores y se depuran. Normalmente, éstas se realizan sobre un conjunto de datos de prueba, que consisten en un conjunto seleccionado y predefinido de datos límite a los que la aplicación es sometida.

La realización de pruebas es imprescindible para asegurar la construido.

Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

PRUEBAS UNITARIAS

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). JUnit es el entorno de pruebas para Java.

PRUEBAS DE INTEGRACIÓN

Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.

La prueba final se denomina comúnmente **Beta Test**, ésta se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa).

5. DOCUMENTACIÓN.

De todas las etapas, se documenta y guarda toda la información.

Tenemos que ir documentando el proyecto en todas las fases del mismo, para pasar de una a otra de forma clara y definida. Una correcta documentación permitirá la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

Los documentos a elaborar en el proceso de desarrollo de software es la **guía técnica, guía de uso y la guía de instalación.**

6. EXPLOTACIÓN.

Instalamos, configuramos y probamos la aplicación en los equipos del cliente.

- En el proceso de **instalación**, los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados. Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación.
  En este momento, se suelen llevan a cabo las Beta Test, que son las últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo.
- Una vez instalada, pasamos a la fase de configuración. En ella, asignamos los parámetros de funcionamiento normal de la empresa y probamos que la aplicación es operativa. También puede ocurrir que la **configuración** la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación. Y también, si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla. (Si el software es "a medida", lo más aconsejable es que la hagan aquellos que la han fabricado).
- Una vez se ha configurado, el siguiente y último paso es la fase de **producción normal.** La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.

7. MANTENIMIENTO.

Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro. Es la más larga de todo el ciclo de vida del software.

El mantenimiento se define como el proceso de control, mejora y optimización del software. Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo.

Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

- **Perfectivos**: Para mejorar la funcionalidad del software.
- **Evolutivos**: El cliente tendrá en el futuro nuevas necesidades. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- **Adaptativos**: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- **Correctivos**: La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).

TEMA 2. Entornos de desarrollo multiplataforma y web

GESTIÓN DE MÓDULOS

Un módulo es un componente software que contiene clases de Java que pueden interactuar con las API del entorno de desarrollo y el manifest file, que es un archivo especial que lo identifica como módulo.

Hay dos formas de añadir módulos y plugins en Netbeans:

1. **Off-line:** Buscar y descargar plugins desde la página web oficial de la plataforma e instalarlo en el IDE.
2. **On-line:** consiste en instalar complementos desde nuestro mismo IDE, sin tener que descargarlos previamente (requiere tener conexión a internet).

A la hora de eliminar un módulo, tenemos dos opciones:

1. Desactivarlo: El módulo o plugin sigue instalado, pero en estado inactivo (no aparece en el entorno).

2. Desinstalarlo: El módulo o plugin se elimina físicamente del entorno de forma permanente.

HERRAMIENTAS CASE

Las herramientas CASE (Computer Aided Software Engineering) son aplicaciones software que nos facilitan el desarrollodel software, reduciendo el tiempo y el coste del mismo. Por ejemplo son las herramientas de los navegadores o el inspector del navegador.