EEDD01

Desarrollo de software

# Software y programa. Tipos de software

Hay tres tipos de software:

1. Software de sistema
2. Software de programación
3. Aplicaciones

El Software de sistema es el que debe estar instalado en el ordenador para que las aplicaciones puedan ejecutarse, como sistema operativo y drivers.

El software de programación son los editores de texto, intérpretes, compiladores etc...

Las aplicaciones son programas con una finalidad concreta.

# 2. Relación hardware-software

El software se ejecuta sobre dispositivos físicos (hardware) y utilizan recursos como tiempo de CPU, memoria, dispositivos entrada/salida etc....

# 3. Fases a seguir en el desarrollo del software

|  |  |
| --- | --- |
| Fase | Tareas |
| Análisis | Analizar los requisitos de la aplicación. |
| Diseño | Se divide el sistema en partes y se determina la función de cada una. Determinar herramientas a utilizar. |
| Codificación y compilación | Una vez sabemos cómo tiene que ser el programa, lo escribimos y compilamos. |
| Pruebas | Comprobamos que funciona. |
| Verificación en cliente/explotación | El usuario final comprueba que cumple con lo pactado. |
| Mantenimiento | Eventualmente habrá que añadir/actualizar cosas. |
| Documentación | Se hace a lo largo de todo el proceso. Se documenta y guarda toda la información. |

## 3.1. Análisis.

Primera fase del proyecto y la de mayor importancia. Se consigue lo siguiente:

* **Documento de especificación de requisitos**.
  + **Requisitos funcionales:** ¿qué funciones debe realizar la aplicación?
  + **No funcionales:** tiempos de respuesta, legislación aplicable, peticiones simultáneas etc...

|  |  |
| --- | --- |
| **Funcionales** | **No funcionales** |
| Lectura de productos mediante código de barras | Los PC suministrados deben ser azules |
| Detallar facturas de compra y formatos. | Habrá que garantizar soporte 24/7 365 d/a. |
| Información de las venas de cada vendedor para comisiones. | Deberá cumplir la legislación de protección de datos. |
| Control de stock | Se debe seguir el desarrollo de acuerdo con algún tipo de certificación. |
| Interfaz sencilla. |  |

* **Documento de diseño de arquitectura:** descripción de la estructura del sistema, cómo interactúan las partes etc...

También se tendrán en cuenta cosas como:

* Planificación de las reuniones que se tendrán.
* Relación de los objetivos del cliente y sistema.
* Objetivos primarios y temporalización.
* Mecanismos de actuación ante contingencias etc...

## 3.2. Diseño

En esta fase se toman decisiones importantes como:

* Entidades y relaciones de bases de datos.
* Selección del los lenguajes de programación a uilizar.
* Selección de SGBD.

En esta fase se obtienen 2 cosas.

* Documentos del diseño del software.
* Plan de pruebas.

## 3.3. Codificación.

Escribir el programa. Las características deseables del código son:

1. Modularidad: está dividido en muchas piezas pequeñas.
2. Corrección: que haga lo que realmente se pide.
3. Fácil de leer: para facilitar el mantenimiento.
4. Eficiencia: que haga buen uso de los recursos.
5. Portabilidad: que se pueda implementar en cualquier equipo.

## 3.4. Compilación

Convertir el código fuente a código máquina. Se puede hacer de dos formas:

* **Compilación:** se traduce el código por completo y se puede ejecutar al final.
* **Interpretación:** se traduce el código línea a línea y se ejecuta simultáneamente.

## 3.5. Pruebas.

Hay dos tipos:

* **Pruebas unitarias:** se prueba cada módulo/función por separado.
* **Pruebas de integración:** se comprueba que el programa en su conjunto funciona.

## 3.6. Explotación.

Cuando se implementa la aplicación y los usuarios empiezan a usarla. Se instala y configura en los equipos de los usuarios, o se les deja hacerlo a ellos dándoles una guía.

## 3.7. Mantenimiento.

La etapa más larga del ciclo de vida del software. Se arreglan bugs, se añaden mejoras, se adecua a hardware moderno etc... Hay estos tipos de cambios:

* Perfectivos, para mejorar la funcionalidad.
* Evolutivos, mejoras para el producto.
* Adaptativos, se adaptan a las nuevas tendencias del mercado, hardware etc...
* Correctivos, corregir bugs.

## 3.8. Documentación.

|  |  |
| --- | --- |
| Guías técnicas, para el personal informático | |
| Aspectos que quedan reflejados: | Diseño de la aplicación.  Codificación de los programas.  Pruebas realizadas. |
| ¿Objetivo? | Facilitar el desarrollo y mantenimiento para programadores futuros. |
| Guías de uso, para los usuarios (clientes) | |
| Aspectos que quedan reflejados: | Descripción de la funcionalidad de la aplicación.  Forma de iniciar la aplicación.  Ejemplos de uso.  Requerimientos de software de la aplicación.  Solución a posibles problemas. |
| ¿Objetivo? | Dar a los usuarios finales la información necesaria para usar la aplicación |
| Guía de instalación, para el personal informático responsable de la instalación. | |
| Aspectos que quedan reflejados: | Puesta en marcha.  Explotación.  Seguridad del sistema |
| ¿Objetivo? | Dar la información para que la instalación se realice de forma segura. |

# 4. Ciclos de vida del software.

Hay varios modelos:

* **Modelo en cascada:** es el modelo clásico. Se pasa de una fase a otra sin posiblidad de volver atrás.
  + - Requiere conocimientos previos del sistema.
    - No se puede volver atrás.
    - No permite modificaciones ni actualizaciones.
    - Es un modelo utópico.
* **Modelo en cascada con retroalimentación:** de los más utilizados. Como el anterior, pero se puede volver atrás. No recomendable si se prevén muchos cambios.
  + - Modelo más utilizado.
    - Se pueden depurar errores de fases anteriores.
    - Idóneo para proyectos rígidos con requisitos claros.
    - Puede que haya que empezar de cero.
* **Modelos evolutivos:** Hay dos variantes:
  + **Modelo iterativo incremental:** basado en el modelo de cascada con retroalimentación. Se van generando versiones parciales que cumplen los requisitos y cada versión es el punto de partida de la siguiente.
    - No ponen en el mercado productos completos, sino versiones.
    - Permite una evolución temporal.
    - Son varios ciclos en cascada que se repiten y refinan.
    - Las versiones cada vez son más completas hasta llegar a la versión final.
  + **Modelo en espiral:** modelo anterior + modelo en cascada.
    - Hay 6 zonas: comunicación con el cliente, planificación, análisis de riesgos, representación de la aplicación, codificación y explotación y evaluación del cliente.
    - Se adapta a la naturaleza evolutiva del software.
    - Reduce riesgos antes de que sean problemáticos.
  + **Modelos ágiles:** metodología Agile, como Scrum.

# 5. Herramientas de apoyo al desarrollo del software.

Las herramientas **CASE (Computer Aided Software Engineering)** ofrecen ayuda durante el desarrollo. Hay varios tipos:

* **U-CASE:** ayuda en la fase de planificación.
* **M-CASE:** ayudan en análisis y diseño.
* **L-CASE:** ayudan en la programación, depuración y pruebas.

# 6. Frameworks.

Estructuras que ayudan al programador para que no tengan que partir de cero.

|  |  |
| --- | --- |
| Ventajas:   * Desarrollo rápido. * Reutilización de código. * Diseño uniforme. * Portabilidad. | Inconvenientes:   * Dependencia del código del framework. * Consumo de recursos del sistema. |

Ejemplos de frameworks serían .NET, Java Spring, Angular o Qt.

# 7. Lenguajes de programación.

Un lenguaje de programación es un conjunto de:

* Alfabeto: caracteres y símbolos permitidos.
* Sintaxis: normas de construcción permitidas.
* Semántica: significado de las construcciones para hacer acciones válidas.

Ordenándolos por nivel, irían:

Lenguaje máquina -> Lenguaje ensamblador -> Lenguaje de alto nivel -> Lenguaje visual

## 7.1. Características de los lenguajes de programación:

* **Lenguaje máquina:**
  + Código binario.
  + No necesita traducción.
  + Único para cada procesador.
  + Nadie lo usa.
* **Lenguaje ensamblador:**
  + Sustituyó al lenguaje máquina.
  + Usa instrucciones complejas en vez de código binario.
  + Tiene que ser traducido a lenguaje máquina.
  + Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo.
* **Lenguaje de alto nivel basado en código:**
  + Sustituyeron al ensamblador.
  + Lenguaje humano.
  + Portables de un equipo a otro.
  + Requieren traducción.

## 7.2. Clasificación de los lenguajes de programación

Según su cercanía al lenguaje humano:

1. Alto nivel (lenguaje humano).
2. Bajo nivel:
   1. Lenguaje máquina.
   2. Lenguaje ensamblador.

Según su forma de ejecución:

1. Lenguajes compilados: lenguajes que han de ser traducidos y crear un archivo ejecutable antes de ser ejecutados.
2. Lenguajes interpretados: no hace falta traducirlos, tienen un intérprete. No hace falta generar archivos ejecutables.

Algunos lenguajes como Java hacen una combinación de ambos métodos. El código fuente se escribe en archivos .java que hacen una traducción (cuasi compilación) a un lenguaje intermedio que crea archivos .class. Después, el archivo .class es interpretado en cada ejecución.

|  |  |
| --- | --- |
| **Código fuente** | Lo que se escribe en el IDE/Editor de texto. |
| **Código objeto** | Código intermedio entre el lenguaje de alto nivel y el código máquina. |
| **Código ejecutable** | Código entendible por la máquina. |

**Lenguajes imperativos vs declarativos:**

En los lenguajes imperativos se escriben, paso a paso, las instrucciones que deben ejecutarse.

En los lenguajes declarativos se describe el problema que se quiere solucionar pero no las instrucciones para hacerlo.

**Según la técnica de programación utilizada:**

* **Lenguajes estructurados**.
* **Lenguajes orientados a objetos.**
* **Lenguajes de programación visuales.**

### 7.2.1. Lenguajes de programación estructurados.

El programa se divide en funciones o módulos pero es un único bloque. Son más fáciles de leer y entender, pero es más difícil reutilizar código o hacer cambios.

## 7.2.2. Lenguajes de programación orientada a objetos.

El programa se divide en clases y objetos. Es el método más popular a día de hoy, pero aunque son más fáciles de mantener y actualizar, es más difícil leerlos y entenderlos.

# 8. Máquinas virtuales.

Emulación de un sistema operativo dentro de otro. Pueden ser máquinas virtuales que emulan el sistema operativo completo, como tener Linux en una máquina de Virtualbox sobre Windows, o simplemente un “runtime” que sólo cargan las funcionalidades necesarias para ejecutar un programa, como los runtime de Java o .Net.

## 8.1. Entornos de ejecución.

Lo que dijimos antes de los runtime. Sólo se cargan una serie de módulos para que el programa que escribimos no dependa del sistema operativo sobre el que se ejecuta.

## 8.3. Java Runtime Environment (JRE)

Entorno de ejecución de Java. Está formado por la máquina virtual de Java (JVM) y la API de Java.