1. Software y programa. Tipos de software.

El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.

Según su función se distinguen tres tipos de software:

- **Sistema Operativo:** Es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar.
- **Software de programación:** Es el conjunto de herramientas que nos permiten desarrollar programas informáticos.
- **Aplicaciones informáticas:** Son un conjunto de programas que tienen una finalidad más o menos concreta. A su vez, un programa es un conjunto de instrucciones escritas en un lenguaje de programación.

2. Relación hardware-software.

La primera arquitectura hardware con programa almacenado se estableció en 1946 por John Von Neumann.

Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:

- **Desde el punto de vista del sistema operativo:** El sistema operativo es el encargado de coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.
- **Desde el punto de vista de las aplicaciones:** Una aplicación no es otra cosa que un conjunto de programas, y éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar. Hay multitud de lenguajes de programación diferentes, sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, el hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión) que en informática se traducen en secuencias de 0 y 1 (código binario). Tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

3. Desarrollo de software.

Es todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

1. Ciclos de vida del software.

Es la serie de pasos a seguir para desarrollar un programa. Los más conocidos y utilizados son los que aparecen a continuación:

- **Modelo en Cascada o clásico:** 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. Las etapas pasan de una a otra sin retorno posible. (se presupone que no habrá errores ni variaciones del software).
- **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. Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

- **Modelos Evolutivos:** Tienen en cuenta la naturaleza cambiante y evolutiva del software. Distinguimos dos variantes:
  - **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.
  - **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.

2. Herramientas de apoyo al desarrollo del software.

Para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.

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 automatizando las fases del desarrollo del software mejorando por tanto la productividad.

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.

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.

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:** 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.

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

4. Lenguajes de programación.

Un Lenguaje de Programación es un idioma creado de forma artificial cuyo objetivo es obtener un código que el hardware de la computadora pueda entender y ejecutar. Son los instrumentos que tenemos para que el ordenador realice las tareas que necesitamos.

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 completamente portables de un equipo a otro.

1. Concepto y características.

La elección del lenguaje de programación para codificar un programa dependerá de las características del problema a resolver.

Podemos clasificar los distintos tipos de Lenguajes de Programación según distintas características:

- **Según lo cerca que esté del lenguaje humano**
  - *Lenguajes de Programación De alto nivel:* por su esencia, están más próximos al razonamiento humano.
  - *Lenguajes de Programación De bajo nivel:* están más próximos al funcionamiento interno de la computadora:
    - Lenguaje Ensamblador.
    - Lenguaje Máquina.

- **Según la técnica de programación utilizada:**
  - *Lenguajes de Programación Estructurados:* Usan la técnica de programación estructurada. Ejemplos: Pascal, C, etc.
  - *Lenguajes de Programación Orientados a Objetos:* Usan la técnica de programación orientada a objetos. Ejemplos: C++, Java, Ada, Delphi, etc.
  - *Lenguajes de Programación Visuales:* Basados en las técnicas anteriores, permiten programar gráficamente, siendo el código correspondiente generado de forma automática. Ejemplos: Visual Basic.Net, Borland Delphi, etc.

2. *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:

- 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 de lenguajes estructurados: Pascal, C, Fortran.

3. *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. Los objetos son reutilizables para proyectos futuros.

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 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.

5. Fases o etapas en el desarrollo y ejecución del software.

Ya hemos visto en puntos anteriores que debemos elegir un modelo de ciclo de vida para el desarrollo de nuestro software. Independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad. Estas etapas son:

- **Análisis de los requisitos:** Se especifican los requisitos funcionales y no funcionales del sistema.
- **Diseño:** Se divide el sistema en partes y se determina la función de cada una.
- **Codificación:** Se elige un Lenguajes de Programación y se codifican los programas.
- **Pruebas:** Se prueban los programas para detectar errores y se depuran.
- **Documentación:** De todas las etapas, se documenta y guarda toda la información.
- **Explotación:** Instalamos, configuramos y probamos la aplicación en los equipos del cliente.
- **Mantenimiento:** Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

1. Análisis.

Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. En esta fase se especifican y analizan los requisitos funcionales y no funcionales del sistema.

- **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.
- **Requisitos no funcionales:**
  - Tiempos de respuesta del programa.
  - Legislación aplicable.
  - Tratamiento ante la simultaneidad de peticiones, etc.

Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.

La culminación de esta fase es el documento ERS (Especificación de Requisitos Software). En este documento quedan especificados:

- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del usuario cliente y del sistema.

- Relación de los requisitos funcionales y no funcionales del sistema.
- Relación de objetivos prioritarios y temporización.
- Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.

2. Diseño.

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es ¿Cómo hacerlo?

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 Sistema Gestor de Base de Datos.
- Etc.

3. Codificación. Tipos de código.

Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente. 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:

- **Modularidad:** que esté dividido en trozos más pequeños.
- **Corrección:** que haga lo que se le pide realmente.
- **Fácil de leer:** para facilitar su desarrollo y mantenimiento futuro.
- **Eficiencia:** que haga un buen uso de los recursos.
- **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 resultado de compilar el código fuente. 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. También es conocido como código máquina y ya sí es directamente inteligible por la computadora.

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.

4. Fases en la obtención de código.

1. Fuente

El código fuente es el conjunto de instrucciones que la computadora deberá realizar escritas por los programadores en algún lenguaje de alto nivel.

Un aspecto muy importante en esta fase es la elaboración previa de un algoritmo, que lo definimos como un conjunto de pasos a seguir para obtener la solución del problema.

Para obtener el código fuente de una aplicación informática:

- Se debe partir de las etapas anteriores de análisis y diseño.
- Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.
- Se elegirá un Lenguaje de Programación de alto nivel apropiado para las características del software que se quiere codificar.
- Se procederá a la codificación del algoritmo antes diseñado.
- La culminación de la obtención de código fuente es un documento con la codificación de todos los módulos, funciones, bibliotecas y procedimientos necesarios para codificar la aplicación.

Un aspecto importante a tener en cuenta es su licencia. Podemos distinguir dos tipos de código fuente:

- **Código fuente abierto:** Es aquel que está disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
- **Código fuente cerrado:** Es aquel que no tenemos permiso para editarlo.

2. Objeto.

Es el resultado de traducir código fuente a un código equivalente formado por unos y ceros que aún no puede ser ejecutado directamente por la computadora, es decir, es el código resultante de la compilación del código fuente.

El proceso de traducción de código fuente a código objeto puede realizarse de dos formas:

- **Compilación:** El proceso de traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá que enlazar. El software responsable se llama compilador.
- **Interpretación:** El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente.

3. Ejecutable.

El código ejecutable es el resultado de enlazar los archivos de código objeto por un software llamado linker formando un único archivo que puede ser directamente ejecutado por la computadora. No necesita ninguna aplicación externa. Este archivo es ejecutado y controlado por el sistema operativo.

Generación de ejecutables:

- A partir de un editor, escribimos el lenguaje fuente con algún Lenguaje de programación. (En el ejemplo, se usa Java).

- A continuación, el código fuente se compila obteniendo código objeto o bytecode.
- Ese bytecode, a través de la máquina virtual (se verá en el siguiente punto), pasa a código máquina, ya directamente ejecutable por la computadora.

5. Máquinas virtuales.

Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

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:

- 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.
- Para ello, la máquina virtual aísla la aplicación de los detalles físicos del equipo en cuestión.
- 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.
- La máquina virtual actúa de puente entre la aplicación y el hardware concreto del equipo donde se instale.

1. Frameworks.

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.

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 (si 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.

2. Entornos de ejecución.

Un entorno de ejecución es un servicio de máquina virtual que sirve como base para la ejecución de programas.

Funcionamiento:

- El Entorno de Ejecución está formado por la máquina virtual y las API's (bibliotecas de clases estándar, necesarias para que la aplicación, escrita en algún Lenguaje de Programación pueda ser ejecutada). Estos dos componentes se suelen distribuir conjuntamente, porque necesitan ser compatibles entre sí.
- El entorno funciona como intermediario entre el lenguaje fuente y el sistema operativo, y consigue ejecutar aplicaciones.

3. Java runtime environment.

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.

6. Pruebas.

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).

7. Documentación.

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas para dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

Distinguimos **tres grandes documentos** en el desarrollo de software:

- **Guía técnica**
- **Guía de uso**

- **Guía de instalación**

8. Explotación.

La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla. Se compone de la **instalación, puesta a punto y funcionamiento** de la aplicación en el equipo final del cliente.

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.

9. Mantenimiento.

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).