ENTORNOS DE DESARROLLO

Dpto. Informática y comunicaciones.
Curso. 2024 / 2025
José Luís Casado Valero

Virrey Morcillo
Instituto de Educación Secundaria

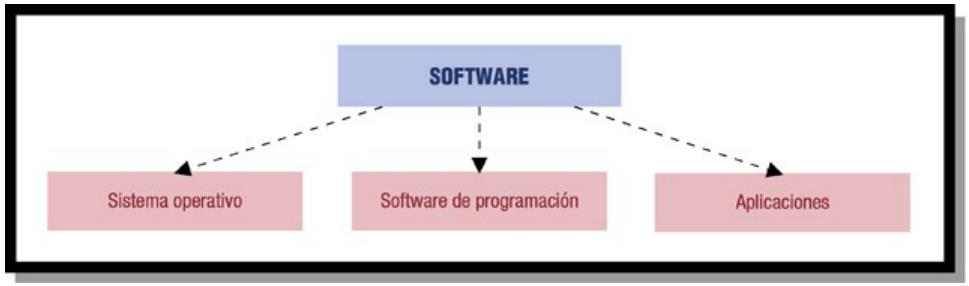
1 Software y programa. Tipos de software.

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

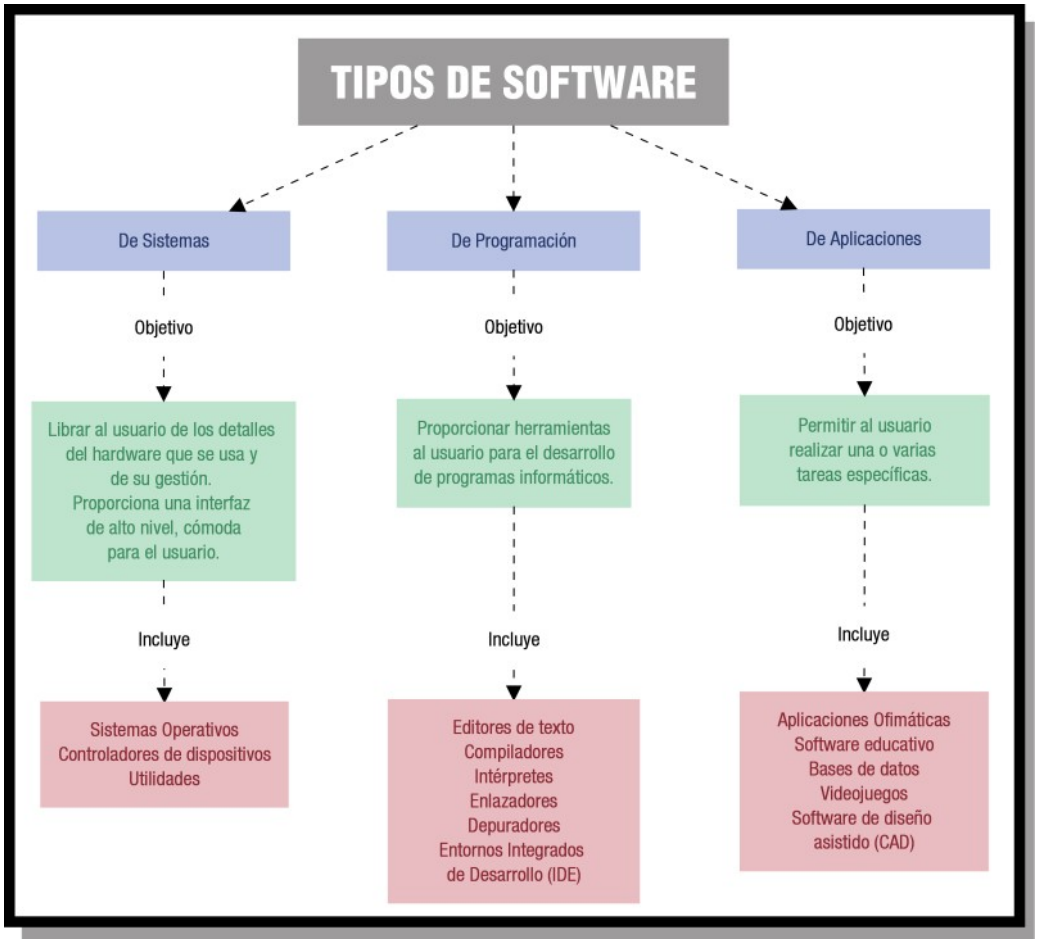
Según su función se distinguen **tres tipos de software**: sistema operativo, software de programación y aplicaciones.

El **sistema operativo** es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. Son ejemplos de sistemas operativos: Windows, Linux, Mac OS X ...

El **software de programación** es el conjunto de herramientas que nos permiten desarrollar programas informáticos, y las **aplicaciones informáticas** son un conjunto de programas que tienen una finalidad más o menos concreta. Son ejemplos de aplicaciones: un procesador de textos, una hoja de cálculo, el software para reproducir música, un videojuego, etc.



Tipos de software.



2 Relación hardware-software.

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

a. **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.
Todas las aplicaciones necesitan recursos hardware durante su ejecución.

b. **Desde el punto de vista de las aplicaciones**

Ya hemos dicho que una aplicación no es otra cosa que un conjunto de programas, y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.
Hay multitud de lenguajes de programación diferentes, 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.

El procesos de desarrollo, que en un principio puede parecer una tarea simple, consta de una serie de pasos de obligado cumplimiento, pues sólo así podremos garantizar que los programas creados son eficientes, fiables y seguros.

3.1 Ciclos de vida del softwware.

Diversos autores han planteado distintos modelos de ciclos de vida, pero los mas conocidos y utilizados son los que aparecen a continuación:

- **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).

- **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.

- **Modelos Evolutivos**

  Son más modernos que los anteriores. 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.

4 Herramientas de apoyo al desarrollo del software.

En la práctica, 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.

Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis del mismo, que son las causas principales de los fallos del software.

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**: 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…

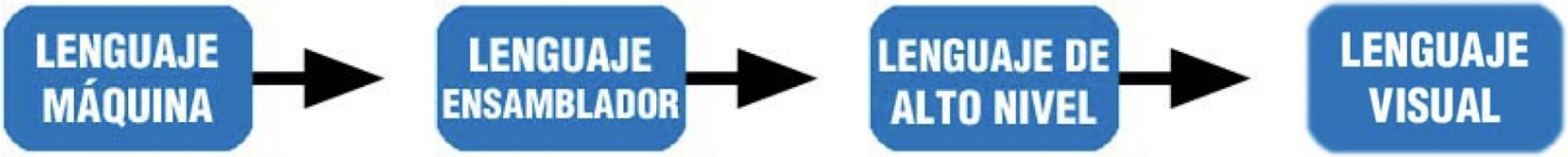
5 Lenguajes de programación.

Podemos definir un Lenguaje de Programación como un idioma creado de forma artificial, formado por un conjunto de símbolos y normas que se aplican sobre un alfabeto para obtener un código, que el hardware de la computadora pueda entender y ejecutar.

Los lenguajes de programación son los que nos permiten comunicarnos con el hardware del ordenador.

Hay multitud de lenguajes de programación, cada uno con unos símbolos y unas estructuras diferentes. Además, cada lenguaje está enfocado a la programación de tareas o áreas determinadas. Por ello, la elección del lenguaje a utilizar en un proyecto es una cuestión de extrema importancia.

Los lenguajes de programación han sufrido su propia evolución, como se puede apreciar en la figura siguiente:



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.

6 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*.

7 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.

En la P.O.O. los programas se componen de objetos independientes entre sí que colaboran 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, 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.*

8 Fases 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 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.

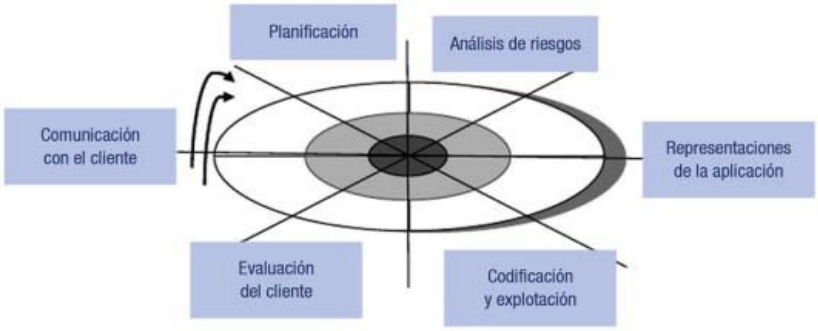
MANTENIMIENTO.

Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

8.1 Análisis.

Esta es la primera fase del proyecto. Una vez finalizada, pasamos a la siguiente (diseño).

Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.



Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista.

¿Qué se hace 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.
- **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).

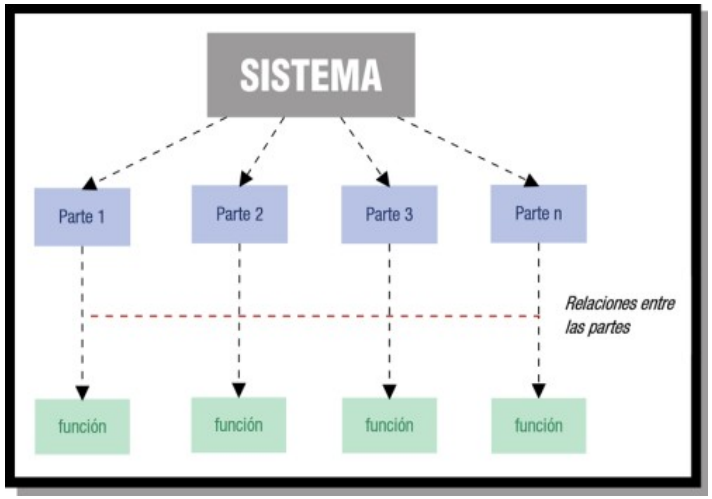
En este documento quedan especificados:

- 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.

Ejemplo de requisitos funcionales, en la aplicación para nuestros clientes de las tiendas de cosmética, habría que considerar:

- Si desean que la lectura de los productos se realice mediante códigos de barras.
- Si van a detallar las facturas de compra y de qué manera la desean.
- Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.
- Si van a operar con tarjetas de crédito.
- Si desean un control del stock en almacén.
- Etc.

8.2 Diseño.



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

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 toman decisiones importantes como:

- Entidades y relaciones de la Base de Datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del Sistema Gestor de Base de Datos.
- Etc.

8.3 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.
- **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.

Máquinas virtuales.

Una maquina 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 compilado como interpretado.

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 posibilidad (Capacidad de un programa para ser ejecutado *en cualquier arquitectura física de un equipo.*) 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, para el control de los dispositivos hardware implicados en los procesos,
- Cumplimiento de las normas de seguridad de las aplicaciones.

Frameworks.

Un framework (Plataforma, entorno, marco de trabajo del desarrollo rápido de aplicaciones) es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir de 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 y partes del 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 de software.
  - **Portabilidad** de aplicaciones de un ordenador a otro.
- Inconvenientes de utilizar un framework:
  - Gran **dependencia** del código respecto al framework utilizado.
  - La instalación e implementación del framework en nuestro equipo **consume bastantes recursos** del sistema.

Entornos de ejecución.

Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. En ocasiones pertenece al propio sistema operativo, pero también se puede instalar como software independiente que funcionará por debajo de la aplicación.
Es decir, es un conjunto de utilidades que permiten la ejecución de programas.

8.4 Pruebas.

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas. 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 validación y verificación del software construido.

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

- **PRUEBAS UNITARIAS:**
  Consiste en probar, una a una, las diferentes partes del software y comprobar su funcionamiento.

- **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.

8.5 Documentación.

Todas las etapas en el desarrollo software deben quedar documentadas. Para dar toda la información a los usuarios de nuestro software y poder acometer posteriores revisiones del proyecto.
Una correcta documentación permitirá la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen de forma modular.

Distinguimos tres grandes documentos en el desarrollo de software.

	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Documentan:	El diseño de la aplicación. La codificación de los programas. Las pruebas realizadas.	Descripción de la funcionalidad de la aplicación. Forma de comenzar a ejecutar la aplicación. Ejemplos de uso del programa. Requerimientos software de la aplicación. Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: • Puesta en marcha. • Explotación. • Seguridad del sistema.
Dirigido a	Al personal técnico en informática (analistas y pro-gramadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios.
Objetivo	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

8.6 Explotación.

La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.

La explotación es 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.

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 f**ase de producción normal**. La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.

8.7 Mantenimiento.

Sería lógico pensar que con la entrega de nuestra aplicación (la instalación y configuración de nuestro proyecto en los equipos del cliente) hemos terminado nuestro trabajo.

En cualquier otro sector laboral esto es así, pero el caso de la construcción de software es muy diferente.

La etapa de mantenimiento es la más larga de todo el ciclo de vida del software.

Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo. Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó.

Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores.

Por todo ello, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

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).