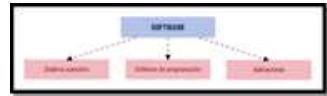


EN
_01

Desarrollo de software.

1.- Software y programa. Tipos de software.

Es de sobra conocido que el ordenador se compone de dos partes bien diferenciadas:
..... y



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

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.

A su vez, un programa es un conjunto de instrucciones escritas en un lenguaje de programación.



En definitiva, distinguimos los siguientes tipos de software:

En este tema, nuestro interés se centra en las aplicaciones informáticas: cómo se desarrollan y cuáles son las fases por las que necesariamente han de pasar.

A lo largo de esta primera unidad vas a aprender los conceptos fundamentales de software y las fases del llamado ciclo de vida de una aplicación informática.

También aprenderás a distinguir los diferentes lenguajes de programación y los procesos que ocurren hasta que el programa funciona y realiza la acción deseada.

2.- Relación hardware-software.

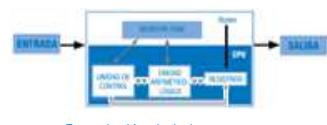
Como sabemos, al conjunto de dispositivos físicos que conforman un ordenador se le denomina hardware.

Existe una relación indisoluble entre éste y el software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione.

El software e se ejecutará sobre los dispositivos físicos.

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:



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 (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de Entrada/Salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).

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 (como ya veremos en su momento). 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).

Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?

Como veremos a lo largo de esta unidad, tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

Autoevaluación

Desarrollo de software.

Para fabricar un programa informático que se ejecuta en una computadora:

- Hay que escribir las instrucciones en código binario para que las entienda el hardware.
- Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
- Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
- Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

3.- Desarrollo de software.

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



El proceso 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, seguros y responden a las necesidades de los usuarios finales (aquellos que van a utilizar el programa).

Como veremos con más detenimiento a lo largo de la unidad, el desarrollo de software es un proceso que conlleva una serie de pasos. Genéricamente, estos pasos son los siguientes:

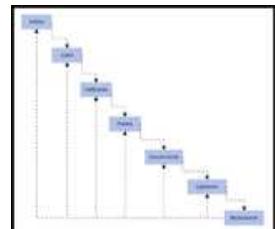
Etapas en el desarrollo de software:

Como vamos a ver en el siguiente punto, según el orden y la forma en que se lleven a cabo las etapas hablaremos de diferentes ciclos de vida del software.

La construcción de software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolle.

3.1.- Ciclos de vida del software.

Ya hemos visto que la serie de pasos a seguir para desarrollar un programa es lo que se conoce como Ciclo de Vida del Software.



Cada etapa vendrá explicada con más detalle en el punto de la presente unidad dedicado a las fases del desarrollo y ejecución del software.

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

Siempre se debe aplicar un modelo de ciclo de vida al desarrollo de cualquier proyecto 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 podemos 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.

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.

3.2.- 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...

4.- Lenguajes de programación.

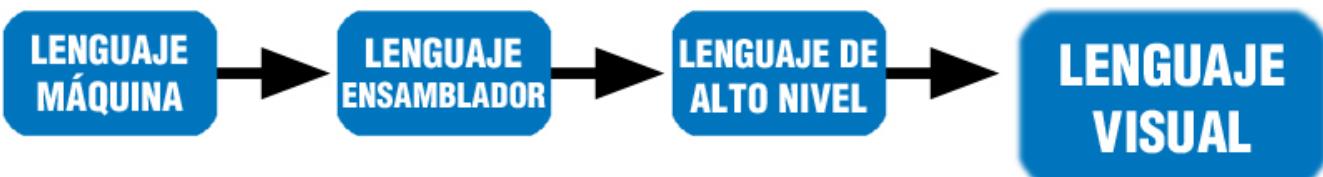
Ya dijimos anteriormente que los programas informáticos están escritos usando algún lenguaje de programación. Por tanto, podemos definir un Lenguaje de Programación como un idioma creado de forma artificial, formado por un conjunto de símbolos y normas que se aplican sobre un alfabeto para obtener un 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.

En otras palabras, es muy importante tener muy clara la función de los lenguajes de programación. Son los instrumentos que tenemos para que el ordenador realice las tareas que necesitamos.

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.

4.1.- Concepto y características.

Ya sabemos que los lenguajes de programación han evolucionado, y siguen haciéndolo, siempre hacia la mayor usabilidad de los mismos (que el mayor número posible de usuarios lo utilicen y exploten).

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

CONCEPTO

Un lenguaje de programación es el conjunto de:

- ✓ **Alfabeto:** conjunto de símbolos permitidos.
- ✓ **Sintaxis:** normas de construcción permitidas de los símbolos del lenguaje.
- ✓ **Semántica:** significado de las construcciones para hacer acciones válidas.

CARACTERÍSTICAS

Podemos clasificar los distintos tipos de Lenguajes de Programación en base a 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.

```
package calculadora;
import junit.framework.TestCase;
/**
 * @author Usuario
 */
public class CalculandoTest extends TestCase {
    public CalculandoTest(String testName) {
        super(testName);
    }
    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }
}
```

A pesar de la inmensa cantidad de lenguajes de programación existentes, Java, C, C++, PHP y Visual Basic concentran alrededor del 60% del interés de la comunidad informática mundial.

4.2.- Lenguajes de programación estructurados.

Aunque los requerimientos actuales de software son bastante más complejos de lo que la técnica de programación estructurada es capaz, es necesario por lo menos conocer las bases de los Lenguajes de Programación estructurados, ya que a partir de ellos se evolucionó hasta otros lenguajes y técnicas más completas (orientada a eventos u objetos) que son las que se usan actualmente.

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).

Los lenguajes de programación que se basan en la programación estructurada reciben el nombre de lenguajes de programación estructurados.

La programación estructurada fue de gran éxito por su sencillez a la hora de construir y leer programas. Fue sustituida por la programación modular, que permitía dividir los programas grandes en trozos más pequeños (siguiendo la conocida técnica "divide y vencerás"). A su vez, luego triunfaron los lenguajes orientados a objetos y de ahí a la programación visual (siempre es más sencillo programar gráficamente que en código, ¿no crees?).

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 de lenguajes estructurados: Pascal, C, Fortran.

La Programación estructurada evolucionó hacia la Programación modular, que divide el programa en trozos de código llamados módulos con una funcionalidad concreta, que podrán ser reutilizables.

4.3.- Lenguajes de programación orientados a objetos.

Después de comprender que la programación estructurada no es útil cuando los programas se hacen muy largos, es necesaria otra técnica de programación que solucione este inconveniente. Nace así la Programación Orientada a Objetos (en adelante, P.O.O.).

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 plantilla para la creación de objetos según un modelo predefinido y se utilizan para representar entidades o conceptos.
- ✓ 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.

5.- 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:

1. ANÁLISIS DE REQUISITOS.

Se especifican los requisitos funcionales y no funcionales del sistema.

2. DISEÑO.

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

3. CODIFICACIÓN.

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

4. PRUEBAS.

Se prueban los programas para detectar errores y se depuran.

5. DOCUMENTACIÓN.

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

6. EXPLOTACIÓN.

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

7. MANTENIMIENTO.

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

Autoevaluación

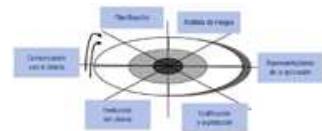
¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?

- Sí.
- No.

5.1.- 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.

Como 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.

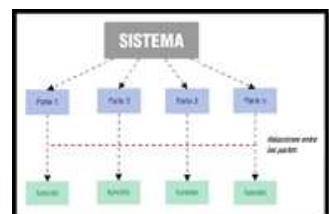
5.2.- Diseño.

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 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.

5.3.- Codificación. Tipos de código.

Durante la fase de codificación se realiza el proceso de programación.

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:

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 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. 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.

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

5.4.- Fases en la obtención de código.

5.4.1.- Fuente.

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

Este conjunto de instrucciones no es directamente ejecutable por la máquina, sino que deberá ser traducido al lenguaje máquina, que la computadora será capaz de entender y ejecutar.

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. El algoritmo lo diseñamos en pseudocódigo y con él, la codificación posterior a algún Lenguaje de Programación determinado será más rápida y directa.

Para obtener el código fuente de una aplicación informática:

1. Se debe partir de las etapas anteriores de análisis y diseño.
2. Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.
3. Se elegirá una Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar.
4. 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.

Puesto que, como hemos dicho antes, este código no es inteligible por la máquina, habrá que TRADUCIRLO, obteniendo así un código equivalente pero ya traducido a código binario que se llama código objeto. Que no será directamente ejecutable por la computadora si éste ha sido compilado.

Un aspecto importante a tener en cuenta es su licencia. Así, en base a ella, 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.

Autoevaluación

Para obtener código fuente a partir de toda la información necesaria del problema:

- Se elige el Lenguaje de Programación más adecuado y se codifica directamente.
- Se codifica y después se elige el Lenguaje de Programación más adecuado.
- Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.

5.4.2.- Objeto.

El código objeto es un código intermedio.

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.

Consiste en un bytecode (código binario) que está distribuido en varios archivos, cada uno de los cuales corresponde a cada programa fuente compilado.

Sólo se genera código objeto una vez que el código fuente está libre de errores sintácticos y semánticos.

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. No existe código objeto intermedio. El software responsable se llama intérprete. El proceso de traducción es más lento que en el caso de la compilación, pero es recomendable cuando el programador es inexperto, ya que da la detección de errores es más detallada.

El código objeto es código binario, pero no puede ser ejecutado por la computadora



5.4.3.- Ejecutable.

El código ejecutable, resultado de enlazar los archivos de código objeto, consta de 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.

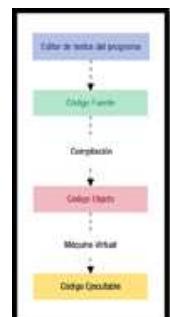
Para obtener un sólo archivo ejecutable, habrá que enlazar todos los archivos de código objeto, a través de un software llamado linker (enlazador) y obtener así un único archivo que ya sí es ejecutable directamente por la computadora.

En el esquema de generación de código ejecutable, vemos el proceso completo para la 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.



Autoevaluación

Relaciona los tipos de código con su característica más relevante, escribiendo el número asociado a la característica en el hueco correspondiente.

Tipo de código.	Relación.	Características.
Código Fuente	<input type="checkbox"/>	1. Escrito en Lenguaje Máquina pero no ejecutable.
Código Objeto	<input type="checkbox"/>	2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable.
Código Ejecutable	<input type="checkbox"/>	3. Escrito en Lenguaje Máquina y directamente ejecutable.

5.5.- Máquinas virtuales.

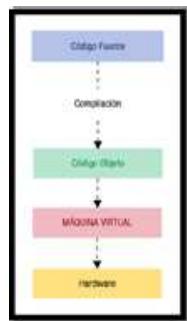
Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

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 podemos 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 LA MÁQUINA VIRTUAL

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.

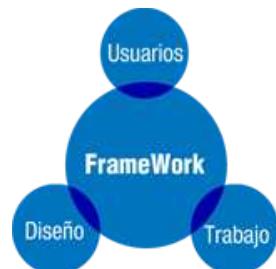
La máquina virtual actúa de puente entre la aplicación y el hardware concreto del equipo donde se instale.

5.5.1.- 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 de Frameworks:

- ✓ .NET es un framework para desarrollar aplicaciones sobre Windows. Ofrece el "Visual Studio .net" que nos da facilidades para construir aplicaciones y su motor es el ".Net framework" que permite ejecutar dichas aplicaciones. Es un componente que se instala sobre el sistema operativo.
- ✓ Spring de Java. Son conjuntos de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones.

5.5.2.- 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.

Se denomina runtime al tiempo que tarda un programa en ejecutarse en la computadora.

Durante la ejecución, los entornos se encargarán de:

- ✓ Configurar la memoria principal disponible en el sistema.
- ✓ Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados. Considerando que las bibliotecas son el conjunto de subprogramas que sirven para desarrollar o comunicar componentes software pero que ya existen previamente y los subprogramas serán aquellos que hemos creado a propósito para el programa.
- ✓ Depurar los programas: comprobar la existencia (o no existencia) de errores semánticos del lenguaje (los sintácticos ya se detectaron en la compilación).

**Funcionamiento del entorno de ejecución:**

El Entorno de Ejecución está formado por la máquina virtual y los 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.

Sin embargo, si lo que queremos es desarrollar nuevas aplicaciones, no es suficiente con el entorno de ejecución.

Adelantándonos a lo que veremos en la próxima unidad, para desarrollar aplicaciones necesitamos algo más. Ese "algo más" se llama entorno de desarrollo.

Autoevaluación

Señala la afirmación falsa respecto de los entornos de ejecución:

- Su principal utilidad es la de permitir el desarrollo rápido de aplicaciones.
- Actúa como mediador entre el sistema operativo y el código fuente.
- Es el conjunto de la máquina virtual y bibliotecas necesarias para la ejecución.

5.5.3.- Java runtime environment.

En esta sección de va a explicar el funcionamiento, instalación, configuración y primeros pasos del Runtime Environment del lenguaje Java (se hace extensible a los demás lenguajes de programación).

Concepto.

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.

Componentes.

JRE está formado por:

- ✓ Una Máquina virtual Java (JVM o JRE 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: JVM y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.

Lo primero es descargarnos el programa JRE. (Java2 Runtime Environment JRE 1.6.0.21). Java es software libre, por lo que podemos descargarnos la aplicación libremente.

Una vez descargado, comienza el proceso de instalación, siguiendo los pasos del asistente.

5.6.- 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

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).

El período de prueba será normalmente el pactado con el cliente.

5.7.- Documentación.

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.

¿Por qué hay que documentar todas las fases del proyecto?

Para dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

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.

Distinguimos tres grandes documentos en el desarrollo de software:

Documentos a elaborar en el proceso de desarrollo de software

	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	<ul style="list-style-type: none"> ✓ El diseño de la aplicación. ✓ La codificación de los programas. ✓ Las pruebas realizadas. 	<ul style="list-style-type: none"> ✓ 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. 	<p>Toda la información necesaria para:</p> <ul style="list-style-type: none"> ✓ Puesta en marcha. ✓ Explotación. ✓ Seguridad del sistema.
¿A quién va dirigido?	Al personal técnico en informática (analistas y programadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su 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.

5.8.- Explotación.

Después de todas las fases anteriores, una vez que las pruebas nos demuestran que el software es fiable, carece de errores y hemos documentado todas las fases, el siguiente paso es la explotación.

Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.

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 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 llevar 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.

Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente: será el momento crítico del proyecto.

5.9.- 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).

Autoevaluación

¿Cuál es, en tu opinión, la etapa más importante del desarrollo de software?

- El análisis de requisitos.
- La codificación.
- Las pruebas y documentación.
- La explotación y el mantenimiento.

Anexo I.- Sentencias de control de la programación estructurada.

SENTENCIAS SECUENCIALES

Las sentencias secuenciales son aquellas que se ejecutan una detrás de la otra, según el orden en que hayan sido escritas.

Ejemplo en lenguaje C:

```
printf ("declaración de variables");
int numero_entero;
espacio=espacio_inicio + velocidad*tiempo;
```

SENTENCIAS SELECTIVAS (CONDICIONALES)

Son aquellas en las que se evalúa una condición. Si el resultado de la condición es verdad se ejecutan una serie de acción o acciones y si es falso se ejecutan otras.

if → señala la condición que se va a evaluar

then → Todas las acciones que se encuentren tras esta palabra reservada se ejecutarán si la condición del **if** es cierta (en C, se omite esta palabra).

else → Todas las acciones que se encuentren tras esta otra palabra reservada se ejecutarán si la condición de **if** es falsa.

Ejemplo en lenguaje C:

```
if (a >= b)
c= a-b;
else
c=a+b;
```

SENTENCIAS REPETITIVAS (ITERACIONES O BUCLES)

Un bucle iterativo de una serie de acciones harán que éstas se repitan mientras o hasta que una determinada condición sea falsa (o verdadera).

while → marca el comienzo del bucle y va seguido de la condición de parada del mismo.

do → a partir de esta palabra reservada, se encontrarán todas las acciones a ejecutar mientras se ejecute el bucle (en C, se omite esta palabra).

done → marca el fin de las acciones que se van a repetir mientras estemos dentro del bucle (en C, se omite esta palabra).

Ejemplo en lenguaje C:

```
int num;
num = 0;
while (num<=10) { printf("Repetición numero %d\n", num);
num = num + 1;
};
```