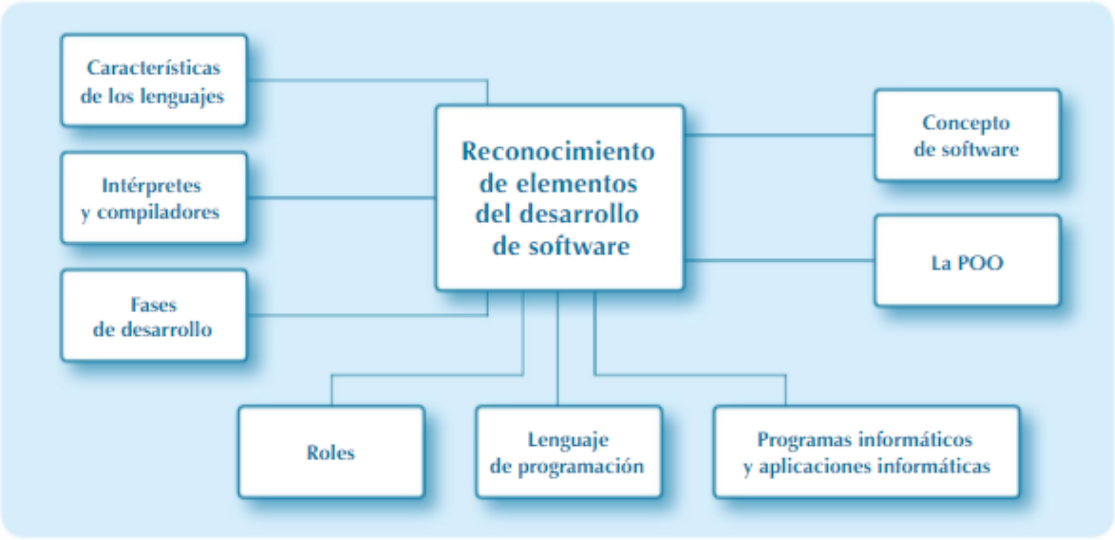
1

Reconocimiento de elementos del desarrollo de software

Objetivos

- ✓ Es preciso que comprendas los principios básicos del desarrollo de software. Se tratarán aspectos como lenguajes de programación y sus características, así como las fases del desarrollo de una aplicación.
- ✓ También es importante que entiendas la diferencia entre compiladores, intérpretes y máquinas virtuales, pues los lenguajes de programación funcionarán y tendrán ventajas y desventajas dependiendo de la tecnología subyacente.

Mapa conceptual



Glosario

Bytecode. Código interpretable directamente para una máquina virtual.

Código máquina. Instrucciones compuestas por unos y ceros interpretables directamente por el hardware.

Criptografía. Arte o ciencia cuyo objetivo es alterar el contenido de un mensaje haciéndolo ininteligible a los receptores que no estén autorizados.

GUI. Acrónimo del inglés *graphical user interface* (interfaz gráfica de usuario).

Hilo de ejecución. Unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. Un proceso o programa puede tener varios hilos de ejecución, lo que implica que están ejecutándose concurrentemente al mismo tiempo distintas instrucciones.

Lenguaje de bajo nivel. Aquel que es parecido al código máquina. Cuanto más bajo nivel tenga el lenguaje, más cercano al código máquina será.

Parámetro. Valor que se pasa a un software para que este, en su ejecución, tome las medidas oportunas en función de lo indicado por dicho valor.

Patrón de diseño. Preestructura de aplicación. Existe una multitud de tipo estándar y su mayor ventaja es la robustez, además de que muchos desarrolladores los conocen, con lo cual comprender, desarrollar y mantener la aplicación será más fácil.

Portabilidad. Facilidad que tiene un software para ejecutarse en diferentes máquinas, diferentes entornos y diferentes sistemas operativos. Cuanto más portable sea un software, en más equipos podrá ejecutarse.

ProFTPd. Servidor FTP *open source* con licencia *GPL*.

1.1. Introducción

El software es la parte intangible de un sistema informático, el equivalente al equipamiento o soporte lógico. Lo constituyen los componentes lógicos (no físicos) y, por tanto, no tangibles. Todo software está diseñado para realizar una tarea determinada en nuestro sistema.

El software es el encargado de comunicarse con el hardware, es decir, se encarga de traducir todas las órdenes que el usuario comunica al software en órdenes comprensibles por el hardware.

Puede definirse software como "el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación" (definición extraída del estándar 729 de IEEE).

PARA SABER MÁS

- ✓ El concepto de software fue utilizado por Charles Babbage (1791-1871) hace mucho tiempo. Cuando trabajaba en su máquina diferencial, utilizaba series de instrucciones que se leían desde la memoria principal del sistema, y a esta serie de instrucciones las denominaba software.

  Si visitas Londres, en el Science Museum, podrás ver algunos mecanismos inconclusos que Charles Babbage desarrolló. Y si quieres conocer más sobre este genio, parte de su cerebro se encuentra conservado en formol en el Royal College of Surgeons de la misma ciudad.
- ✓ Posteriormente, Alan Turing (1912-1954) profundizó en el concepto de software. Este desarrolló su carrera como matemático, pero destacó en su momento como informático y criptógrafo. Sin embargo, sus logros se vieron truncados, ya que, tras ser acusado y procesado por ser homosexual, se suicidó.

  Turing contribuyó a descifrar la máquina Enigma de los alemanes durante la Segunda Guerra Mundial, elaboró la máquina de Turing y desarrolló la teoría de la computación, la cual se tiene como referente hoy en día y forma parte del origen del software moderno.

  Se considera a Turing uno de los padres de la ciencia de la computación, antecedente de la informática moderna.
- ✓ La palabra software, entendida como tal, la empleó por primera vez John Wilder Tukey (1915-2000) en un artículo de la revista *American Mathematical Monthly* en 1958.

  En este artículo, donde se empleó el término *computer software* por primera vez, se hablaba de aprovechar las capacidades de cálculo de los ordenadores de tal manera que los programadores pudiesen escribir conjuntos de instrucciones (programas), los cuales podrían llegar a ser complejos, que luego se traducirían en otras más comprensibles para las máquinas en las que fueran a ser ejecutadas.

  Como puede observarse, se establecían las bases de los modernos compiladores. J. W. Tukey también creó otro término imprescindible en la tecnología computacional, la palabra *bit* como contracción de "dígito binario", por su abreviación del inglés *binary digit*.

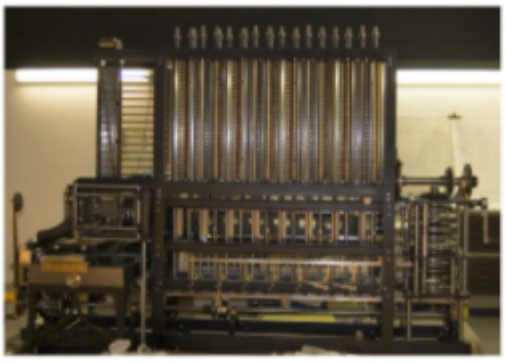


Figura 1.1
Máquina diferencial de Charles Babbage.

Como todo el mundo sabe, el software tiene una serie de características muy particulares que lo definen. Estas características son las siguientes:

a) El *software es lógico,* no físico. Es intangible.
b) El *software se desarrolla,* no se fabrica. Es mejor decir *desarrolladores de software* que *fabricantes de software.*
c) El *software no se estropea* y una copia suya da lugar a un clon con las mismas características del original.
d) En ocasiones, *puede construirse a medida.* Existe software a medida y software enlatado.

Investiga

Busca información sobre:

a) La vida y obra de Alan Turing, puesto que, a la vez que interesante, forma parte del origen del software moderno.
b) El *firmware.*

1.2. Programas informáticos y aplicaciones informáticas

1.2.1. Concepto de programa informático

Un programa es una serie de órdenes o instrucciones secuenciadas u ordenadas con una finalidad concreta y que realizan una función determinada.

Véase el típico programa Hola mundo desarrollado en lenguaje C:

```
//**Código:Programa Holamundo****
/* Programa holamundo.c */
#include <stdio.h>
main(){
   printf("Hola Mundo");
}
```

El texto anterior representa un ejemplo de lo que sería un programa. Se compone de 6 líneas, que van a comentarse a continuación:

```
/* Programa holamundo.c */
Esta línea no realiza ninguna función solo dice cuál es el nombre del pro-
   grama.
#include <stdio.h>
Esta línea es necesaria si va a sacarse algo por pantalla.
main ()
Esta línea indica que esto es lo primero que va ejecutar el programa (lo
   contenido entre { y }).{
   printf ("Hola Mundo");
   Esta línea muestra las palabras Hola mundo por pantalla.
}
```

En resumen, este programa mostrará las palabras *Hola mundo* por pantalla.

1.2.2. Concepto de aplicación informática

Existen en el mercado muchas aplicaciones informáticas y cada una tiene su utilidad y finalidad concreta. Generalmente, las aplicaciones suelen estar formadas por varios programas con sus librerías correspondientes, aunque podrían constar solamente de un programa. Cuando son varios programas que pueden ejecutarse independientemente uno de otro, suele denominarse *suite* o *paquete integrado* como, por ejemplo, la *suite* ofimática de OpenOffice.

Habitualmente, estos programas tienen un nexo de unión en común, comparten librerías, almacén e incluso datos. La compatibilidad entre ellos es completa.

Una aplicación informática está en contacto con el usuario y no con el hardware. Será el sistema operativo el que haga de nexo de unión entre ambos (aplicación informática y hardware).

Existe multitud de aplicaciones informáticas que automatizan o ayudan a la realización de ciertas tareas como, por ejemplo:

- Programas de contabilidad.
- Bases de datos.
- Programas de diseño gráfico.
- Procesadores de texto.
- Programa de facturación.
- Aplicaciones multimedia.
- Hojas de cálculo.
- Presentaciones.
- Herramientas de correo electrónico.

1.2.3. Software a medida y software estándar

Un *software a medida* es una o varias aplicaciones realizadas según los requerimientos e instrucciones de una empresa u organismo. Dichos programas se amoldan o adecuan a la actividad desarrollada y son diseñados a la medida del organismo, su forma de trabajar, sus necesidades.

Algunas características del software a medida son las siguientes:

1. Como todo software, necesita un tiempo de desarrollo.
2. Se adapta a las necesidades específicas de la empresa. Eso implica que, en ocasiones, ese software no sea trasladable a otras empresas diferentes (incluso a otras empresas del mismo sector).
3. Generalmente, suele contener errores y se necesita una etapa de mantenimiento en la que se subsanan y se mejora dicho software.
4. En general, es más costoso que el software estándar, debido a que el precio lo soporta un solo cliente. En el software enlatado o estándar, ese precio se comparte entre los distintos compradores.

El *software estándar o enlatado* es un software genérico (válido para cualquier cliente potencial), que resuelve múltiples necesidades. Normalmente, para hacerlo más adaptable, dicho software tiene herramientas de configuración que lo parametrizan y lo adaptan a las necesidades del cliente. En ocasiones, no es el software ideal, puesto que le faltan opciones, procedimientos y procesos que la empresa realiza y que, a la postre, tendrán que realizarse con otra herramienta.

Sus principales características son:

a) Se compra *ya hecho*. El software ya fue desarrollado en su momento y lo único que podría hacerse es adaptarlo a las necesidades de la empresa.
b) Suele tener muchos *menos errores* que el software a medida, dado que fue probado por múltiples empresas.
c) Suele ser *más barato* que el software a medida, puesto que los costes de desarrollo se reparten entre las múltiples licencias que se venden.
d) Generalmente, tiene funciones que la empresa *no usará* y también *carecerá* de otras opciones. Por regla general, no se adapta completamente a las necesidades de una empresa (es más, a veces, la empresa tiene que adaptarse al software).

1.3. Lenguajes de programación

Todos los programas se desarrollan en algún lenguaje de programación. Nadie (más bien casi nadie) programa directamente con instrucciones máquina, debido a que son ininteligibles para el ser humano, y realizar un programa de este tipo provocaría numerosos errores.

Los lenguajes de programación son, por lo tanto, lenguajes artificiales creados para que, al traducirse a código máquina, cada una de las instrucciones de dicho lenguaje dé lugar a una o a varias instrucciones máquina.

Habitualmente, dichos lenguajes tienen una sintaxis y un conjunto de normas y palabras reservadas, de tal manera que la traducción sea lo más efectiva posible.

Investiga

¿Qué es ReactJS y AngularJS y para qué sirven?

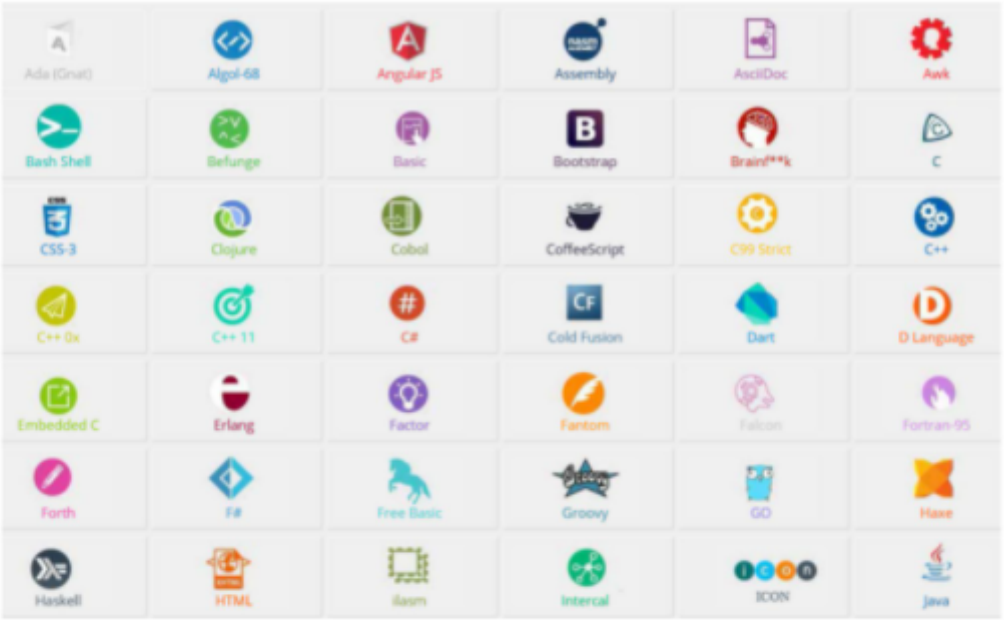


Figura 1.2
Plataforma Coding Ground.

En la actualidad, hay multitud de lenguajes de programación. Uno de los más conocidos es Java, pero no es el único.

1.3.1. Tipos de lenguajes de programación

Los lenguajes de programación han evolucionado a lo largo del tiempo para aumentar su rendimiento y facilitar el trabajo a los programadores. Cada vez, existen lenguajes de programación que son más *user friendly* y, por tanto, más fáciles de programar.

Muchas veces, el que se programe más rápido y los programas sean más sencillos de realizar provoca que estos sean más lentos y que ocupen más espacio.

A continuación, se muestran las características de los lenguajes de programación existentes en la actualidad:

1. *Lenguaje máquina:*

   - Sus instrucciones son complejas e ininteligibles. Se componen de combinaciones de unos y ceros.
   - No necesita ser traducido, por lo tanto es el único lenguaje que entiende directamente el ordenador.
   - Fue el primer lenguaje utilizado. En su momento, los expertos debían tener un dominio profundo del hardware para poder entender este lenguaje de programación.
   - Difiere para cada procesador. Las instrucciones no son portables de un equipo a otro.
   - Salvo excepciones, actualmente, nadie programa en este lenguaje.

2. *Lenguaje de medio nivel o ensamblador:*

   - Dada la dificultad y poca portabilidad del lenguaje máquina, el ensamblador lo sustituyó para facilitar la labor de programación.
   - Sigue estando cercano al hardware, pero, en lugar de unos y ceros, se programa usando mnemotécnicos, que son instrucciones más inteligibles por el programador que permiten comprender de una forma más sencilla qué hace el programa.
   - Este lenguaje necesita compilarse y traducirse al lenguaje máquina para poder ejecutarse.
   - Se trabaja con los registros del procesador y direcciones físicas. En lenguajes de nivel más alto, ya se utilizan variables y estructuras más sofisticadas.
   - Es difícil de comprender y programar.
   - Dada la dificultad y poca portabilidad del lenguaje máquina, el ensamblador lo sustituyó para facilitar la labor de programación.

3. *Lenguajes de alto nivel:*

   - La mayoría de los lenguajes de programación actuales pertenecen a esta categoría.
   - Tienen una forma de programar más intuitiva y sencilla.
   - Más cercano al lenguaje humano que al lenguaje máquina (por ejemplo: WHILE var DO….DONE).
   - Suelen tener librerías y funciones predeterminadas que solucionan algunos de los problemas que suelen presentarse al programador.
   - En ocasiones, ofrecen *frameworks* para una programación más eficiente y rápida.
   - Suelen trabajar con mucha abstracción y orientación a objetos. De esa manera, es más fácil la reutilización y el encapsulamiento de componentes.

TOMA NOTA

Un *framework* es un conjunto de conceptos, estructuras, funciones, componentes, etc. Todo este conjunto de elementos son imprescindibles para personalizarlas y adaptarlas a las necesidades de la aplicación. De esa manera, van construyéndose las aplicaciones.

WWW

Recurso web

Un ejemplo de *framework* es Bootstrap, utilizado en Twitter y liberado para que cualquiera pueda realizar páginas y aplicaciones web con sus componentes.

Si quieres ver más ejemplos de sitios web diseñados con Bootstrap, puedes acceder a su zona de exposición.

Dependiendo de la forma en la que son ejecutados, puede hacerse otra clasificación:

a) *Lenguajes compilados.* Necesitan un programa traductor (compilador) para convertir el código fuente en código máquina. Este tipo de programas se ejecutan de forma más rápida que los interpretados o los virtuales. Además del compilador, existe un programa llamado *enlazador* o *linker* que permite unir el código objeto del programa con el código objeto de las librerías.
b) *Lenguajes interpretados.* No se genera código objeto. El intérprete es un programa que tiene que estar cargado en memoria y se encarga de leer cada una de las instrucciones, interpretarlas y ejecutarlas. Las instrucciones se traducen *on the fly* y solamente aquellas que van ejecutándose, en vez de interpretar todo el programa.
c) *Lenguajes virtuales.* Son lenguajes más portables que los lenguajes compilados, puesto que el código que se genera tras la compilación es un código intermedio o *bytecode*. Este código puede ser, a su vez, interpretado por una máquina virtual instalada en cualquier equipo. Tienen una ejecución lenta, pero su versatilidad para poder ejecutarse en cualquier entorno los hace muy apreciados.

1.3.2. Características de los lenguajes más difundidos

A) Java

Java se consideró en su momento el lenguaje de internet. Surgió como la evolución de C++ pero adaptado a las nuevas necesidades y tendencias de conectividad.

- Java es una simplificación de C++, dado que no permite la sobrecarga de operadores ni herencia múltiple.
- Para cualquier programador, el manejo de *string* (cadena de caracteres) en Java es mucho más eficiente y fácil que en C y C++.
- Es un lenguaje orientado a objetos.
- Está pensado para trabajar con redes y protocolos TCP/IP, HTTP, FTP, etc.
- Es un lenguaje virtual interpretado.
- Muy portable. Ejecutable en cualquier plataforma.
- Ofrece múltiples aspectos de seguridad. Actualmente, se trabaja en la encriptación del código fuente para una mayor seguridad.
- Permite multihilos. Múltiples hilos de ejecución en un mismo programa.

```
public class HolaMundo {
    public static void main(String[] args) {
    System.out.println("Hola Mundo");
    }
}
```

B) Python

Creado por Guido Van Rossum, recibió su nombre por la afición de Guido a los humoristas Monty Python.

- Es un lenguaje de alto nivel. Su ventaja es la portabilidad que ofrece. Si se evita la dependencia de las librerías particulares de cada sistema, un programa en Python puede ejecutarse en cualquier máquina.
- Es un lenguaje interpretado. Al igual que Java, Python convierte el código fuente en *bytecode,* que luego se traduce y se ejecuta en la máquina. No hay que enlazar el código con librerías. Eso se hará a la hora de ejecutar el *bytecode*.
- Está orientado a objetos, como la mayoría de lenguajes de programación modernos.
- Permite escribir código en C y luego combinarlo con programas en Python, de esta manera esa porción de código se ejecutará más rápido.
- Puede incrustarse también en otros lenguajes de programación como C y C++.
- Es uno de los lenguajes más simples y sencillos de aprender.

```
# Hola mundo en Python
print ("¡Hola mundo!")
```

C) C y C++

Son y han sido unos de los lenguajes más potentes y versátiles de la historia de la informática. Mientras que otros lenguajes se han dejado de utilizar, estos siguen utilizándose en múltiples entornos de desarrollo.

- La entrada y salida se ejecuta a través de funciones.
- C estuvo en su momento muy ligado a Unix como sistema operativo, pero C++ intentó desligarse de este.
- Lenguajes estructurados y muy ligados a funciones.
- Incluyen el concepto de *puntero,* que es una variable que contiene la dirección de memoria de otra variable. Este aspecto ofrece mucha flexibilidad, pero, por otra parte, su utilización por parte del programador puede ser compleja.
- Combinan comandos de alto nivel, aunque pueden incluirse fragmentos de código que trabajen a más bajo nivel.
- Los programas son muy eficientes y rápidos.
- El tamaño de los programas compilados es pequeño.

- Son lenguajes relativamente portables. Pueden recompilarse en cualquier tipo de máquina realizando ninguno o pocos cambios.

```
using namespace std;
int main(int argc, char *argv[]) {
    std::cout «"Hola mundo" «endl;
    return 0;
}
```

D) JavaScript

Este lenguaje se utiliza mucho en programación web y sobre todo los *frameworks* basados en él como Angular, React, etc.

- Tiene la ventaja de parecerse mucho a Java, C y C++. Por lo tanto, los programadores de estos lenguajes se sienten cómodos al programar con JavaScript.
- Es un lenguaje de *scripting*.
- Se ejecuta en el lado del cliente.
- Es un lenguaje seguro y fiable.
- Su código es visible y cualquiera puede leerlo, ya que, como se ha explicado, se interpreta en el lado cliente.
- Tiene sus limitaciones como cualquier lenguaje ejecutado en el lado del cliente. Esas limitaciones tienen más que ver con el tema de la seguridad.
- Actualmente, existen muchas librerías basadas en JavaScript como AngularJS, ReactJS, MeteorJS, JQuery, Foundation JS y BackBone.js, entre otras.

```
<script type="text/javascript">
    alert("Hola Mundo!");
</script>
```

E) PHP

Lenguaje web de propósito general utilizado frecuentemente en el *backend* de muchos productos como PrestaShop, WordPress, etc.

- Lenguaje multiplataforma. Puede instalarse prácticamente en cualquier sistema.
- Se orientó desde el principio al desarrollo de webs dinámicas.
- Conocidos son su buena integración con Mysql y otros servicios como ProFTPd, etc.
- Permite aplicar técnicas de orientación a objetos.
- Lenguaje interpretado, con lo cual no hace falta definir variables.
- Existen muchos *frameworks* basados en PHP que permiten trabajar los patrones de diseño modelo-vista-controlador (MVC).

```
<?php echo '<p>Hola Mundo</p>'; ?>
```

F) VB.NET

Microsoft tenía que poner al día su famoso Visual Basic y lo consiguió con su nuevo lenguaje VB.NET

- Desde hace mucho tiempo Visual Basic es uno de los referentes de los lenguajes de programación.
- Visual Basic es el lenguaje que se utiliza para programar macros en Microsoft Office.
- VB.NET es un lenguaje orientado a objetos implementado sobre el *framework*.NET.
- La mayoría de programadores utilizan la herramienta de Microsoft Visual Studio.
- Existen también entornos libres de desarrollo en .NET como Sharpdevelop, pero no es tan potente como el primero.
- Es posible desde el Visual Studio 2008 programar en Ajax.

```
Module VBModule
Sub Main()
Console.WriteLine("Hello, world!")
End Sub
End Module
```

SABÍAS QUE...

Microsoft fue una de las primeras compañías en desarrollar software para Apple. Su fundador, Steve Jobs, tenía muy claro que necesitaba software para que Macintosh fuese un éxito, y Microsoft era la solución.

En sus orígenes, la empresa de Bill Gates se especializaba en los lenguajes de programación y han creado Basic y Fortran. Más tarde, comprarían y revenderían a IBM el DOS entrando de lleno en el mercado de los sistemas operativos.

Para poder escribir un programa de ordenador, es necesario conocerlo todo sobre él, por lo que Jobs tuvo que mostrar a Microsoft los primeros prototipos de Macintosh. A Bill Gates le causó muy buena impresión el sistema operativo de Apple, y comenzó a buscar la forma de emplear los iconos, ventanas y *mouse* en la plataforma IBM PC.

Según cuentan algunos, Gates presionó a Jobs para que le permitiera utilizar partes de la interfaz de Macintosh en PC a cambio de no demorar el lanzamiento de las aplicaciones que Apple necesitaba. Esto sería el detonante para lo que, luego, se llamaría *Windows 1.0*. Todas las futuras demandas legales de Apple contra Microsoft por el uso de algunos elementos de la GUI se vieron debilitadas debido a este antiguo acuerdo entre ambas empresas.

1.4. El proceso de traducción/compilación

Los *traductores* son programas cuya finalidad es traducir lenguajes de alto nivel (en los que se programa) a lenguajes de bajo nivel como ensamblador o código máquina. Existen dos grandes grupos de tipos de traductores: los compiladores y los intérpretes.

Un *intérprete* traduce el código fuente línea a línea como se describe a continuación: primero, traduce la primera línea, detiene la traducción y, posteriormente, la ejecuta; lee la siguiente línea, detiene la traducción y la ejecuta, y así sucesivamente. El intérprete tiene que estar en memoria ejecutándose para poder ejecutar el programa. Al igual que el intérprete, el código fuente tiene que estar también en memoria.

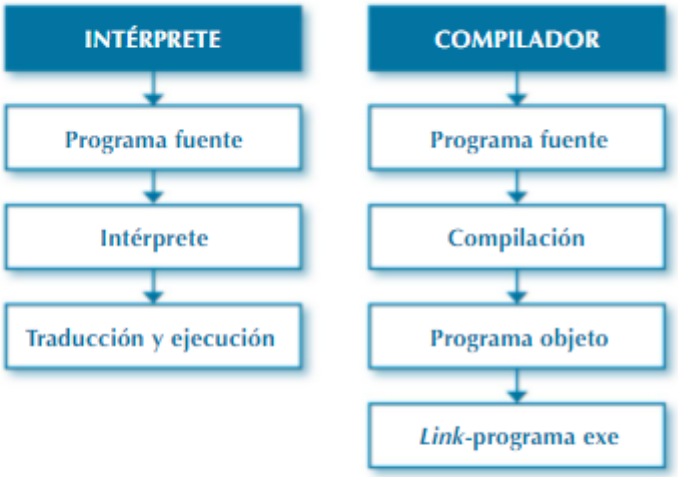


Figura 1.3
Fases de un intérprete y un compilador.

Un *compilador* traduce el código fuente a código máquina. El compilador solamente está en la máquina de desarrollo. El código generado solo funcionará en una máquina con un hardware y un software determinados. Si cambian de hardware o software, hay que volver a recompilarlo.

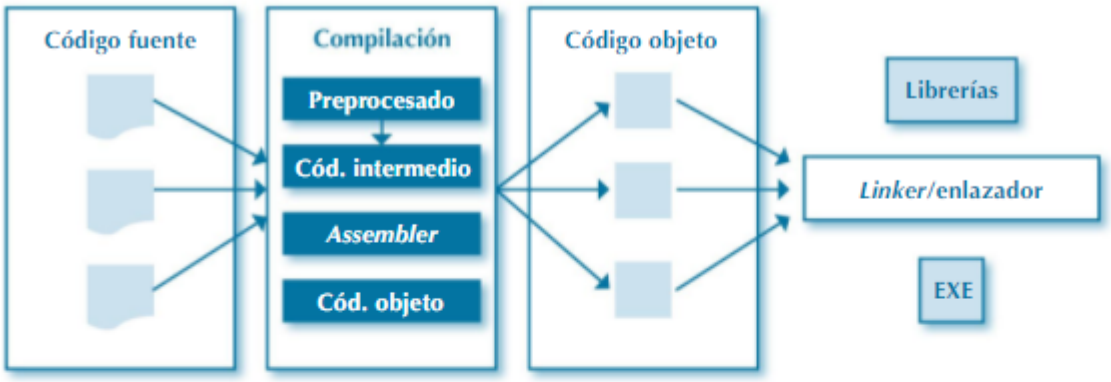


Figura 1.4
Fases de un compilador.

En la figura 1.4, puede observarse cómo funcionan los compiladores.

En el proceso de compilación, primero, se realiza un *preprocesado* del código. Todos los comandos de preprocesamiento se ejecutan y traducen.

Una vez realizado este paso, se compilan los ficheros del código fuente generándose un *código intermedio*. Existen muchos compiladores de código intermedio a código máquina y el objetivo de este paso es reutilizar dichos compiladores, puesto que son rápidos, eficientes, están probados y reducen el tiempo de desarrollo.

El código intermedio, generalmente, vuelve a compilarse y traducirse a *ensamblador* o *código objeto* directamente.

Dependiendo de lo eficiente que sea el compilador, estas fases llevarán más o menos tiempo.

El código objeto es básicamente un código máquina, lo único que le falta es enlazarlo a las librerías para generar un programa ejecutable.

> **RECUERDA**
>
> Durante la fase de compilación, se realizan dos subfases. La primera consiste en realizar un análisis léxico de la aplicación. Se lee secuencialmente el programa y se detectan los *tokens* o lexemas, que son las palabras reservadas del lenguaje, las operaciones, los caracteres de puntuación, etc.
>
> En una segunda fase, se realiza un análisis sintáctico (e incluso semántico). En este análisis, se comprueba que, gramaticalmente, el programa u aplicación es correcto: si el tamaño de las variables es el adecuado, si las conversiones son posibles y, en general, que se cumplen las reglas sintácticas y semánticas del lenguaje.

El proceso de enlazado (incluir en el código ejecutable el código máquina de las librerías) lo realiza un programa específico llamado *enlazador* o *linker*, el cual, además de enlazar todo el código objeto, optimiza los programas para que se ejecuten más rápido en un futuro.

FUNDAMENTAL

Es importante tener clara la diferencia entre:

- ✓ *Código fuente*. Código escrito en un lenguaje de programación.
- ✓ *Código objeto*. Resultado de compilar el código fuente. Puede ser código máquina o *bytecode* si es un lenguaje interpretado luego por una VM *(virtual machine)*.
- ✓ *Código ejecutable*. Resultado de compilar y enlazar el código con las librerías. Ejecutable directamente sobre una máquina concreta.

1.5. Desarrollo de una aplicación

1.5.1. Fases del desarrollo de una aplicación

Existen muchos paradigmas de desarrollo, pero todos ellos suelen tener una serie de etapas en común. A continuación, se detallarán las etapas que se dan en un desarrollo software (figura 1.5), con independencia del tipo de ciclo de vida.

Figura 1.5
Fases del desarrollo de una aplicación.

A) Fase inicial

En esta, se planifica el proyecto, se hacen estimaciones, se decide si el proyecto es rentable o no, etc. Es decir, se establecen las bases de cómo va a desarrollarse el resto de fases del proyecto. En un símil con la construcción de un edificio, consistiría en ver si se dispone de licencia de construcción, cuánto va a costar el edificio, cuántos trabajadores van a necesitarse para construirlo, quién va a construirlo, etc. La fase de planificación y estimación son las más complejas de un proyecto. Para esto, se necesita gente con experiencia tanto en la elaboración de proyectos como en las platafor-

mas en las que va a realizarse su construcción. De esta fase, surgen muchos documentos tanto de planificación (general y detallada) como de estimaciones, en los que se incluyen datos económicos, posibles soluciones al problema y sus costes, etc. Son documentos que se realizan a alto nivel y se acuerdan con la dirección de la empresa o las personas responsables del proyecto. En estas fases iniciales, suelen tomarse decisiones que, a veces, afectan a todas las demás fases del proyecto, con lo cual tiene que estar todo bien documentado, detallado y soportado por datos concretos.

B) Análisis

En esta fase, se analiza el problema. Consiste en recopilar, examinar y formular los requisitos del cliente y analizar cualquier restricción que pueda aplicarse. Todas las entrevistas con el cliente, por lo tanto, tienen que estar registradas en documentos, y, generalmente, esos documentos se consensúan con el cliente y algunos tienen hasta carácter contractual. Por ejemplo, en algunos proyectos, se hace firmar al cliente un documento de requisitos de la aplicación en el cual el equipo de desarrollo se compromete a realizar las especificaciones indicadas por el cliente y este, a su vez, se compromete a no variar sus necesidades hasta por lo menos terminar una primera *release*. Como puede observarse, es un documento que obliga a ambas partes a cumplir con lo acordado.

C) Diseño

Esta fase consiste en determinar los requisitos generales de la arquitectura de la aplicación y en dar una definición precisa de cada subconjunto de la aplicación. En esta fase, los documentos ya son más técnicos.

Suelen crearse dos documentos de diseño: uno más genérico, en el que se tiene una visión de la aplicación más general, y otro detallado, en el que se profundizará en los detalles técnicos de cada módulo concreto del sistema. Estos documentos los realizarán los analistas, junto con la supervisión del jefe de proyecto.

D) Codificación o implementación

Esta fase consiste en la implementación del software en un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.

Durante esta fase, se crea documentación muy detallada en la que se incluye código. Aunque mucho código suele comentarse en el mismo programa, también tienen que generarse documentos donde se indica, por ejemplo, para cada función, las entradas, salidas, parámetros, propósito, módulos o librerías donde se encuentra, quién la ha realizado, cuándo, las revisiones que se han realizado de esta, etc.

Como puede verse, el detalle es máximo teniendo en cuenta que ese código, en un futuro, va a tener que ser mantenido por la misma o, seguramente, otra persona y toda información que pueda recibir, a veces, es poca.

E) Pruebas

En esta fase, se realizarán pruebas para garantizar que la aplicación se ha programado de acuerdo con las especificaciones originales y los distintos programas de los que consta la aplica-

ción están perfectamente integrados y preparados para la explotación. Como se ha visto anteriormente, las pruebas son de todo tipo.

Podría clasificarse la documentación de las pruebas en dos bloques diferentes:

1. *Funcionales.* Existen unas pruebas funcionales en las que se prueba que la aplicación hace lo que tiene que hacer con las funciones acordes a los documentos de especificaciones que se establecieron con el cliente, por lo que esas pruebas deberían realizarse con el cliente presente.

   Mientras están realizándose las pruebas, tiene que hacerse todo tipo de anotaciones para luego plasmarlas en un documento, al que tendrá que darle el visto bueno el cliente (que, por ese motivo, estuvo en las pruebas).

   En ese documento, van detallándose fallos, tanto de la propia aplicación como modificaciones que se hayan realizado en ella, si no cumple con las especificaciones iniciales. En el caso de que haya discrepancias, suele consultarse documentación anterior para que no se incluyan en este punto funcionalidades nuevas o variaciones de las funcionalidades.
2. *Estructurales.* En otro documento aparte, se detallarán los resultados de las pruebas técnicas realizadas. A estas pruebas, ya no hace falta que asista el usuario o cliente, puesto que son de carácter meramente técnico. En estas pruebas, se harán cargas reales, se someterá la aplicación y el sistema a estrés, etc.

F) Explotación

En esta fase, se instala el software en el entorno real de uso y se trabaja con él de forma cotidiana. Generalmente, es la fase más larga y suelen surgir multitud de incidencias, nuevas necesidades, etc.

Toda esta información suele detallarse en un documento en el que se incluyen los errores o fallos detectados intentando ser lo más explícito posible, puesto que luego los programadores y analistas deben revisar estos fallos o *bugs* y darle la mejor solución posible. También surgirán otras necesidades que van a ir detallándose en un documento y que pasarán a realizarse en operaciones de mantenimiento.

G) Mantenimiento

En esta fase, se realiza todo tipo de procedimientos correctivos (corrección de fallos) y actualizaciones secundarias del software (mantenimiento continuo), que consistirán en adaptar y evolucionar las aplicaciones.

Para realizar las labores de mantenimiento, hay que tener siempre delante la documentación técnica de la aplicación. Sin una buena documentación de la aplicación, las labores de mantenimiento son muy difíciles y su garantía, en ese caso, sería poca.

Todas las operaciones de mantenimiento tienen que estar documentadas porque ha de saberse quién ha realizado la operación, qué ha hecho y cómo. También tienen que estar documentadas porque debería probarlas otra persona distinta al programador.

Como se ha dicho, el comienzo y el final de cada fase no están claros. Todas las fases se solapan y, una vez que concluyen, el resultado es la aplicación.

mas en las que va a realizarse su construcción. De esta fase, surgen muchos documentos tanto de planificación (general y detallada) como de estimaciones, en los que se incluyen datos económicos, posibles soluciones al problema y sus costes, etc. Son documentos que se realizan a alto nivel y se acuerdan con la dirección de la empresa o las personas responsables del proyecto. En estas fases iniciales, suelen tomarse decisiones que, a veces, afectan a todas las demás fases del proyecto, con lo cual tiene que estar todo bien documentado, detallado y soportado por datos concretos.

B) Análisis

En esta fase, se analiza el problema. Consiste en recopilar, examinar y formular los requisitos del cliente y analizar cualquier restricción que pueda aplicarse. Todas las entrevistas con el cliente, por lo tanto, tienen que estar registradas en documentos, y, generalmente, esos documentos se consensúan con el cliente y algunos tienen hasta carácter contractual. Por ejemplo, en algunos proyectos, se hace firmar al cliente un documento de requisitos de la aplicación en el cual el equipo de desarrollo se compromete a realizar las especificaciones indicadas por el cliente y este, a su vez, se compromete a no variar sus necesidades hasta por lo menos terminar una primera *release*. Como puede observarse, es un documento que obliga a ambas partes a cumplir con lo acordado.

C) Diseño

Esta fase consiste en determinar los requisitos generales de la arquitectura de la aplicación y en dar una definición precisa de cada subconjunto de la aplicación. En esta fase, los documentos ya son más técnicos.

Suelen crearse dos documentos de diseño: uno más genérico, en el que se tiene una visión de la aplicación más general, y otro detallado, en el que se profundizará en los detalles técnicos de cada módulo concreto del sistema. Estos documentos los realizarán los analistas, junto con la supervisión del jefe de proyecto.

D) Codificación o implementación

Esta fase consiste en la implementación del software en un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.

Durante esta fase, se crea documentación muy detallada en la que se incluye código. Aunque mucho código suele comentarse en el mismo programa, también tienen que generarse documentos donde se indica, por ejemplo, para cada función, las entradas, salidas, parámetros, propósito, módulos o librerías donde se encuentra, quién la ha realizado, cuándo, las revisiones que se han realizado de esta, etc.

Como puede verse, el detalle es máximo teniendo en cuenta que ese código, en un futuro, va a tener que ser mantenido por la misma o, seguramente, otra persona y toda información que pueda recibir, a veces, es poca.

E) Pruebas

En esta fase, se realizarán pruebas para garantizar que la aplicación se ha programado de acuerdo con las especificaciones originales y los distintos programas de los que consta la aplica-

ción están perfectamente integrados y preparados para la explotación. Como se ha visto anteriormente, las pruebas son de todo tipo.

Podría clasificarse la documentación de las pruebas en dos bloques diferentes:

1. *Funcionales.* Existen unas pruebas funcionales en las que se prueba que la aplicación hace lo que tiene que hacer con las funciones acordes a los documentos de especificaciones que se establecieron con el cliente, por lo que esas pruebas deberían realizarse con el cliente presente.

   Mientras están realizándose las pruebas, tiene que hacerse todo tipo de anotaciones para luego plasmarlas en un documento, al que tendrá que darle el visto bueno el cliente (que, por ese motivo, estuvo en las pruebas).

   En ese documento, van detallándose fallos, tanto de la propia aplicación como modificaciones que se hayan realizado en ella, si no cumple con las especificaciones iniciales. En el caso de que haya discrepancias, suele consultarse documentación anterior para que no se incluyan en este punto funcionalidades nuevas o variaciones de las funcionalidades.
2. *Estructurales.* En otro documento aparte, se detallarán los resultados de las pruebas técnicas realizadas. A estas pruebas, ya no hace falta que asista el usuario o cliente, puesto que son de carácter meramente técnico. En estas pruebas, se harán cargas reales, se someterá la aplicación y el sistema a estrés, etc.

F) Explotación

En esta fase, se instala el software en el entorno real de uso y se trabaja con él de forma cotidiana. Generalmente, es la fase más larga y suelen surgir multitud de incidencias, nuevas necesidades, etc.

Toda esta información suele detallarse en un documento en el que se incluyen los errores o fallos detectados intentando ser lo más explícito posible, puesto que luego los programadores y analistas deben revisar estos fallos o *bugs* y darle la mejor solución posible. También surgirán otras necesidades que van a ir detallándose en un documento y que pasarán a realizarse en operaciones de mantenimiento.

G) Mantenimiento

En esta fase, se realiza todo tipo de procedimientos correctivos (corrección de fallos) y actualizaciones secundarias del software (mantenimiento continuo), que consistirán en adaptar y evolucionar las aplicaciones.

Para realizar las labores de mantenimiento, hay que tener siempre delante la documentación técnica de la aplicación. Sin una buena documentación de la aplicación, las labores de mantenimiento son muy difíciles y su garantía, en ese caso, sería poca.

Todas las operaciones de mantenimiento tienen que estar documentadas porque ha de saberse quién ha realizado la operación, qué ha hecho y cómo. También tienen que estar documentadas porque debería probarlas otra persona distinta al programador.

Como se ha dicho, el comienzo y el final de cada fase no están claros. Todas las fases se solapan y, una vez que concluyen, el resultado es la aplicación.

El paradigma de la programación orientada a objetos

Figura 1.6
Paradigma de la POO.

Desde hace mucho tiempo, la programación estructurada era el único paradigma efectivo y eficiente en programación, pero las cosas cambiaron… nació la programación orientada a objetos que rompía con lo establecido. No era una evolución, era una nueva filosofía que no tenía nada que ver con lo anterior.

Los programas, subprogramas, rutinas, funciones, etc., ahora en programación orientada a objetos no tenían tanta importancia como los objetos, métodos, propiedades, herencia, polimorfismo, etc.

Muchos programadores veteranos alucinaban con los principios de la orientación a objetos. Todos los programas se resumen a objetos que tienen una serie de atributos y, asociados, un comportamiento o procedimientos llamados *métodos*. Estos objetos, que son instancias de clases, interaccionarán unos con otros y, de esa manera, se diseñarán aplicaciones y programas.

Hoy en día, es prácticamente imposible programar sin utilizar programación orientada a objetos. Esta forma de programar es más cercana a cómo se expresarían las cosas en la vida real que en otros tipos de programación clásicos.

Analistas y programadores piensan y describen las cosas y el entorno de una manera distinta para plasmar dichos conceptos en programas en términos de objetos, atributos y métodos.

Como puede observarse en la figura 1.6, los pasos del paradigma de la POO se entremezclan con el anterior y el siguiente. Primero, se analiza el mundo real. Este primer trabajo dará paso al siguiente: el diseño. Después, se diseñará en detalle para poder programar los objetos.

1.5.2. La documentación

En cada una de las fases anteriores, se generan uno o más documentos. En ningún proyecto, es viable comenzar la codificación sin haber realizado las fases anteriores porque eso equivaldría a un desastre absoluto.

Además, la documentación debe ser útil y estar adaptada a los potenciales usuarios de dicha documentación (cuando se crea un coche, existen los manuales de usuario y los manuales técnicos para los mecánicos. "Para qué quiero yo saber dónde están situados los inyectores, las bujías o la trócola si nunca la voy a cambiar. A mí lo que me interesa es saber cómo se regula el volante, cómo funciona la radio, etc.").

En cualquier aplicación, como mínimo, deberán generarse los siguientes documentos:

a) *Manual de usuario.* Es, como ya se ha comentado anteriormente, el manual que utilizará el usuario para desenvolverse con el programa. Deberá ser autoexplicativo y de ayuda

para el usuario. Este manual debe servirle para aprender cómo se maneja la aplicación y qué es lo que hay que hacer y lo que no. Si, como técnico, no va a hacerse un manual que sirva al usuario en su comienzo o práctica diaria, es mejor no hacerlo o realizar otro tipo de documentación.

b) *Manual técnico.* Es el manual dirigido a los técnicos (el manual para los mecánicos citado anteriormente). Con esta documentación, cualquier técnico que conozca el lenguaje con el que la aplicación ha sido creada debería poder conocerla casi tan bien como el personal que la creó.

c) *Manual de instalación.* En este manual, se explican paso a paso los requisitos y cómo se instala y pone en funcionamiento la aplicación.

Desde la experiencia, es preciso recalcar una vez más la importancia de la documentación, puesto que, sin documentación, una aplicación o programa es como un coche sin piezas de repuesto, cuando tenga un problema o haya que repararlo, no podrá hacerse nada.

1.5.3. Roles o figuras que forman parte del proceso de desarrollo de software

El equipo de desarrollo de software se compone de una serie de personas, o más bien roles, los cuales tienen unas atribuciones y responsabilidades diferentes. A continuación, se describirán los principales roles en un proyecto de desarrollo de software:

A) Arquitecto de software

Es la persona encargada de decidir cómo va a realizarse el proyecto y cómo va a cohesionarse. Tiene un conocimiento profundo de las tecnologías, los *frameworks,* las librerías, etc. Decide la forma y los recursos con los que va a llevarse a cabo un proyecto.

B) Jefe de proyecto

Dirige el proyecto. Muchas veces, un jefe de proyecto puede ser un analista con experiencia, un arquitecto o simplemente una persona dedicada solamente a ese puesto. Tiene que saber gestionar un equipo y los tiempos, tener una relación fluida con el cliente, etc.

C) Analista de sistemas

Es un rol tradicional en el desarrollo de software. Es una persona con experiencia que realiza un estudio exhaustivo del problema que ha de analizarse y ejecuta tanto el análisis como el diseño de todo el sistema.

La experiencia que tiene este tipo de personas es fundamental, puesto que, muchas veces, es necesaria a la hora de tener reuniones con el cliente, establecer los requisitos de la aplicación, etc.

D) Analista programador

Puesto a caballo entre el analista y el programador. Es un programador sénior, por así decirlo. Realiza funciones de análisis porque sus conocimientos lo permiten y también codifica. En proyectos pequeños, puede realizar ambas funciones (analista y programador).

E) Programador

Su función es conocer en profundidad el lenguaje de programación y codificar las tareas que le han sido encomendadas por el analista o analista-programador.

Resumen

- El software es la parte intangible de un sistema informático, el equivalente al equipamiento o soporte lógico.
- Características del software:
  - El software es lógico, no físico. Es intangible.
  - El software se desarrolla, no se fabrica.
  - El software no se estropea y una copia suya da lugar a un clon.
  - En ocasiones, puede construirse a medida.
- Un programa es una serie de órdenes o instrucciones secuenciadas u ordenadas con una finalidad concreta que realizan una función determinada.
- Un software a medida es una o varias aplicaciones realizadas según los requerimientos e instrucciones de una empresa u organismo.
- El software estándar o enlatado es aquel de tipo genérico que resuelve múltiples necesidades.
- Los lenguajes de programación son, por lo tanto, un lenguaje artificial creado para que, al traducirse a código máquina, cada una de las instrucciones de dicho lenguaje den lugar a una o varias instrucciones máquina.
- El lenguaje máquina es ininteligible y se compone de combinaciones de unos y ceros. No necesita ser traducido, por lo tanto, es el único lenguaje que entiende directamente el ordenador. Fue el primer lenguaje utilizado y difiere para cada procesador.
- El lenguaje de medio nivel o ensamblador sustituyó al lenguaje máquina para facilitar la labor de programación. Sigue estando cercano al hardware, pero, en lugar de unos y ceros, se programa usando mnemotécnicos. Trabaja con los registros del procesador, direcciones físicas y es difícil de comprender y programar.
- Los lenguajes de alto nivel tienen una forma de programar más intuitiva y sencilla. Son más cercanos al lenguaje humano que al lenguaje máquina.
- Los lenguajes pueden clasificarse en:
  - Lenguajes compilados.
  - Lenguajes interpretados.
  - Lenguajes virtuales.
- Los traductores son programas cuya finalidad es traducir lenguajes de alto nivel a lenguajes de bajo nivel como ensamblador o código máquina.
- Existen dos grandes grupos de tipos de traductores:
  - *Compiladores:* traducen el código fuente a código máquina.
  - *Intérpretes:* traducen el código fuente línea a línea.
- Los estados de un programa son los siguientes:
  - *Código fuente.* Código escrito en un lenguaje de programación.
  - *Código objeto.* Resultado de compilar el código fuente.
  - *Código ejecutable.* Resultado de compilar y enlazar el código con las librerías.
- Las fases clásicas del desarrollo de una aplicación son las siguientes:

- Fase inicial.
- Análisis.
- Diseño.
- Codificación.
- Pruebas.
- Explotación.
- Mantenimiento.

- En cualquier aplicación, como mínimo, deberán generarse los siguientes documentos:
  - Manual de usuario.
  - Manual técnico.
  - Manual de instalación.
- Los roles en un proyecto software son:
  - Arquitecto de software.
  - Jefe de proyecto.
  - Analista de sistemas.
  - Analista programador.
  - Programador.