

UNIDAD DIDÁCTICA 1

ELEMENTOS DEL DESARROLLO DEL SOFTWARE

**MÓDULO PROFESIONAL:
ENTORNOS DE DESARROLLO**



CESUR
Tu Centro Oficial de FP

Índice

RESUMEN INTRODUCTORIO	2
INTRODUCCIÓN	2
CASO INTRODUCTORIO.....	2
1.INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN	4
2. TIPOS DE LENGUAJES DE PROGRAMACIÓN.....	8
2.1 Según niveles.....	8
2.2 Según paradigmas de programación	12
2.3 Según la forma en la que son ejecutados	13
2.4 Según los lenguajes de programación más difundidos.....	14
3. COMPILADORES Y DEPURADORES	18
3.1 Análisis	19
3.2 Síntesis	21
3.3 Depuradores.....	23
4. FASES DEL DESARROLLO DE UNA APLICACIÓN.....	24
5. DOCUMENTACIÓN DURANTE LAS FASES DEL DESARROLLO.....	32
6. METODOLOGÍAS ÁGILES	41
RESUMEN FINAL	49

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos relacionados con los elementos básicos que todo programador debe conocer para el desarrollo del software. En este sentido, se pretenden analizar los diferentes tipos de lenguajes de programación que están o han estado “de moda”, la función que desempeñan tanto los compiladores como los depuradores, las fases de desarrollo de una aplicación (análisis, diseño, codificación, pruebas, explotación y mantenimiento) así como la importancia de la documentación durante estas fases del desarrollo, y las metodologías ágiles que se aplican en dicho desarrollo, siendo la más extendida la metodología SCRUM.

INTRODUCCIÓN

Los lenguajes de programación son parte esencial del software, siendo el conjunto de sus líneas de código la esencia de su función. La implementación de aplicaciones informáticas forma parte del día a día de multitud de empresas de programación, conocidas como factorías de software, donde los empleados trabajan para dar soluciones a diferentes clientes como el sector energético, el transporte, servicios financieros, las telecomunicaciones o administraciones públicas.

La realidad que involucra a este desarrollo no es trivial. Se deben aplicar ciertas metodologías que permitan seguir este proceso de desarrollo basado, normalmente en etapas. Una de las metodologías más ampliamente aceptadas es la formada por las siguientes 5 fases: análisis, diseño, implementación, pruebas y mantenimiento. Todos los profesionales involucrados deben conocer en profundidad cuál es la función que desarrolla en cada una de ellas.

De forma transversal a todo el desarrollo, la documentación será un pilar fundamental que debe tenerse siempre presente. Es imprescindible documentar cada proceso, fase o etapa en detalle para disponer de una buena gestión del conocimiento de todo el desarrollo de software realizado y acudir a él siempre que sea necesario.

CASO INTRODUCTORIO

Trabajas en una de las grandes factorías de software del país, eres jefe de proyecto y diriges tu propio equipo de desarrollo. Tu equipo además se encuentra ubicado en distintas zonas geográficas del país, ya que el teletrabajo se ha implantado en el sector.

Se pone en contacto un cliente con la empresa que desea crear un software que permite gestionar una biblioteca, en consecuencia, deberás reunir a tu equipo y aplicar metodologías de desarrollo adecuadas para desarrollar el producto que se precisa dentro de unos estándares de calidad atendiendo a las diferentes tareas que se deban asignar, dentro del plazo previsto.

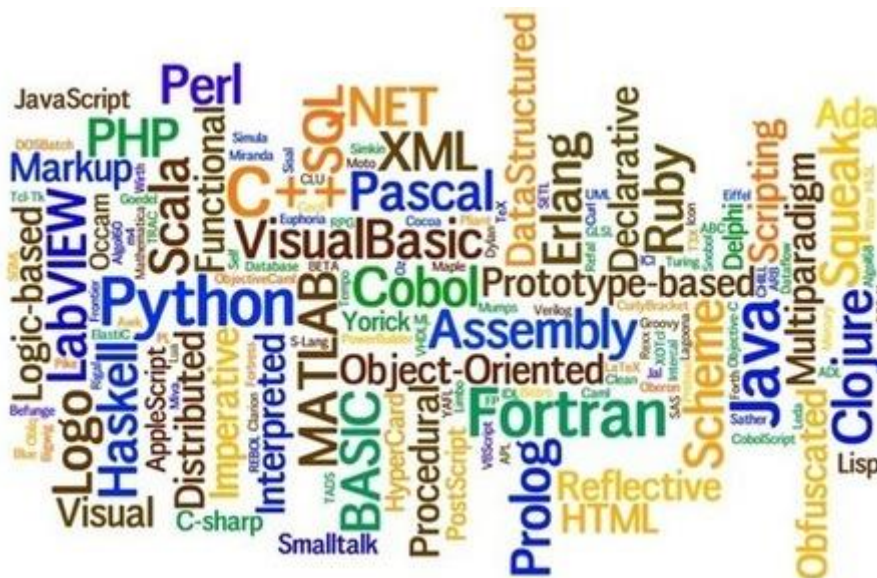
Al finalizar el estudio de la unidad, conocerás los diferentes tipos de lenguajes de programación, los procesos de compilación y depuración de los programas, así como comprender y aplicar la metodología de desarrollo, teniendo en cuenta sus diferentes etapas. Además, comprenderás la importancia de la documentación en el desarrollo del software.

1.INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN

Para poder llevar a cabo lo solicitado por el cliente, debes de recordarle a tu equipo la importancia de tener claro las diferentes cuestiones que caracterizan a los lenguajes de programación, de este modo, sobre todo, para los nuevos empleados que se acaban de incorporar, realizarás una pequeña introducción a dichos lenguajes.

Se entiende por **programa informático** a una serie de comandos ejecutados por el equipo que permiten obtener, modificar y mostrar información. En otras palabras, gestionar la información.

Un programa, o una parte de un programa, estará formado por es un simple archivo de texto (escrito a través de procesador, editor de texto, o entorno de desarrollo integrado-IDE), llamado archivo fuente o código fuente. Los ordenadores, a través de su CPU, sólo son capaces de procesar código binario, es decir, una serie de 0s y 1s, por tanto, necesita de un lenguaje de programación para escribir de manera legible, esto quiere decir, escribir a través de comandos, que facilite la comprensión de que lo que se necesita ejecutar.

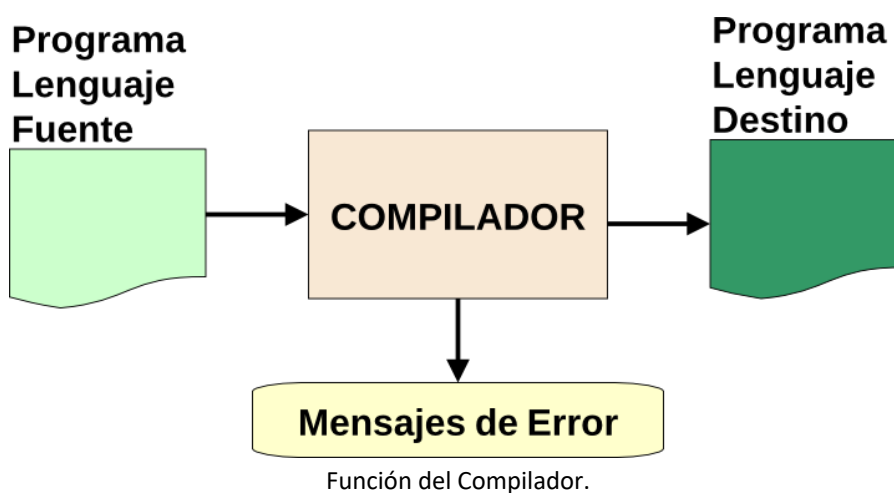


Collage de diferentes lenguajes de programación.

Fuente: <https://www.hackingpublico.net/lenguajes-de-programacion-i/>

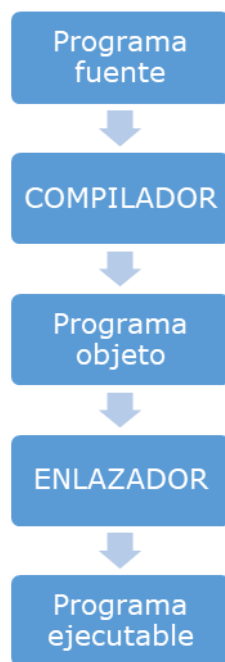
A través de un lenguaje de programación se realiza la escritura de un programa, y está formado por conjunto de símbolos y palabras reservadas que permitirán al programador aportar instrucciones a un ordenador para que sean ejecutadas. El primer paso a realizar, por tanto, es escribir el programa en un lenguaje de programación. Se recomienda que este lenguaje seleccionado sea bien conocido por el programador.

Una vez realizado, el programa escrito se va a traducir al lenguaje máquina. Este proceso es al que se conoce como **compilación**. Cada lenguaje de programación tiene su propio compilador (excepto los lenguajes interpretados).



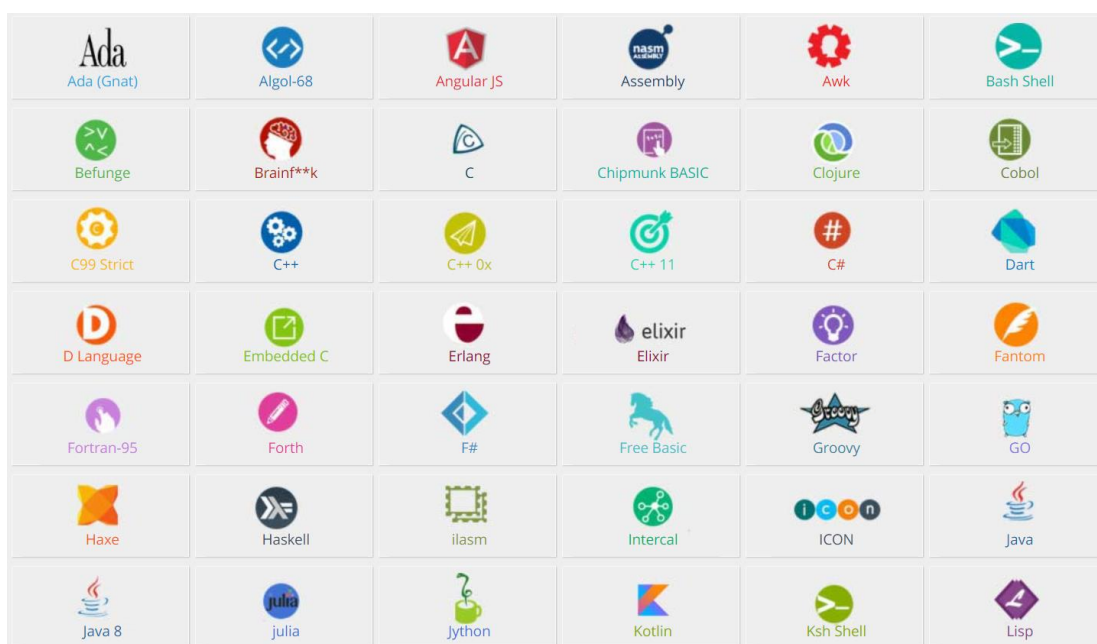
El archivo que contiene el código fuente incluye el código del programa que ha escrito el programador. Este archivo, debe compilarse una vez se ha completado. La compilación se realiza en dos pasos:

- El compilador transforma el código fuente en código objeto y lo guarda en un archivo objeto, es decir, traduce el archivo fuente a lenguaje máquina. Algunos compiladores crean un archivo en ensamblador (lenguaje de segunda generación cercano al lenguaje máquina), se trata de un lenguaje similar al lenguaje máquina ya que posee las funciones básicas, con la diferencia de que puede ser leído por el ser humano.
- Seguidamente, el compilador llama a un editor de vínculos (enlazador o ensamblador) que permite insertar los elementos adicionales (funciones y bibliotecas) a los que hace referencia el programa dentro del archivo final.



Compilación.

El resultado será un archivo ejecutable, que será el punto inicial donde arrancará la aplicación.



Lenguajes de programación.

Fuente: <https://www.tutorialspoint.com/codingground.htm>



PARA SABER MÁS

Para ampliar información sobre el proceso de compilación se recomienda descargar el siguiente material:



EJEMPLO PRÁCTICO

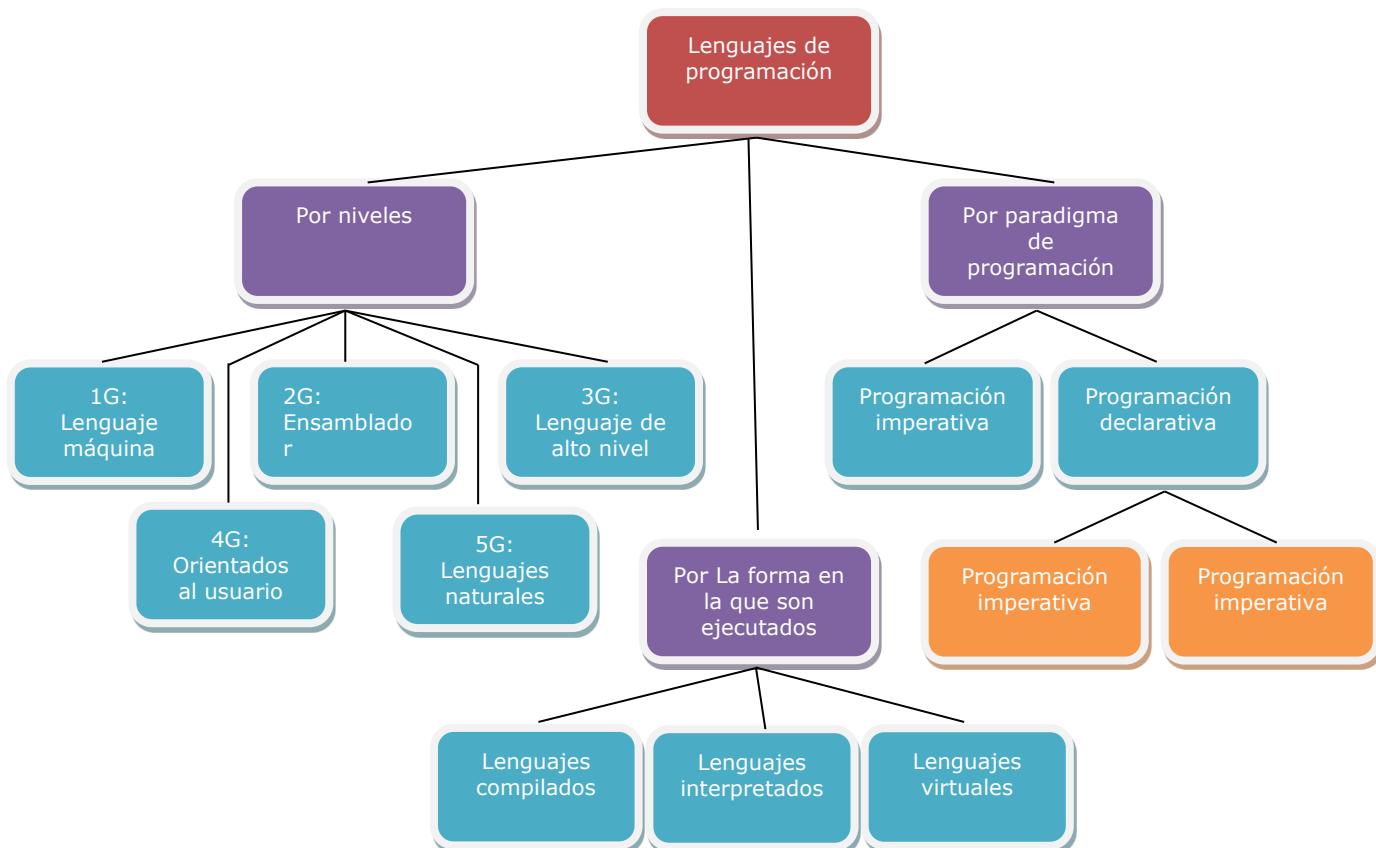
Se procede a implementar un programa Java que permite saber si un número introducido por pantalla es o no número primo. Una vez escrito el programa, ¿cuál es la primera tarea a realizar?

Solución:

Se deberá proceder a la compilación del mismo para conocer si existe algún tipo de error y poder subsanarlo. Si no se encuentran errores, se procede a su ejecución.

2. TIPOS DE LENGUAJES DE PROGRAMACIÓN

A la hora de poder realizar lo planteado por el cliente, es importante elegir un lenguaje de programación adecuado a lo que se pretende realizar, para ello es necesario plantear una reunión de equipo y que entre todos los integrantes se vea que lenguaje de programación es el más idóneo para el proyecto según sus características.



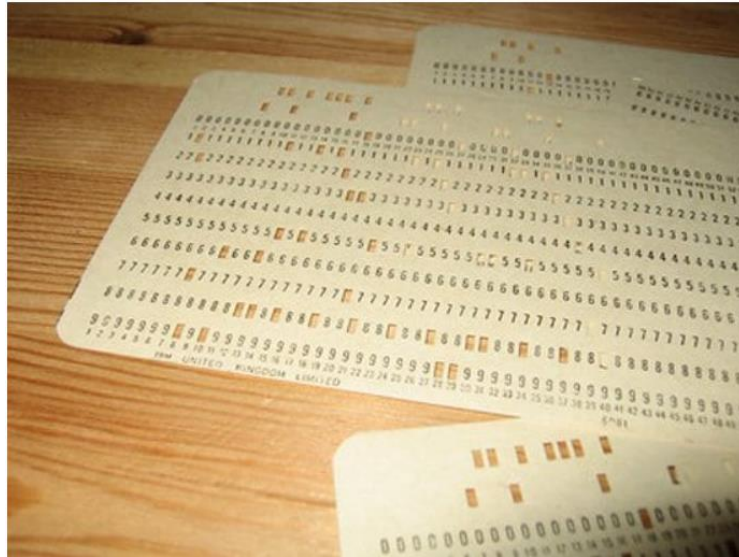
Tipos de lenguajes de programación.

Fuente: propia.

2.1 Según niveles

En esta clasificación, los lenguajes se diferencian principalmente por el nivel al que pertenecen, donde en el nivel más bajo se encuentran los lenguajes que son interpretados por la computadora. Los lenguajes de programación de mayor nivel necesitarán de intérpretes o compiladores. Estos niveles, también pueden entenderse como división en el tiempo, el lenguaje máquina es el lenguaje más antiguo de comunicación entre las personas y un ordenador, mientras que los lenguajes de tipo natural son más actuales.

- **El lenguaje máquina (1ª generación 1G):** es el lenguaje de programación que entiende directamente la computadora o máquina. Este lenguaje de programación utiliza el alfabeto binario, es decir, el 0 y el 1. Con estos dos únicos dígitos, se forman las cadenas binarias (combinaciones de ceros y unos) formando las instrucciones entendibles por el microprocesador. Se utilizaban tarjetas perforadas para insertar el código en la máquina.



Tarjeta perforada.

Fuente: <https://www.tecnologia-informatica.com/las-tarjetas-perforadas-historia-de-la-computadora/>

- **Lenguajes de programación de bajo nivel (2ª generación 2G):** Son mucho más fáciles de utilizar que el lenguaje máquina, pero dependen mucho de la máquina o computadora. El lenguaje ensamblador fue el primer lenguaje de programación que trató de sustituir el lenguaje máquina por otro mucho más entendible para los seres humanos. Los lenguajes de bajo nivel consiguen crear programas muy rápidos, pero son muy difíciles de aprender y tienen el inconveniente de ser específicos de cada procesador, con lo que si se ejecuta en otro procesador se deberá reescribir.

```
[x] HOLAMU~1.ASM x
[+] HOLAMU~1.ASM
1  ;model small // Acosta el espacio de memoria
2  ;el punto y coma (;) define comentarios
3  -----
4  .model small ;se define el tipo de modelo, corto
5  .data ;aquí se definen las variables que se van a ocupar, es el segmento de datos
6
7  mensaje db 'Este es mi Hola mundo!$' ;Mensaje
8
9  .code ;bloque de código
10 inicio: ;se define un comienzo del cuerpo del programa
11
12 mov ax,@data ;movemos al registro dx, los datos a utilizar
13 mov ds,ax ; de nuevo modemos los registros de derecha a izquierda
14
15 ;en ah siempre se cargan las funciones, en este caso la 09h, permite desplegar un mensaje en pantalla
16 mov ah,09h
17 mov dx,offset mensaje ;movemos al registro dx el contenido de mensaje
18 ; se llama a la interrupcion 21h
19 int 21h ;interrupcion
20
21 ; con esta instruccion terminamos el programa
22 mov ax, 4c00h
23 int 21h ;interrupcion
24 .stack ;segmento de pila
25
26 end inicio ;termina el segmento definido como inicio
27
28
29
30
```

Hola mundo en lenguaje ensamblador.

Fuente: <https://jimmyvazz.medium.com/ensamblador-lenguaje-maquina-escribiendo-un-hola-mundo-beafd9848576>

- **Lenguajes de programación de alto nivel (3ª Generación 3G):** Este tipo de lenguajes de programación son, por regla general, independientes de la máquina, y se puede usar en cualquier otro computador con pocas modificaciones. Los lenguajes de programación de alto nivel necesitan de un programa intérprete, como se ha comentado anteriormente, llamado compilador, que traduzca este lenguaje de programación de alto nivel a uno de bajo nivel (lenguaje máquina) que la computadora pueda entender. De forma general, los lenguajes de programación de alto nivel son fáciles de aprender y suelen usar palabras del lenguaje natural (casi siempre palabras en inglés como FOR, WHILE, IF, etc.).



Lenguajes alto nivel.

Fuente: <https://www.tecnologia-informatica.com/lenguaje-de-alto-nivel-que-es-ejemplos/>

- **Lenguajes orientados al usuario (4ª Generación 4G):** El usuario que utiliza estos lenguajes genera de forma automática la mayor parte de las partes del programa. Por lo que el programador solo debe indicar que se debe hacer y no como hacerlo. La diferencia principal con un lenguaje 3G es que el programador del 3G debe decir lo que se debe hacer y programar como debe hacerse.

La mayoría de estos lenguajes disponen de una interfaz gráfica y sólo obligan al usuario o programador a usar instrucciones sencillas y fáciles de manejar.

Estos lenguajes convierten las citadas especificaciones en:

- Lenguajes de tercer nivel que posteriormente un programador puede refinar.
- Lenguaje máquina, directamente.

Las herramientas CASE de diseño de bases de datos y modelado de procesos son claro ejemplo de estos lenguajes.

- **Lenguajes naturales (5ª Generación 5G):** Son lenguajes que se asemejan más al lenguaje humano que los anteriores.

Estos lenguajes aún se encuentran en sus inicios, aunque la mayoría de herramientas de trabajo y ordenadores tiende a tener estos lenguajes.

2.2 Según paradigmas de programación





Otra clasificación de los lenguajes de programación es según su paradigma. Los paradigmas de programación son propuestas tecnológicas y se enfocan a resolver problemas definidos y delimitados.

- **Programación Imperativa:** Con este paradigma se resuelven los problemas especificando una secuencia de acciones a realizar a través de uno o varios procedimientos denominados funciones o subrutinas. Las sentencias modifican el estado del programa (el estado de un programa viene definido por la instrucción que se está ejecutando y el valor de los datos que está tratando en un momento dado). Ejemplo de programación imperativa: Pascal, C.
- **La Programación Orientada a Objetos y la Programación Estructurada** forman parte de la programación imperativa, ya que, al programar con algunos de estos paradigmas, se describe la secuencia que debe seguir el programa para resolver un problema dado.

Ejemplo de programación orientada a objetos: JAVA.

La diferencia principal entre estos dos lenguajes de programación es que la programación estructurada tiene subrutinas (funciones y/o procedimientos), y en la programación orientada a objetos, se programa pensando en objetos del mundo real, coches, aviones, personas, donde se crearan clases de estas que contendrán características de esos objetos (atributos) y comportamientos (métodos) con los que el programa trabajara con esos objetos.

- **Programación Declarativa:** Con la programación declarativa, la solución es alcanzada a través de mecanismos internos de control, no se especifica exactamente cómo llegar a ella. En la programación declarativa, una expresión puede ser sustituida por el resultado de ser evaluada. Dentro de la programación declarativa se engloban la programación funcional, cuyo lenguaje más utilizado es Haskell y la programación lógica con Prolog como principal lenguaje utilizado.

Aug 2023	Aug 2022	Change	Programming Language	Ratings	Change
1	1		 Python	13.33%	-2.30%
2	2		 C	11.41%	-3.35%
3	4	▲	 C++	10.63%	+0.49%
4	3	▼	 Java	10.33%	-2.14%
5	5		 C#	7.04%	+1.64%
6	8	▲	 JavaScript	3.29%	+0.89%
7	6	▼	 Visual Basic	2.63%	-2.26%
8	9	▲	 SQL	1.53%	-0.14%
9	7	▼	 Assembly language	1.34%	-1.41%
10	10		 PHP	1.27%	-0.09%

Lenguajes de programación más populares agosto 2023.

Fuente: <https://www.tiobe.com/tiobe-index/>

2.3 Según la forma en la que son ejecutados

Dependiendo de la forma en la que se ejecuten los programas creados, podemos tener tres tipos distintos de lenguajes:

- **Lenguajes compilados:** Necesitan un programa traductor, llamado compilador, para convertir el código fuente en código máquina (en su totalidad). Son los lenguajes que más rápidos se ejecutan. Junto con el compilador, se necesita de un programa llamado enlazador o linker que permite unir el código objeto del programa con el código objeto de las librerías que necesite.
- **Lenguajes interpretados:** No se genera el código objeto, se va interpretando instrucción a instrucción, cargando en memoria y ejecutando. Las instrucciones se ejecutan “en directo” y solamente aquellas instrucciones que van a ejecutarse, a diferencia de los lenguajes compilados que son todas las instrucciones.

- **Lenguajes virtuales:** Es un código intermedio o **bytecode**, es un código que puede ser interpretado por una máquina virtual instalada en cualquier equipo. Tienen una ejecución lenta, pero tienen la particularidad de que pueden ser portables a cualquier sistema, siempre que tenga una versión de la máquina virtual asociada.

2.4 Según los lenguajes de programación más difundidos

Y, según los lenguajes de programación más difundidos, podemos tener distintos tipos:

Java

Java surge como una evolución de C++ adaptado a nuevas necesidades y conectividad, en sus comienzos se consideró el lenguaje de internet:

Algunas de sus características:

- El manejo de cadena de caracteres o string es mucho más sencillo en JAVA que en C o C++.
- No se permite la sobrecarga de operadores ni herencia múltiple.
- Es un lenguaje orientado a objetos del mundo real. Como por ejemplo coches, personas, alumnos, etc.
- Está pensado para trabajar con protocolos de redes.
- Es un lenguaje virtual interpretado.
- Es un lenguaje portable, ya que necesita de una máquina virtual que puede estar instalada en sistemas operativos distintos.
- Permite multihilo de ejecución en un mismo programa

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

Programa "Hola Mundo" en JAVA.

Fuente: Propia.

Python

Fue creado por Guido Van Rossum, recibió su nombre por la afición de su creador a los Monty Python. El Big Data y la inteligencia artificial se apoyan mucho en este lenguaje. Aunque es más viejo que JAVA, es ahora con la proliferación de estas últimas tecnologías cuando se ha puesto más de moda. Algunas de sus características son:

- Lenguaje de alto nivel, portable, ya que, si se evita la dependencia de ciertas librerías, el programa en Python puede ejecutarse en distintas máquinas.
- Es un lenguaje interpretado. Tiene en común con JAVA la creación de un programa intermedio en bytecode que luego se traduce y se ejecuta en la máquina.
- Está orientado a objetos.
- Permite combinar con C parte del código para poder ejecutar un programa más rápido.
- Es uno de los lenguajes más simples y sencillos de aprender.

```
print("Hola mundo!!")
```

C y C++

Son de los lenguajes más usados de la historia por su potencia y versatilidad. Mientras que otros lenguajes de la época se han dejado de utilizar, estos siguen usándose en múltiples entornos de desarrollo. La mayoría de sistemas operativos actuales tienen una base sólida en estos lenguajes, o derivados. Algunas de las características de estos lenguajes son:

- Lenguajes estructurados y muy ligados a funciones.
- Uso de punteros → variables que contienen la dirección de memoria de otra variable.
- Se combinan comandos de alto nivel con comandos de bajo nivel.
- Programas eficientes y rápidos.
- Programas compilados de pequeño tamaño.
- Lenguajes relativamente portables. Se deben recompilar en la máquina en la que se quieran ejecutar.


```
#include <iostream>

int main ()
{
    std::cout << "Hola, mundo";
    return 0;
}
```

JavaScript

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

Algunas de las características de este lenguaje son:

- Se parece mucho a Java, C y C++.
- Es un lenguaje de scripting.
- Se ejecuta del lado del cliente, lo que hace que el código sea visible y cualquier persona pueda leerlo
- Seguro y fiable.

```
<script>
    document.write("Hola Mundo");
</script>
```

PHP

Lenguaje Web utilizado principalmente para el backend de muchos productos como PrestaShop, Wordpress, etc.

- Lenguaje multiplataforma. Instalación en cualquier sistema.
- Buena integración con Mysql.
- Permite orientación a objetos.
- Lenguaje interpretado.
- Existen muchos frameworks basados en PHP que permiten trabajar patrones con el modelo-vista-controlador (MVC)

```
<?php
    echo "¡Hola mundo!";
?>
```

VB.NET

Evolución del famoso lenguaje Visual Basic de Microsoft. Algunas de sus características son:

- Se utiliza para programar y crear macros en Microsoft Office.
- Lenguaje orientado a objetos implementado sobre el framework .NET
- La mayoría de programadores que lo usan, utilizan el Microsoft Visual Studio

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    MsgBox("Hola Mundo")
End Sub
```



ENLACE DE INTERÉS

Visualiza una comparativa de 6 lenguajes de programación actuales en este enlace:



EJEMPLO PRÁCTICO

Se procede a implementar un programa para distintos sistemas operativos de escritorio, para un concesionario de coches. Conociendo las especificaciones de los lenguajes de programación, ¿Qué lenguaje de programación se decidiría?

Solución:

Si se desea que el programa pueda ser ejecutado en sistemas operativos Apple, Linux y Windows, por ejemplo, JAVA sería el lenguaje elegido, ya que para su ejecución es necesaria la máquina virtual, y java tiene máquina virtual para todos esos sistemas operativos.

3. COMPILADORES Y DEPURADORES

Dentro del proyecto solicitado, es importante saber distinguir entre compiladores y depuradores, ya que nos permitirá también distinguir entre unos lenguajes de programación u otros, y dependiendo la prisa que se tenga en acabar el proyecto puede ser clave para la elección del lenguaje de programación y del IDE con el que se va a desarrollar.

Un **compilador** es un programa que traduce un software escrito en un lenguaje de alto nivel, lenguaje de cuarta generación o de quinta generación, denominado programa fuente, en otro denominado programa objeto.

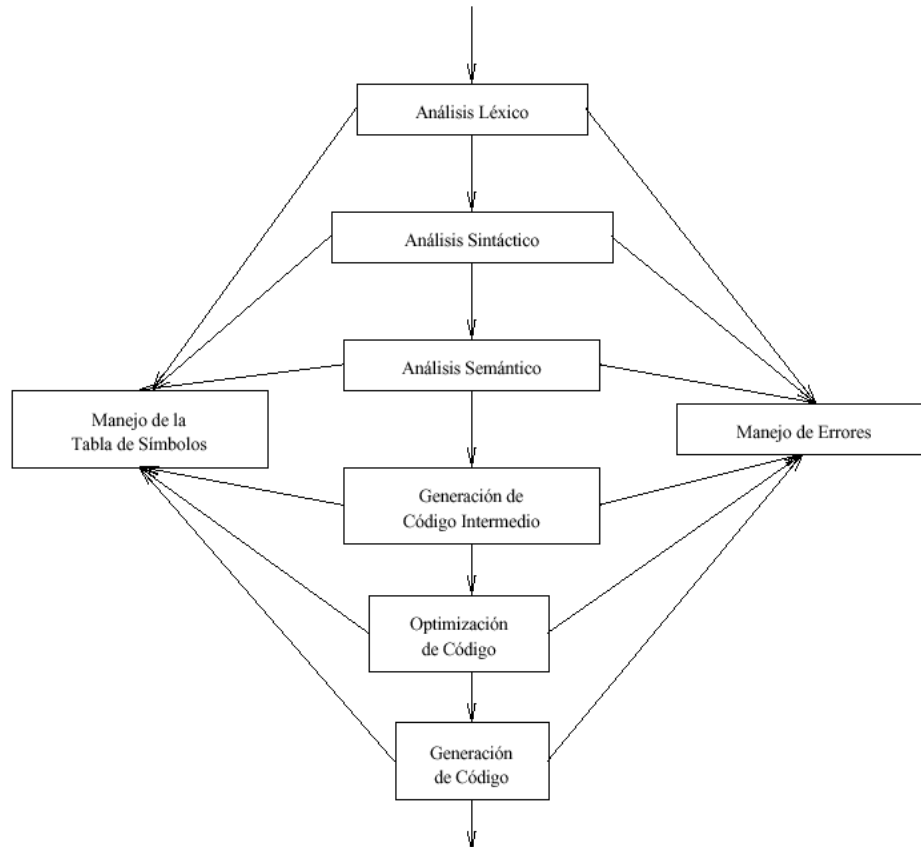


Esquema del Compilador.

El programa objeto puede almacenarse en memoria auxiliar para ser ejecutado posteriormente sin necesidad de volver a realizar la traducción. La traducción de un compilador consta de 2 etapas: la etapa de **análisis** y la etapa de **síntesis**.

La compilación es un proceso complejo que consume a veces un tiempo superior a la ejecución del propio programa. En cualquiera de las fases, el compilador puede generar mensajes de error detectados en el programa fuente, cancelando en ocasiones la compilación, siendo necesario que el programador realice las correcciones correspondientes en el programa fuente.

En la fase de análisis se ejecutan tres fases fundamentales: el análisis léxico, el análisis sintáctico y el análisis semántico. Cada una de estas fases la realiza un analizador determinado. La síntesis del programa objeto conduce a la generación de código intermedio, su optimización y la generación del código final.



Fases de la compilación.

Fuente: <http://robots-men.blogspot.com/2010/05/compiladores.html>

3.1 Análisis

Existen diferentes tipos de analizadores:

Analizador léxico: Examina el programa fuente para localizar las unidades básicas de información pertenecientes al lenguaje, denominadas **tokens**. Los tokens son caracteres o secuencias de caracteres que tienen un significado especial en el programa fuente (palabras reservadas, identificadores, etc.). El analizador léxico pasa por alto los comentarios, aísla los tokens, identifica el tipo y los almacena en unas tablas denominadas tablas de símbolos. Se puede considerar que, como resultado del análisis léxico, se obtiene una representación del programa formada por la descripción de símbolos en las tablas y una secuencia de símbolos junto con la referencia a la ubicación del mismo en la tabla. Esta representación contiene la misma información que el programa fuente, pero en una forma más compacta, no presentando ya el código como una secuencia de caracteres, sino de símbolos.

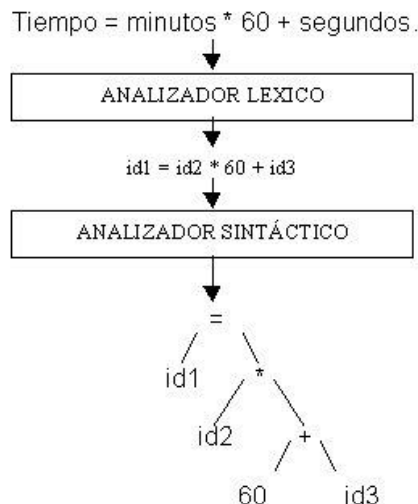
La información almacenada en las tablas de símbolos se completa y utiliza en las fases posteriores de la traducción. La tabla de símbolos es la estructura que almacena toda la información relativa a identificadores, variables, constantes, estructuras de datos y otros elementos pertenecientes al programa que se está compilando. Esta información suele incluir el tipo de cada elemento, sus dimensiones y otras características. La tabla de símbolos está relacionada con todas las fases del proceso de compilación y está presente en todas ellas, siendo utilizada por cada uno de los módulos. Algunas de las fases se ocupan de completar esta tabla mientras que otras utilizan la información en ella contenida.

Analizador sintáctico: La sintaxis de un lenguaje de programación especifica cómo deben escribirse los programas, mediante un conjunto de reglas de sintaxis o gramática del lenguaje. Un programa es sintácticamente correcto cuando sus estructuras están escritas en un orden correcto. Por ejemplo, las asignaciones normalmente en la mayoría de los lenguajes requieren que a la izquierda del operador de asignación aparezca un identificador y a la derecha del mismo aparezca una expresión, la cual deberá cumplir también un esquema especificado por un conjunto de reglas para las expresiones. De esta forma el analizador sintáctico recibe la cadena de tokens obtenida por el analizador léxico y busca en ella los posibles errores sintácticos que aparezcan. Entre los posibles errores que se pueden presentar están la duplicidad de identificadores de variables o las instrucciones escritas incorrectamente.

Un analizador sintáctico es un programa que reconoce si una o varias cadenas de caracteres forman parte de un determinado lenguaje. Se han definido varios sistemas para definir la sintaxis de los lenguajes de programación.

Entre ellos, cabe destacar la notación BNF (Backus–Naur Form) y los diagramas sintácticos. Entre los métodos para realizar el análisis sintáctico, uno bastante sencillo es realizar un procedimiento para reconocer cada uno de los símbolos terminales del lenguaje. Con este esquema de análisis el analizador puede generar mensajes de error cuando se encuentre un símbolo no esperado. Los lenguajes habitualmente reconocidos por los analizadores sintácticos son los lenguajes libres de contexto.

Cabe notar que existe una justificación formal que establece que los lenguajes libres de contexto son aquellos reconocibles por un autómata de pila, de modo que todo analizador sintáctico que reconozca un lenguaje libre de contexto es equivalente en capacidad computacional a un autómata de pila. Los analizadores sintácticos fueron extensivamente estudiados durante los años 70 del siglo XX, detectándose numerosos patrones de funcionamiento en ellos, lo que permitió la creación de programas generadores de analizadores sintácticos a partir de una especificación de la sintaxis del lenguaje.



Ejemplo analizador léxico y sintáctico.

Fuente: https://www.ecured.cu/Analizador_sint%C3%A1ctico

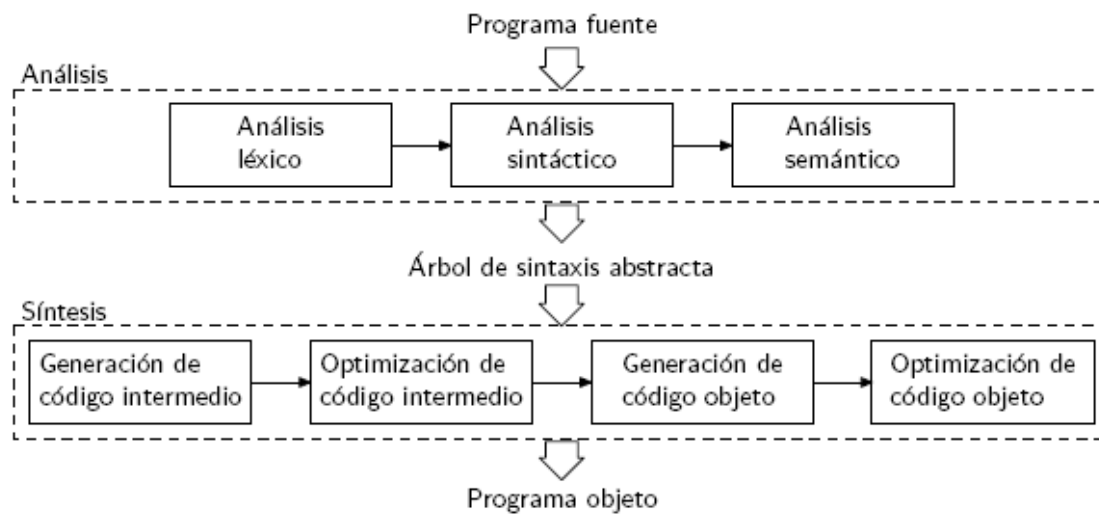
Analizador semántico: La fase de análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En el análisis semántico se realizan, entre otras funciones, la verificación de los tipos de las variables declaradas, comprobando si cada operador tiene operandos permitidos por la especificación del lenguaje. Al producirse un error en cualquiera de las fases de análisis, éste se comunica al programador mediante un mensaje indicativo del error producido y el lugar en que éste ocurrió. En algunos casos, ciertos errores no afectan gravemente al proceso de ejecución, e incluso permiten la ejecución del programa. Este tipo de errores se denominan advertencias o warnings.

3.2 Síntesis

En la fase de síntesis se lleva a cabo la generación de código intermedio, la optimización de código y la generación del código final.

Generación de código intermedio: Si no se han producido errores en alguna de las etapas anteriores, este módulo realiza la traducción a un código interno propio del compilador, denominado código intermedio, a fin de permitir la transportabilidad del lenguaje a otros ordenadores. Para un determinado lenguaje de alto nivel se hace común todo el proceso de análisis y generación de código intermedio y es la generación de código objeto la que se particulariza para cada tipo de microprocesador.

Optimización de código: La misión del optimizador de código consiste en recibir el código intermedio y optimizarlo atendiendo a determinados factores, tales como la velocidad de ejecución o el tamaño del programa objeto. Se realiza una optimización, analizándose el programa objeto globalmente. Existen compiladores que permiten al usuario omitir o reducir las fases de optimización, disminuyéndose así el tiempo global de la compilación. Tras la optimización del código intermedio se lleva a cabo la generación del código final del programa. Esta fase se basa en una traducción del código intermedio optimizado.

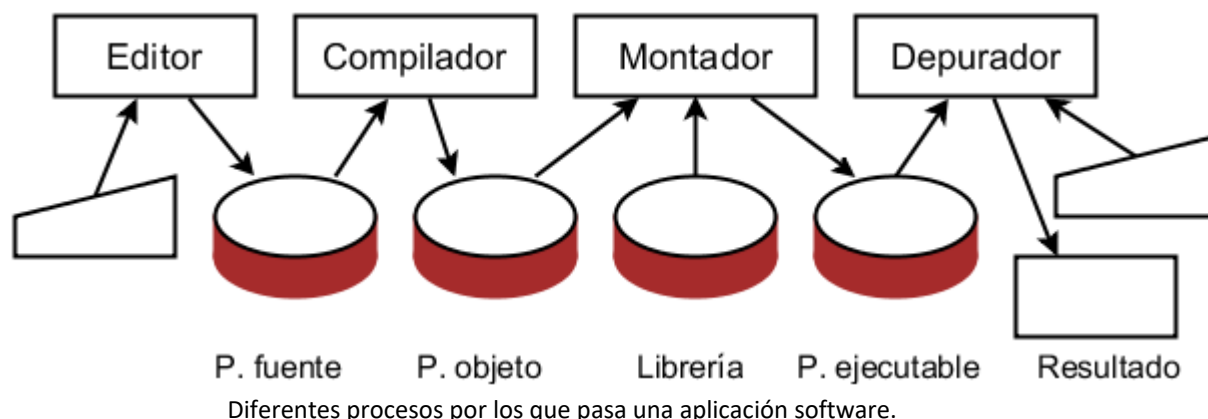


Fases de análisis y de síntesis.

Fuente: <https://loscompiladores.wordpress.com/category/procesos-postcompilacion/>

3.3 Depuradores

Los **depuradores** facilitan la detección y, en muchos casos, la recuperación de los errores producidos en las diferentes fases de la compilación. El compilador, cuando detecta un error, trata de buscar su localización exacta y su posible causa para presentar al programador un mensaje de diagnóstico, que será incluido en el listado de compilación.



Fuente: <https://slideplayer.es/slide/1715003/>



EJEMPLO PRÁCTICO

Se va a crear un programa que nos ha encargado la empresa “Los PEPES S.A.” Se está pensando en elegir el lenguaje de programación dependiendo si el código es interpretado o es compilado, ¿qué lenguaje de programación se elegiría teniendo en cuenta que no hay mucho tiempo para la creación de la aplicación?

Solución:

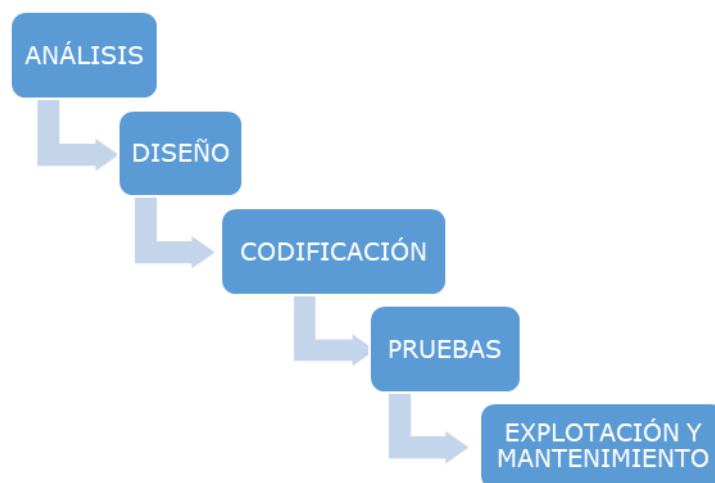
Dado que no hay mucho tiempo para la creación de la aplicación, la mejor opción sería un lenguaje de programación interpretado. Los lenguajes interpretados no requieren de un paso de compilación, por lo que el código se puede ejecutar directamente. Esto puede ahorrar un tiempo considerable en el desarrollo de la aplicación.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Volviendo a nuestro proyecto sobre una aplicación para gestionar una biblioteca, es imprescindible que nuestro equipo de desarrollo conozca las fases en el ciclo de vida del software, ya que se pasaran por ellas durante todo el proyecto que se va a desarrollar, y en futuros proyectos

Existen muchos paradigmas de desarrollo de aplicaciones, que dependiendo del autor, del responsable del equipo, etc., suelen dividirse en más o menos etapas, agrupando algunas o desagrupando otras. Al final el concepto de lo que debe hacerse queda reflejado en las etapas que se van a desarrollar en este apartado.

Las 5 **fases o etapas** del desarrollo de una aplicación software son:



Fases del desarrollo de una aplicación.

1. Análisis

En esta fase se establece el producto a desarrollar, siendo necesario especificar los procesos y la estructura de datos, en definitiva, la funcionalidad del software que se van a desarrollar en la **toma de requisitos**.

En esta fase debe existir una clara comunicación entre el cliente y el analista para poder conocer todas las necesidades que precisa la aplicación. En el caso de falta de información ya sea por falta de comunicación o conocimientos por parte del cliente, se puede recurrir al desarrollo de prototipos (software de pruebas que irán definiendo la aplicación final) para saber con más precisión sus requerimientos y funcionalidades.



CITA

El principio de Pareto establece que “el 80% del tiempo de elaboración de un programa se debe destinarlo a la planificación y el 20% restante a la codificación”.

En la fase de análisis se pueden definir las siguientes etapas:

Diagramas de flujo de datos

- Sirven para conocer el comportamiento del sistema mediante representaciones gráficas.

Modelos de datos

- Sirven para conocer las estructuras de datos y sus características. (Entidad relación y formas normales)

Diccionario de datos

- Sirven para describir todos los objetos utilizados en los gráficos, así como las estructuras de datos

Definición de los interfaces de usuario

- Sirven para determinar la información de entrada y salida de datos.

Etapas de la fase de análisis

Al final de esta fase se han de tener claro las especificaciones, es decir, las funcionalidades y requerimientos a desarrollar en la aplicación.

2. Diseño

En esta fase se alcanza con mayor precisión una solución óptima de la aplicación, es decir, se deben tener en cuenta los recursos físicos del sistema (tipo de equipos, servidores, periféricos, comunicaciones, etc.) y los recursos lógicos (sistemas operativos, programas de utilidad, lenguaje/s de programación, bases de datos, etc.) donde se ejecutará la aplicación. En el diseño estructurado se definen las siguientes etapas:

Diseño externo:

- Se especifica la interfaz para los usuarios, formatos de información de entrada y salida (pantalla y listados de información, gráficos, etc.).

Diseño de datos:

- Establece las estructuras de datos de acuerdo con su soporte físico y lógico.

Diseño modular:

- Es una técnica de representación en la que se refleja de forma descendente la división de la aplicación en módulos normalmente a través de interfaces, siguiendo las pautas de algún diseño de patrones de software.

Diseño procedimental:

- Establece las especificaciones para cada módulo, escribiendo el algoritmo necesario que permita posteriormente una rápida codificación. Se emplean técnicas de programación estructurada, normalmente ordinogramas y pseudocódigo.

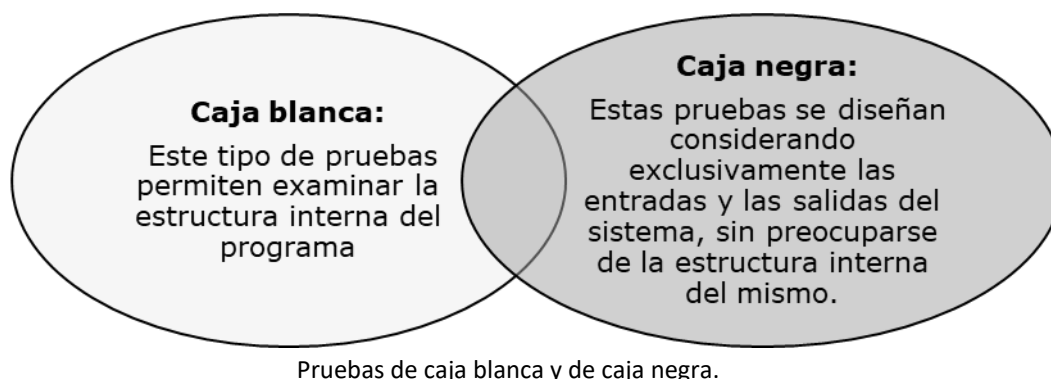
Etapas de la fase de diseño.

3. Codificación

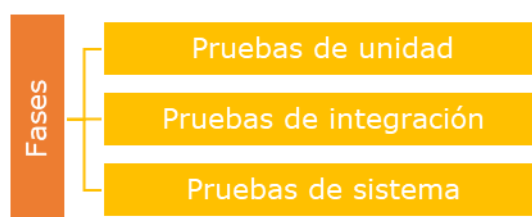
Una vez se tienen claras las funcionalidades y los requerimientos y se ha diseñado como será la aplicación a nivel de módulos, clases, etc. en la etapa de diseño, se procede a traducir y crear el código fuente en el lenguaje de programación que se decidió en la etapa de diseño, además del entorno u entornos de desarrollo y las tecnologías a utilizar. La implementación de código implica realizar pruebas de funcionamiento para comprobar si el software realiza las funciones programadas. No obstante, en la siguiente fase se realizan pruebas más exhaustivas y de mayor precisión.

4. Pruebas

Tras la codificación de la aplicación se deben de realizar las pruebas necesarias para comprobar la calidad y estabilidad del programa. Existen dos métodos generales en el diseño de pruebas de programas:



Se pueden distinguir 3 fases en el proceso de pruebas:



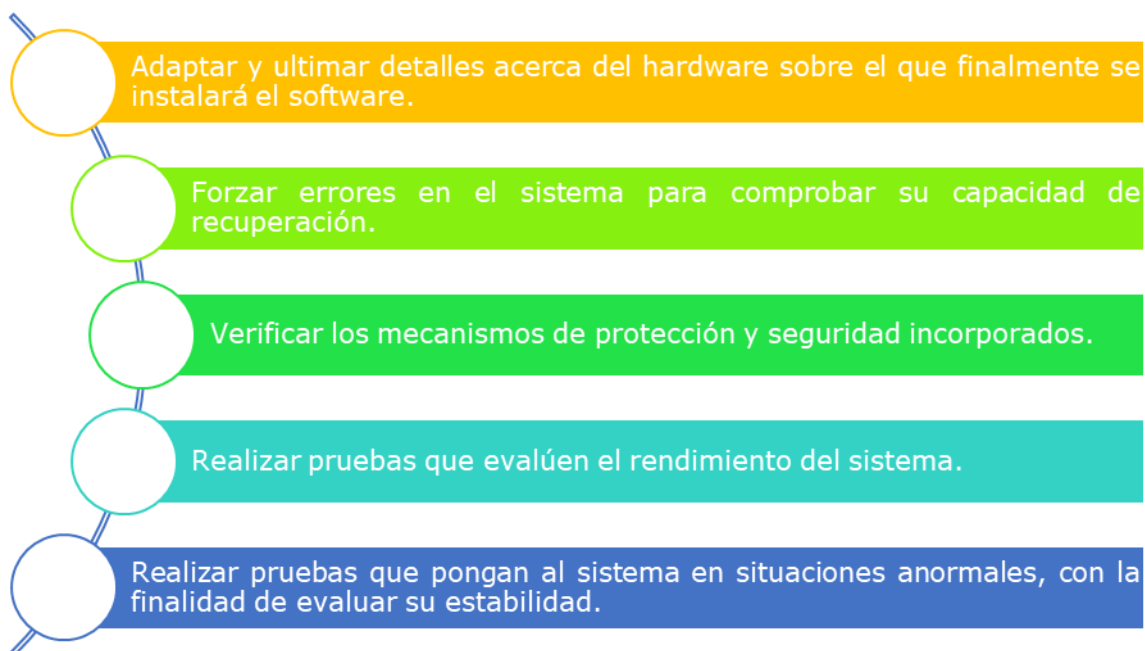
Pruebas de unidad: Están destinadas a la prueba de cada una de las unidades del sistema, esto es, se centran en la verificación de la menor unidad en el diseño del software (el módulo). Se verifican por tanto las estructuras de datos, la interfaz, las estructuras de control, el control de errores, las condiciones límite de funcionamiento y la eficiencia del código. Dentro de cada una de éstas, las pruebas se centrarán en unos u otros aspectos.

Pruebas de integración: Una vez que se ha comprobado la corrección de errores en cada uno de los módulos por separado con las pruebas de unidad, la siguiente fase de pruebas va dirigida a probar la correcta interconexión de los módulos, ya que esto no queda garantizado con las pruebas de unidad. Algunos de los problemas que pueden surgir son la modificación no deseada de variables globales (causando posibles efectos laterales) y el acarreo de imprecisiones por parte de varias funciones involucradas en el mismo cálculo. La integración de los módulos ha de realizarse de manera incremental, es decir, el programa se irá construyendo añadiendo los módulos poco a poco, de manera que los posibles errores sean fácilmente localizables y corregibles en cada uno de los ellos. Según la estrategia de integración elegida, se pueden distinguir entre una integración entre módulos **ascendente** o bien una integración **descendente**.

- La integración **ascendente** comienza con la construcción y prueba de los módulos de más bajo nivel y va realizando la integración de los mismos en módulos mayores, es decir, de manera ascendente. Los programas probados siguiendo esta estrategia no existen hasta que se han añadido y probado todos los módulos. Además, se consume mucho tiempo realizando pruebas para cada módulo que se genera, no siendo útiles muchos de los programas realizados para realizar dichas pruebas.
- En la integración **descendente** se construye primero el programa principal, y a éste se le van incorporando los módulos conforme se vayan implementando. Esta estrategia permite que las interfaces de nivel más alto sean las primeras en probarse. Pueden surgir problemas cuando se requiere el procesamiento de algún módulo de nivel inferior para poder probar adecuadamente los de nivel superior. Puede ser difícil encontrar datos de entrada del nivel más alto que sirvan para probar correctamente un módulo de nivel inferior.

La selección de una u otra estrategia depende de las características del software que se esté desarrollando. No obstante, muchas veces se opta por una solución mixta que emplea ambas estrategias. En estas ocasiones se suele utilizar una integración descendente para los módulos superiores y una ascendente para los módulos de más bajo nivel.

Pruebas de sistema: En esta fase, el sistema construido debe quedar en perfecto estado de funcionamiento. Para ello se realizan diversas tareas como:



Tareas de las pruebas de sistema.

5. Explotación y Mantenimiento

La **explotación** consiste en realizar la implantación de la aplicación en el sistema o sistemas físicos donde van a funcionar habitualmente (entorno de producción) y su puesta en marcha para comprobar el buen funcionamiento:

Instalación del/los programa/s.

Eliminación del sistema anterior.

Conversión de la información del antiguo sistema al nuevo (si la hubiera).

Pruebas de aceptación al nuevo sistema.

Actividades en la fase de explotación.

Al final de la explotación se le facilita al cliente toda la documentación necesaria para la explotación del sistema (manuales de ayuda, de uso, guía de la aplicación, información sobre el soporte, etc.).

En cuanto al **mantenimiento**, es la fase que completa el ciclo de vida y en ella se deben solventar los posibles errores o deficiencias de la aplicación. Es posible que se necesiten aclarar nuevas especificaciones que el cliente no informó de forma correcta, lo cual implicaría reiniciar el ciclo de vida, siendo esta la opción menos deseable de todas. Existen fundamentalmente 3 tipos de mantenimiento:

Mantenimiento correctivo

- Consiste en corregir errores no detectados en pruebas anteriores y que aparezcan con el uso normal de la aplicación. Este mantenimiento puede estar incluido en la garantía o mantenimiento de la aplicación

Mantenimiento adaptativo

- Consiste en modificar el programa a causa de cambio de entorno gráfico y lógico en el que estén implantados. (Nuevas generaciones de ordenadores, nuevas versiones del sistema operativo, etc.)

Mantenimiento perfectivo:

- Consiste en una mejora sustancial de la aplicación al recibir por parte de los usuarios propuestas sobre nuevas posibilidades y modificaciones de las existentes

Tipos de mantenimiento.

Los tipos de mantenimiento adaptativo y perfectivo reinician el ciclo de vida, debiendo proceder de nuevo al desarrollo de cada una de sus fases para obtener un nuevo producto.



ENLACE DE INTERÉS

Conoce más información sobre el desarrollo software en esta página:



VÍDEO DE INTERÉS

Visualiza una explicación de lo que es el ciclo de vida del software



Durante todas las fases del desarrollo del software es muy importante ir documentando todo, utilizando tanto documentación externa como interna. Son muchos los programadores que aún no documentan el código internamente con los problemas que esto conlleva. Existen ocasiones en las que las empresas no disponen de informáticos especializados en la plantilla por lo que es de vital importancia dejarles un buen manual de instalación y uso de la aplicación. Además, por si en el futuro fuera necesario, disponer de todo el código documentado para que sea más entendible para posteriores cambios o mejoras por su parte o la de otro compañero es una práctica imprescindible.



EJEMPLO PRÁCTICO

Se observa que un software realizado por el equipo de desarrollo tiene cierta funcionalidad que parece ralentizar el equipo donde se encuentra instalado. Las diferentes opciones y acciones cumplen los requisitos del cliente, sin embargo, parece que el sistema operativo se encuentra en un estado de máximo consumo de recursos. ¿Cuáles son las pruebas que se han de realizar? ¿Cómo se puede proceder a resolver este tipo de inconvenientes?

Solución:

En primer lugar, es necesario realizar un chequeo del equipo informático donde se están realizando las ejecuciones. Es probable que el equipo necesite algún tipo de actualización software o revisión del hardware por si estuviera obsoleto.

Si el equipo se encuentra en un estado óptimo, sería recomendable realizar las mismas pruebas en un equipo diferente, y comprobar su funcionalidad. En función de ello, es posible obtener 3 resultados: O bien encontrar la misma lentitud que en el primer equipo, aumentar esa lentitud o reducirla. Si la lentitud es similar o incluso mayor, se hace necesario revisar el código de la aplicación software e ir analizando cada funcionalidad, ya que es posible que exista algún cuello de botella (acceso a una base de datos o servidor web, recursos hardware bajo mínimos, etc.). En caso de que la lentitud desaparezca, es conveniente realizar una comparativa entre los equipos de prueba, para intentar observar qué está ocurriendo en el primero de ellos y actuar en consecuencia.

Es importante tener en cuenta que existen aplicaciones software en las que va a haber un acceso concurrente importante (aplicaciones bancarias, consultas, etc.). Para ello es importante realizar un tipo de pruebas llamadas pruebas de rendimiento, capaces de detectar anomalías como las del ejemplo anterior.

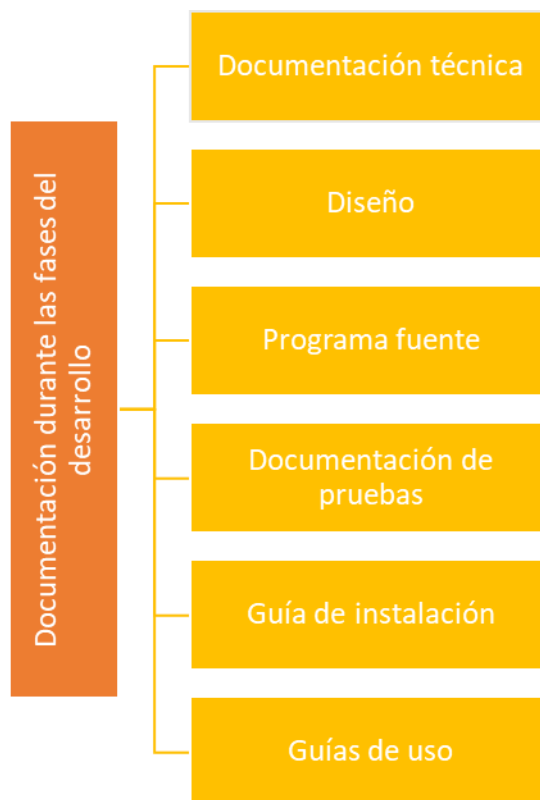


COMPRUEBA LO QUE SABES

¿Es lo mismo una prueba de estrés que una prueba de rendimiento? Coméntalo en el foro de la unidad.

5. DOCUMENTACIÓN DURANTE LAS FASES DEL DESARROLLO

Dentro de nuestros proyectos sobre la aplicación para una biblioteca, vamos a llevar una rigurosidad extrema en la parte de documentación ya que el cliente nos pide tener documentado todo el proceso, además de que se deben cumplir ciertos estándares de calidad que implican gran parte de documentación. Por otro lado, aun no tenemos ganado el concurso para el mantenimiento de la aplicación una vez que se cree, por lo que todo debe quedar documentado para las personas que vayan a llevar el mantenimiento, ya sea tu equipo, parte de tu equipo, otros equipos dentro de la empresa, u otras empresas.



Documentación durante las fases del desarrollo.

El ciclo de vida de una aplicación o las diferentes fases por las que pasa son: análisis, diseño, codificación, pruebas, explotación y mantenimiento.



CITA

"If your program isn't worth documenting, it probably isn't worth running" J. Nagler. 1995, Coding Style and Good Computing Practices.

1.Documentación técnica

La guía técnica o manual técnico es el documento donde queda reflejado el diseño del proyecto o aplicación, la codificación de los programas y las pruebas realizadas para su correcto funcionamiento. Este documento está destinado al personal técnico en informática (analistas y programadores). El principal objetivo de este documento es el de facilitar el desarrollo, la corrección y el futuro mantenimiento de los programas de forma rápida y precisa.

2. Diseño

Es el documento donde queda reflejada la solución de la aplicación, basándose en las necesidades del usuario y en el análisis efectuado previamente por los analistas con el fin de obtener los requerimientos de la aplicación. Este documento está destinado a los programadores que van a desarrollar los programas de la aplicación, con objeto de que la codificación sea efectuada con la mayor calidad y rapidez posible. Este documento debe estar realizado de tal forma que permita la división del trabajo de programación, para que varios programadores puedan trabajar de forma independiente, aunque coordinados por uno de los programadores o por un analista. En el cuaderno de carga hay que ajustarse a las características del sistema físico donde se va implantar dicha aplicación, así como al tipo de sistema operativo y lenguaje que se va a utilizar. El diseño de la aplicación se puede dividir en las siguientes etapas:

Tratamiento general: describe la solución informática escogida para resolver la aplicación, adaptándola al sistema físico en el que se va a implementar.

Diseño de entrada-salida: el objetivo de esta etapa es obtener el diseño externo de pantallas e impresos utilizados por la aplicación. La presentación de este diseño es esencialmente gráfica, pudiendo ir acompañada por las descripciones de los controles que se van a efectuar sobre los campos que hay que presentar.

Diseño de datos: se obtienen las especificaciones de los ficheros de datos que se utilizan en la aplicación.

Diseño modular: el objetivo de esta etapa es obtener la estructura y descripción de los módulos principales de la aplicación, así como la división en distintos programas.

Diseño de programas: el objetivo de esta etapa es obtener de forma independiente todas las especificaciones de cada uno de los programas en que se ha dividido la aplicación, con el objeto de facilitar su codificación, pruebas y mantenimiento.

Etapas en el diseño de la aplicación.

- **Programa fuente**

En el código fuente escrito en los programas se debe cuidar que éste sea claro, legible y bien documentado. Para ello se deben seguir una serie de normas referentes al estilo de escritura, las zonas en las que ubicar ciertas secciones del código y no escatimar en colocar comentarios en el código fuente. En el estilo de escritura se deben tener en cuenta las siguientes normas:

- Utilizar suficientes espacios y líneas en blanco para separar bien las expresiones y los bloques de código cuya funcionalidad sea diferente a la del código adyacente en el mismo fichero fuente.
- Utilizar siempre tabuladores para delimitar el ámbito de las estructuras de control, respetando el sangrado (espacios en blanco introducidos delante de una línea) de las estructuras anidadas.
- Utilizar paréntesis, ya que, en expresiones complicadas, resulta más rápido desglosar mentalmente la expresión si ésta tiene los paréntesis apropiados.
- Utilizar separadores visuales, como guiones entre comentarios, para separar los grandes bloques del programa, por ejemplo, los módulos entre sí y del programa principal. En algunos lenguajes se pueden colocar módulos diferentes en ficheros diferentes, ayudando a diferenciar completamente el código de unos y otros.

- Utilizar palabras representativas para los nombres de identificadores (variables y constantes), evitando nombres muy cortos (poca legibilidad) o muy largos (incómodos a la hora de programar, a la vez que alargan mucho la línea de código).
- Intentar simplificar el código, evitando expresiones complicadas en la medida de lo posible.
- Evitar las negaciones en las expresiones comparativas, la afirmación es normalmente más clara.
- Evitar grandes anidamientos si no son necesarios.
- Diferenciar claramente las zonas de inclusiones e importaciones, declaraciones de identificadores, procedimientos y funciones. Aunque algunos lenguajes no obligan a que las declaraciones ocupen un lugar específico, es buena costumbre establecerse un criterio estándar en este orden, de esta forma los atributos serán fácilmente localizables y modificables. Si el diseño presenta alguna estructura compleja, se deben usar comentarios explicativos que la aclaren, preferiblemente antes de dicho código.

En relación a los comentarios, se debe tener en cuenta lo siguiente:

- Clarifican la estructura de cada módulo. Se incluirán comentarios que clarifiquen la estructura de cada módulo, así como la función realizada por cada bloque de instrucciones y las variables y tipos.
- Son recomendables al principio de cada procedimiento o función, aclarando la funcionalidad del mismo, además es recomendable indicar en éstos también el significado de los argumentos tomados y el valor devuelto. En cualquier lugar del código en que se vaya a realizar una tarea complicada de comprender es recomendable la aclaración mediante un comentario explicativo.

- Son muy recomendables principalmente por dos razones:
 - El propio programador puede requerir la reutilización de un código escrito tiempo atrás, habiendo ya olvidado en la fecha presente el código que se escribió.
 - La programación no es tarea de una única persona, y al tener que compartir el código con otras personas debe aclarar de la mejor manera posible la forma en que ha escrito dicho código, para facilitar la tarea al resto del grupo.
- Se pueden distinguir dos tipos de comentarios, comentarios de prólogo y descriptivos:
 - Los comentarios de prólogo aparecen al principio de cada módulo y consisten en un resumen de la función que realiza, nombre del autor, versión del módulo, fechas de modificaciones, y la información adicional que se considere necesaria.
 - Los comentarios descriptivos se insertan dentro del código y describen en lenguaje natural que hace cada bloque de código.



RECUERDA

Cuando se realiza un proyecto software es muy importante documentarlo porque en ese momento se recuerda el porqué de todas las sentencias de código, pero ¿qué pasa si pasados unos meses se deben realizar algunos cambios en la codificación?

4. Documentación de pruebas

Las pruebas se pueden realizar en cualquiera de las fases de la aplicación, pudiéndose establecer tres tipos de pruebas:

Pruebas unitarias: Para cada módulo dentro de un programa

Pruebas de integración: Para cada uno de los programas con todos sus módulos

Pruebas de sistemas: Para todos los programas que componen la aplicación

Tipos de pruebas.

En el documento de juego de pruebas se pueden especificar el tipo de prueba, módulos y programas para los que se efectúan dichas pruebas, datos de entrada, datos previstos de salida y los datos reales de salida obtenidos en las pruebas.

La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. El hecho de realizar pruebas del software proporciona una garantía de calidad del software. Además, representa una revisión final de las especificaciones, del diseño y de la codificación. La prueba no puede asegurar la ausencia de errores en el programa, solo puede indicar su existencia. Un buen caso de prueba será aquel que muestre un error no descubierto hasta entonces. Se deben seguir los siguientes principios para la realización de pruebas:

Las pruebas deben ser llevadas a cabo por personas diferentes a las que desarrollaron el programa, tanto para verificar que el programa funciona correctamente, como para validar que ese programa ha sido concebido e interpretado correctamente.

Los casos de pruebas deben ser escritos tanto para condiciones de entradas inválidas o inesperadas como para condiciones válidas y esperadas.

La posibilidad de encontrar errores adicionales en una sección del programa es proporcional al número de errores ya encontrados en la misma sección.

Principios en la realización de las pruebas.



PARA SABER MÁS

Las pruebas de software son una fase muy importante para garantizar calidad en los programas. La calidad del software se puede medir y evaluar, de tal manera que un software pueda ser evaluado atendiendo al nivel de calidad que ofrece. Con esta premisa, se recomienda la lectura de este post.





PARA SABER MÁS

Conoce más sobre cómo estandarizar la evaluación de la calidad software en este enlace:



5. Guía de instalación

La guía de instalación o manual de explotación es el documento que contiene la información necesaria para poner en marcha el sistema diseñado y para establecer las normas de explotación.

Estas normas harán precisiones sobre las siguientes tareas:

- Pruebas de implantación de los programas en el sistema físico donde van a funcionar en adelante, con la colaboración entre usuarios y desarrolladores.
- Forma en que se van a capturar los datos existentes en el sistema anterior al que se va a implantar. Esto implicará un trasvase de información que puede estar en soportes de uso manual o en soportes informáticos que exijan una conversión del formato de la información y la realización de programas de captura de datos.
- Pruebas del nuevo sistema con toda la información capturada, lo que equivale a tener funcionando en paralelo los dos sistemas, corrigiendo los posibles errores de funcionamiento. Cuando estas pruebas sean satisfactorias, el nuevo sistema sustituye al anterior y entra en vigor la nueva documentación.



ARTÍCULO DE INTERÉS

La relevancia de las técnicas y tecnologías para la documentación del software. Calero, C., Piattini, M. y Moraga, M.A. (2010). Calidad del producto y proceso software. Madrid: RaMa.

6. Guía de uso

La guía de uso o manual de usuario es el documento que contiene la información precisa y necesaria para que los usuarios utilicen correctamente los programas de la aplicación. La información contenida en la guía de usuario proviene de la guía técnica de la aplicación, pero se presenta de forma que el usuario la comprenda con toda claridad, prescindiendo de toda la parte técnica de desarrollo y centrándose en los aspectos de la entrada y salida de la información que maneja la aplicación.

La guía de uso no debe hacer referencia a ningún apartado de la guía técnica, pues normalmente la guía técnica queda en poder de los desarrolladores y al usuario solamente se le proporciona las guías de uso y de instalación.

La guía de uso debe estar redactada con un estilo claro y en el caso de que se haga necesario el uso de terminología informática no conocida por los usuarios, debe ir acompañada por un glosario de términos informáticos. Este documento puede ser utilizado para completar la formación que los usuarios de la aplicación deben recibir para el correcto manejo de la misma.

La guía de uso puede contener la siguiente información:



Información de la guía de uso.



EJEMPLO PRÁCTICO

Se ha entregado un programa que nos ha encargado la empresa Los PEPES S.A. Además del programa, ¿qué documentación se debe entregar?

Solución:

La documentación técnica, los diagramas de diseño, el código fuente, la documentación de prueba, la guía de instalación, y la guía de uso.



COMPRUEBA LO QUE SABES

¿Cuáles son las diferencias entre la guía de uso y la guía de instalación? ¿Podrían unirse en un único documento? Coméntalo en el foro de la unidad.

6. METODOLOGÍAS ÁGILES

Dentro del equipo de desarrollo para la aplicación de gestión de una biblioteca se han planteado distintas metodologías para el desarrollo del software, saber el ciclo de vida, es solo el punto inicial, hay muchas metodologías, que combinan de distinta forma este ciclo de vida. Es importante estar al tanto de las metodologías ágiles, y sobre todo de SCRUM, que es la metodología que más se aplica en la actualidad a la hora de empezar un proyecto.

En los últimos años han aparecido una serie de metodologías para la realización de proyectos y trabajos en general. Estas metodologías se conocen como metodologías ágiles.

Por definición, las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

En esencia, las empresas que apuestan por esta metodología consiguen gestionar sus proyectos de forma flexible, autónoma y eficaz reduciendo los costes e incrementando su productividad.

La metodología ágil como la conocemos en la actualidad nació en el año 2001. En respuesta a los enfoques en cascada de la gestión de proyectos, en los cuales estos se organizan como series de secuencias lineales, un grupo de desarrolladores de software redactó el Manifiesto para el desarrollo ágil de software. En este documento, los programadores propusieron un nuevo enfoque para el desarrollo de software y describieron cuatro características fundamentales que deberían priorizarse por encima de otras cuestiones.

De acuerdo con lo que establecieron, los equipos de desarrollo ágil de software debían valorar:

- **Las personas y las interacciones** antes que los procesos y las herramientas
- **El software en funcionamiento** antes que la documentación exhaustiva
- **La colaboración con el cliente** antes que la negociación contractual
- **La respuesta ante el cambio** antes que el apego a un plan

Los autores aclaran que todos los puntos de la lista anterior tienen cierto valor inherente. Sin embargo, proponen que valorar los puntos de la izquierda (en negrita) antes que los de la derecha puede dar lugar a mejores resultados en el desarrollo del producto. El manifiesto no busca imponer un conjunto de prácticas, sino ser una guía que permita pensar en el desarrollo de software de otra manera.

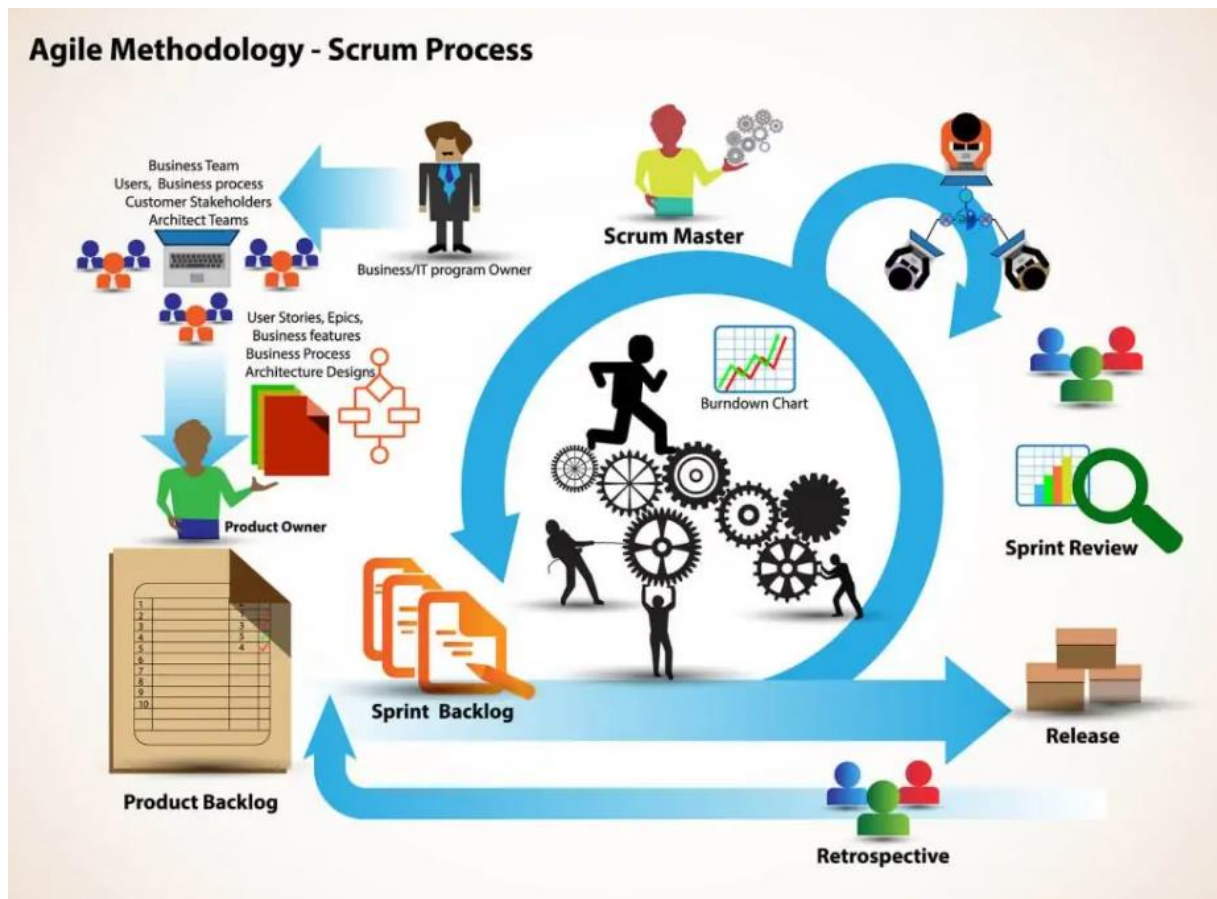
Gracias a este manifiesto, se han obtenido varios resultados prácticos. Por ejemplo, en lugar de desarrollar sistemas de software en una secuencia que va de una fase a la siguiente (que es como el método en cascada garantiza la calidad de un producto), el método ágil promueve que los procesos de desarrollo y prueba sean simultáneos y constantes. Dicho de otra forma, en el desarrollo en cascada, una fase debe finalizarse por completo antes de poder pasar a la siguiente; el desarrollo ágil, por otro lado, permite que varias secuencias sucedan al mismo tiempo.

Dentro de las metodologías ágiles, la que se usa más actualmente es la metodología **SCRUM**.

Esta metodología usada para desarrollar software está basada en un proceso iterativo e incremental, que puede adaptarse, es rápido, flexible y eficaz, diseñado para ofrecer valor al cliente mientras dura el proceso de desarrollo del software.

Cómo objetivo principal SCRUM se plantea satisfacer las necesidades del cliente mientras existe un entorno de comunicación, responsabilidad colectiva y progreso, transparente para el cliente.

El proceso parte de una idea original que hay que ir construyendo, elaborando una lista de características que hay que ordenar por prioridad por la que el cliente quiere obtener el producto (producto backlog).



Esquema metodología Scrum en desarrollo de software.

Fuente: <https://www.nimblework.com/es/agile/que-es-scrum/>

Esta metodología se basa en una serie de prácticas y roles a realizar durante el proceso de desarrollo del software.

SCRUM, es una metodología flexible que premia la aplicación de los 12 principios ágiles en un contexto acordado por todos los miembros del equipo del producto.

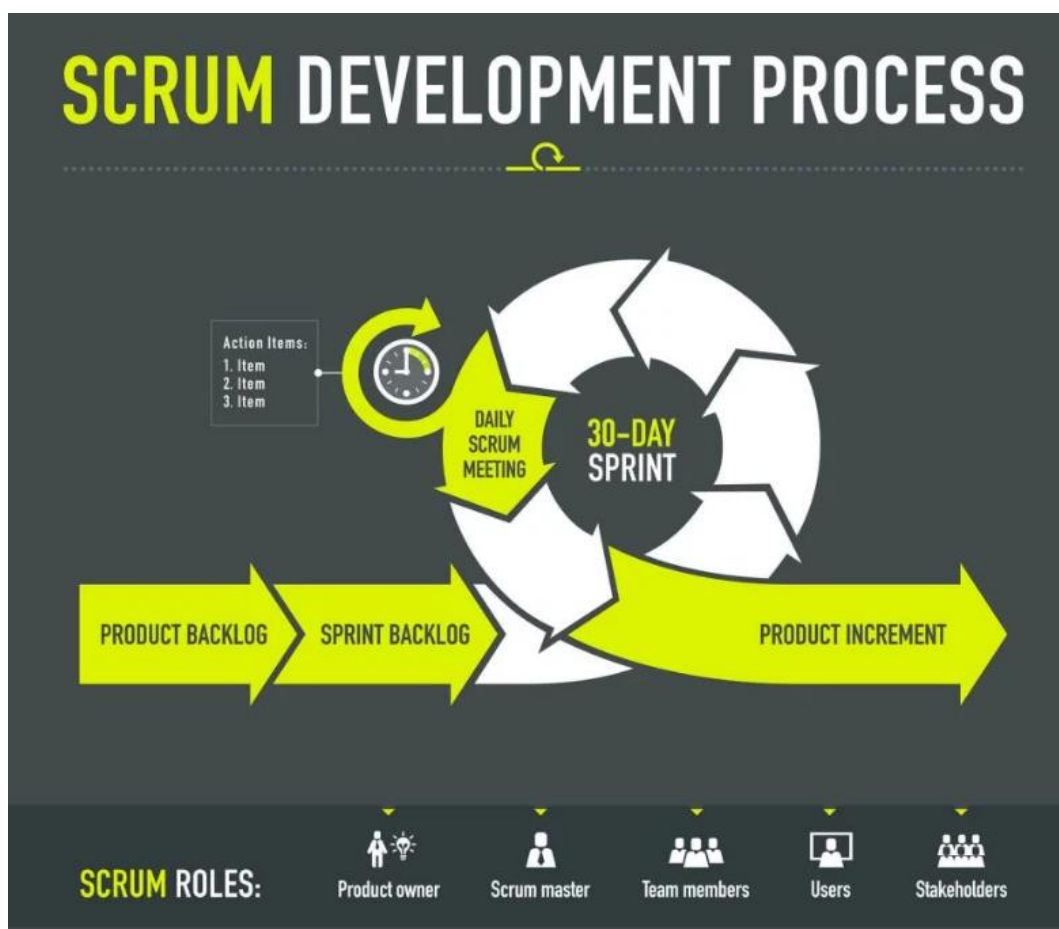
Esta metodología consiste en ejecutar bloques temporales cortos y periódicos, llamados Sprints, que suelen durar entre 2 y 4 semanas, que además es el plazo de retroalimentación y reflexión.

Cada Sprint es independiente, por lo que proporciona un resultado completo, que puede entregarse al cliente al finalizarlo.

El proceso tiene como punto de partida una lista de objetivos/requisitos que conforman el plan del proyecto. Es el cliente del proyecto el que prioriza estos objetivos considerando un equilibrio entre el valor y el coste de los mismos, así se determinan las iteraciones y las consiguientes entregas.

Por un lado, el mercado exige calidad, rapidez en la entrega y menores costes, para lo cual una empresa debe ser muy ágil y flexible en el desarrollo de los productos, para conseguir ciclos de desarrollo cortos que puedan satisfacer la demanda de los clientes sin mermar la calidad del resultado. Es una metodología muy fácil de implementar y muy popular por los rápidos resultados que obtiene.

La metodología Scrum se utiliza principalmente para el desarrollo de software, pero otros sectores también están aprovechando sus ventajas implementando esta metodología en sus modelos organizativos como los equipos de ventas, equipos de marketing y recursos humanos, etc.



Proceso de desarrollo con metodología Scrum.

Fuente: <https://www.nimblework.com/es/agile/que-es-scrum/>

Roles en SCRUM

La metodología SCRUM aplicada a la construcción de software, se centra en tener claro que es lo que el cliente quiere, que es lo que NO quiere, y en el orden que lo quiere. De esta forma se puede planificar por Sprints todo el proceso de creación del software para poder ir entregando al cliente partes del mismo. Para ello esta metodología divide al equipo en los siguientes roles:

- **Scrum master:** Esta persona dirige el proyecto, hace que el equipo cumpla las reglas, y siga el proceso que dicta la metodología. También se encarga de los problemas que puedan surgir dentro del proyecto, gestionándolos para que no sean un impedimento a la hora de concluir el mismo. Está en constante comunicación con el Product owner. Otra tarea del Scrum Master es la de mantener al equipo al día en lo que se refiere a SCRUM, proporcionando coaching, mentoring y formación, lo que hace que guíe a los componentes del equipo hacia una mejora continua.
- **Product owner (PO):** Es la persona más cercana a los clientes, hace ver al resto del equipo lo que el cliente quiere, suele ser el responsable del proyecto y se centra en la parte de negocio. Traslada al Product Backlog las tareas y las prioriza.
- **Team Members (Equipo):** Resto de personas del equipo con conocimientos técnicos necesarios para llevar a cabo el proyecto.

Eventos en SCRUM

Cada uno de los eventos de Scrum facilita la adaptación de alguno de los aspectos del proceso, el producto, el progreso o las relaciones.

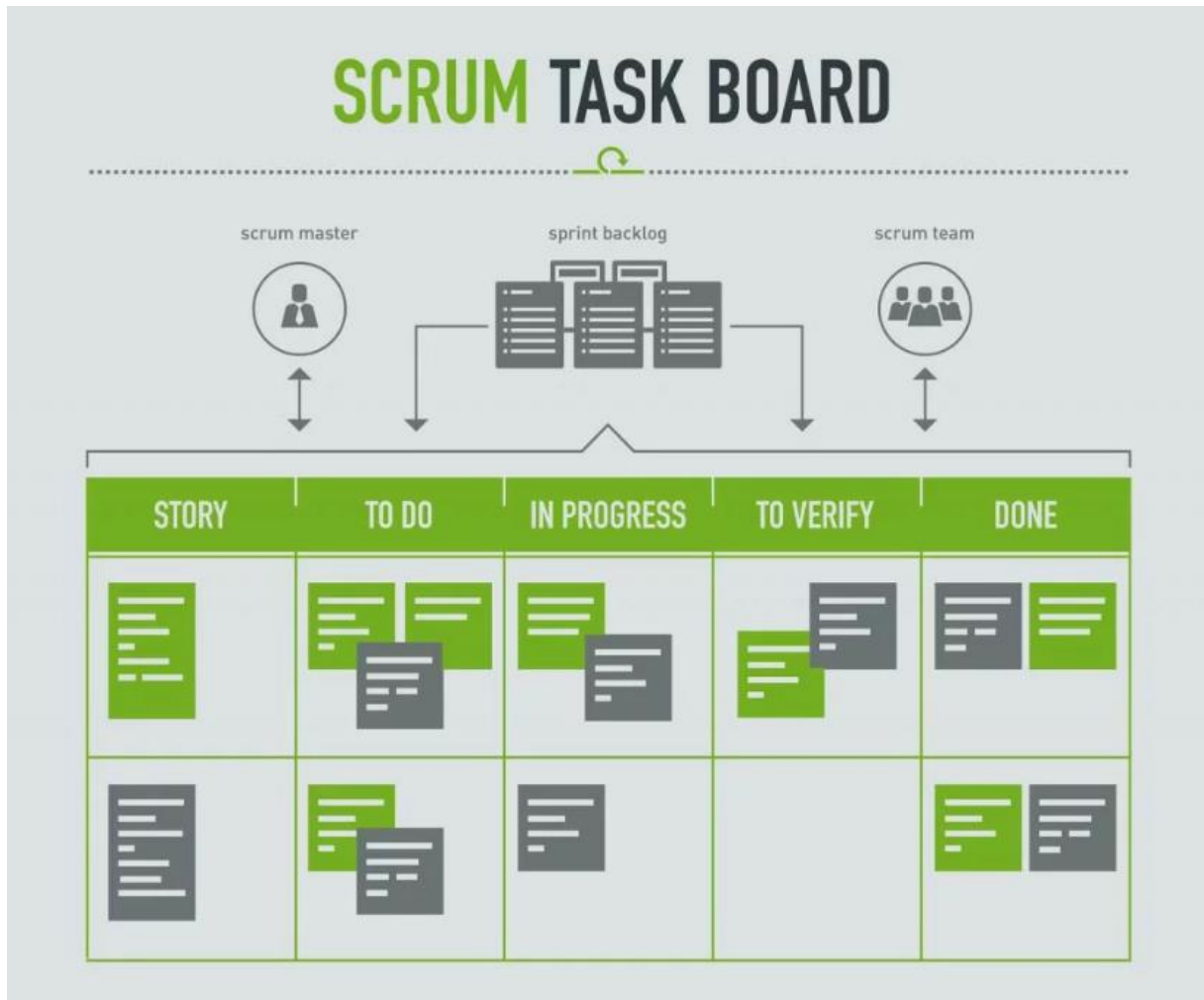
- ***Sprint:*** Es la característica principal de la metodología SCRUM, cuando se comienza un proyecto, se suele dividir el mismo en distintas partes, cada una de estas partes debe tener un fin y a estas se le conoce como sprint. Es la principal diferencia entre esta metodología y otros modelos de desarrollo ágil.
- ***Planificación del Sprint:*** Se conoce por este nombre a la parte en la que se define que y como se va a hacer en Sprint. Es una reunión que se realiza al principio de cada Sprint para definir el enfoque del proyecto a partir de las etapas y plazos del Product Backlog.

- *Scrum Diario*: Es una reunión diaria para ver el progreso y la tendencia que lleva el Sprint hasta su final generando un plan para las siguientes 24 horas. Es una reunión diaria donde se suelen hacer tres preguntas de forma individual: ¿Qué hice ayer? ¿Qué voy a hacer hoy? ¿Qué ayuda necesito? Una de las acciones que debe llevar a cabo el Scrum Master es proporcionar la ayuda necesaria.
- *Revisión del sprint*: Esta parte responde a la pregunta ¿Qué trabajo se ha completado?, se verifica con respecto al backlog del software para futuras entregas.
- *Retrospectiva del sprint*: Reunión de equipo que revisa los objetivos que se han cumplido, lo que se ha hecho bien, y lo que no se ha hecho tan bien, para que los errores cometidos no se repitan en futuros Sprints o proyectos, implementando mejoras en la forma de desarrollo.

Artefactos de Scrum

Se conoce como artefactos de Scrum aquellos elementos que garantizan la transparencia de información para la toma de decisiones:

- *Product Backlog (PB)*: Tiene una lista que tiene todos los elementos necesarios para satisfacer las necesidades del cliente en el software. Esta lista se elabora junto con el cliente y se prioriza en función de lo que es más o menos importante para el cliente. El objetivo es que el cliente responda a la pregunta «¿Qué hay que hacer?».
- *Sprint Backlog (SB)*: son una serie de elementos del producto backlog, que se seleccionan por el equipo de trabajo para realizar durante el sprint en el que se trabaja. También se establece la duración de cada Sprint. Se suelen mostrar en tableros que se llaman Tableros Scrum, de esta forma el proceso y avance es visible para todo el que se encuentra en la zona de desarrollo.
- *Incremento*: Son todas las tareas, casos de uso, historias de usuario, backlogs de producto y cualquier elemento que se haya desarrollado durante el sprint y que se pondrá a disposición del usuario final en forma de Software.



Tablero de tareas Scrum.

Fuente: <https://www.nimblework.com/es/agile/que-es-scrum/>



ENLACE DE INTERÉS

En esta web se recoge mucha información sobre la metodología SCRUM:





COMPRUEBA LO QUE SABES

¿Cuáles son las diferencias entre el Scrum Master y el Product owner? Coméntalo en el foro de la unidad



VÍDEO DE INTERÉS

En este vídeo verás una explicación de lo que es la metodología SCRUM.



RESUMEN FINAL

Durante esta unidad has estudiado distintos lenguajes de programación, clasificados según distintos criterios (niveles, paradigmas de la programación, la forma en la que son ejecutados y los lenguajes de programación más difundidos) lo que te permitirá a lo hora de desarrollar un proyecto, poder elegir el lenguaje de programación dependiendo de lo que se quiera programar. Además, se ha aprendido a distinguir entre compiladores, intérpretes y depuradores a lo hora de implementar código.

También se han podido ver las distintas etapas en el desarrollo del software (análisis, diseño, implementación, pruebas, explotación y mantenimiento), lo que se hace en cada una de ellas, y el tipo de documentación que se debe hacer dentro de un proyecto software y que, además, la documentación no es una etapa aislada dentro del ciclo de vida del software, sino que es una etapa transversal en todo el ciclo de vida.

Además, se ha hecho hincapié en conocer la metodología ágil más actual y que lleva implantada en grandes empresas desde un tiempo hasta ahora, la metodología SCRUM, la cual es usada para desarrollar software y está basada en un proceso iterativo e incremental y diseñado para ofrecer valor al cliente mientras dura el proceso de desarrollo del software.