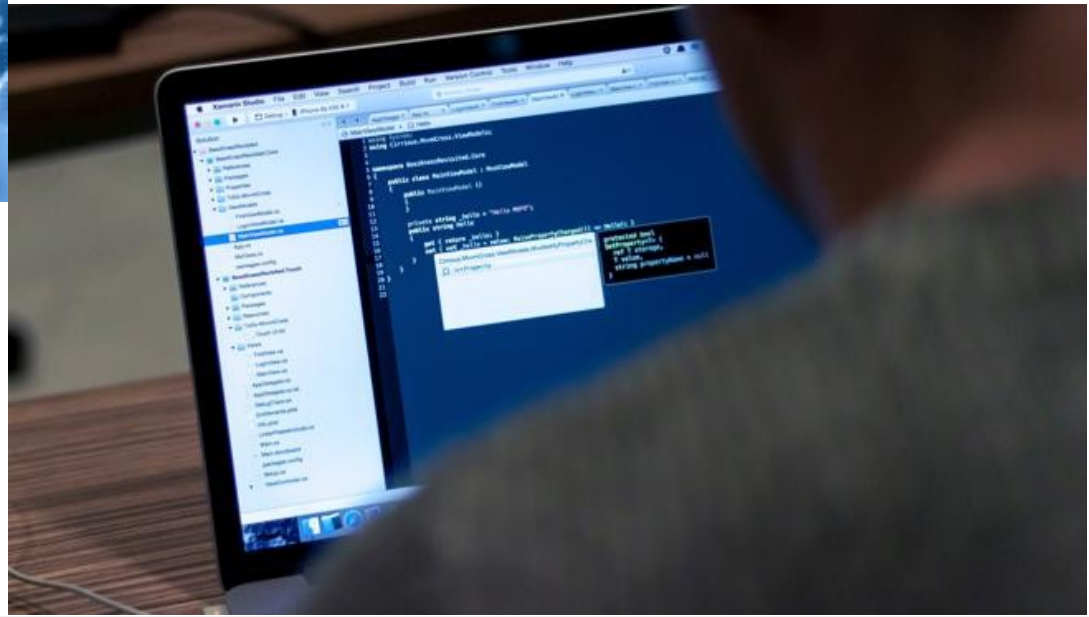
1. Introducción al desarrollo de software

Curso 2022-2023

Entornos de desarrollo

Desarrollo de Aplicaciones Web

1. Desarrollo de software



Sistema informático

- **Sistema informático:** Sistema que nos permite almacenar y procesar información mediante una serie de partes interrelacionadas, cuya finalidad es **obtener nueva información** a partir de la ya existente.

¿Qué elementos lo forman?

- **Hardware**: es el conjunto de todos los elementos físicos que compone el ordenador.

Informalmente se trata de todas las partes del ordenador que pueden ser tocadas con las manos: procesador, memoria, periféricos, placas, buses, circuitos, etc.

- **Software**: es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee. Es la parte intangible del sistema, y el encargado de comunicarse con el HW
- **Usuario**: no podemos olvidar que un sistema informático necesita de nuestra intervención

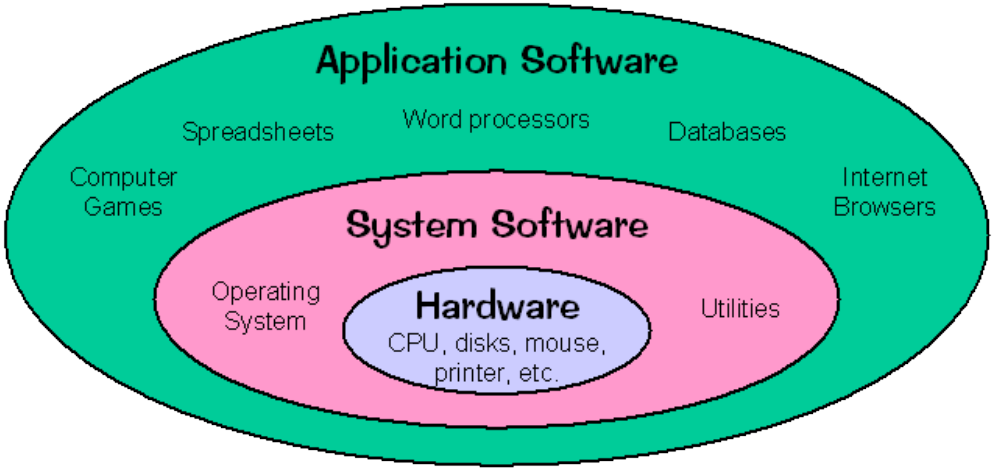
para la organización y planificación de las tareas.

¿Ejemplos?

Tipos de Software

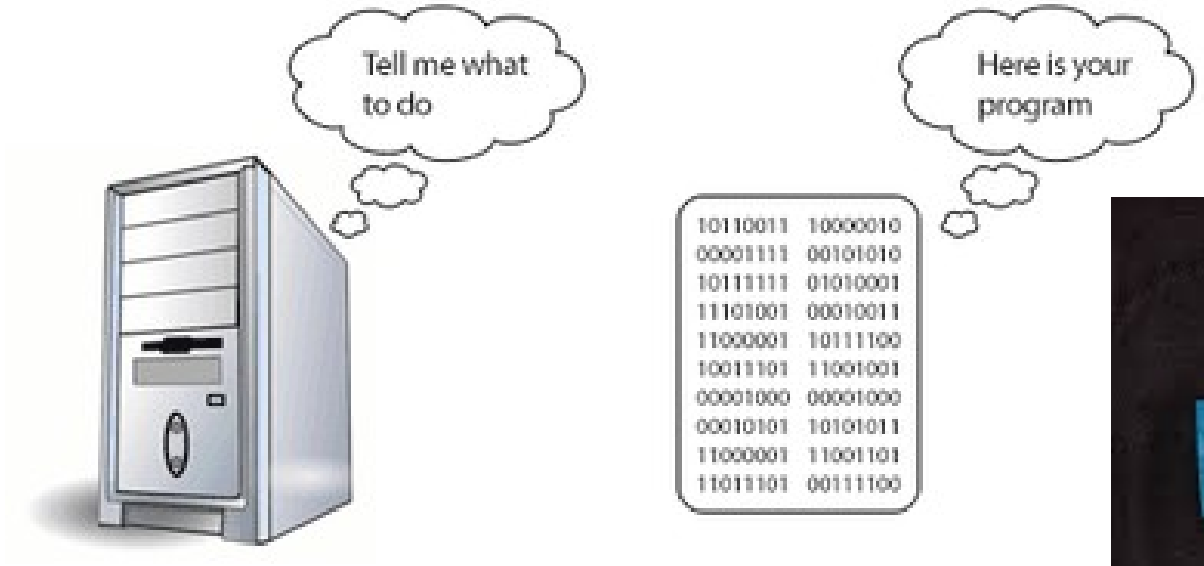
Según su ***función*** hay tres tipos de SW:

- **De sistema**: Son aquellos programas escritos para servir y gestionar otros programas. Un sistema operativo es software desarrollado para gestionar la comunicación entre el SW y HW.
- **De aplicación**: Aquellos programas que realizan tareas específicas en el sistema informático
- **De programación**: Aquellos programas cuya finalidad es ayudar a crear otros programas informáticos: compiladores, IDE (*Entornos de desarrollo*), depuradores…



Programa informático

- **Programa informático** -> conjunto de **datos** e **instrucciones** que se ejecutan de **manera secuencial** con el objetivo de realizar una o varias tareas en un sistema informático.

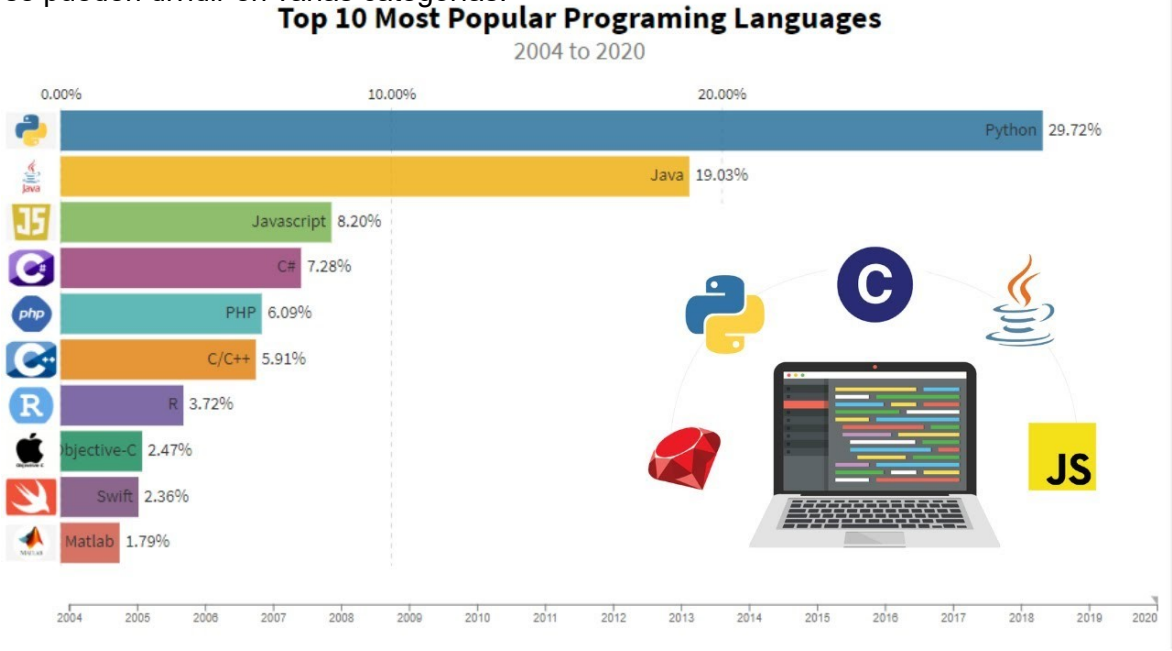


Lenguaje de programación

Conjunto de **instrucciones**, **operadores** y **reglas de sintaxis**, que se ponen a disposición del programador para diseñar y desarrollar programas informáticos

Según el aspecto de los programas se pueden dividir en varias categorías:

- Nivel de abstracción
- Según la ejecución
- Según el paradigma



Tipos de lenguajes de programación: bajo nivel vs alto nivel

Por ***abstracción*** nos referimos al nivel que se desacopla o aleja del código máquina, es decir, dichos programas escritos en binario.

Levels of Programming Languages

High-level program

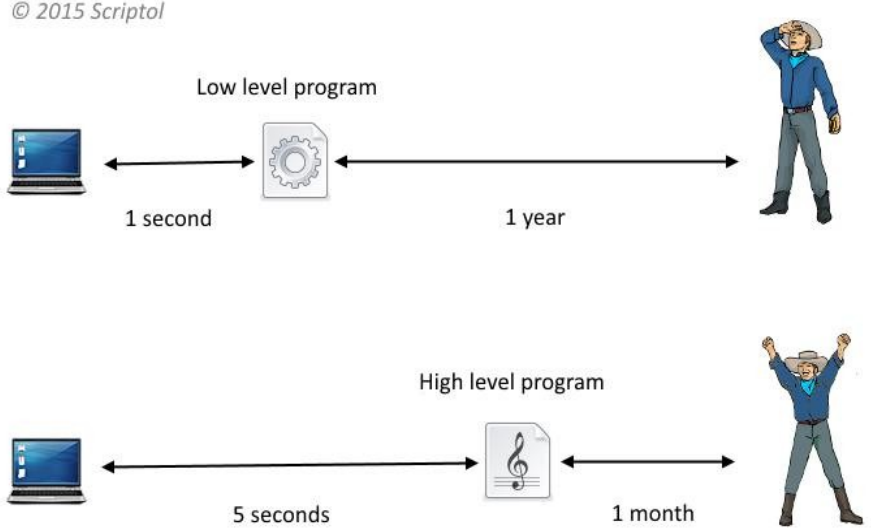
```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
0001001001000101
0010010011101100
10101101001...
```



Tipos de lenguajes de programación: bajo nivel vs alto nivel

```
#include <st
int main() {
    // printf
    printf("H
    return 0;
}
```

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 03 00 01 00 00 00  80 80 04 08 34 00 00 00  |............4...|
00000020  c8 00 00 00 00 00 00 00  34 00 20 00 02 00 28 00  |........4. ...(.|
00000030  04 00 03 00 01 00 00 00  00 00 00 00 00 80 04 08  |................|
00000040  00 80 04 08 9d 00 00 00  9d 00 00 00 05 00 00 00  |................|
00000050  00 10 00 00 01 00 00 00  a0 00 00 00 a0 90 04 08  |................|
00000060  a0 90 04 08 0e 00 00 00  0e 00 00 00 06 00 00 00  |................|
00000070  00 10 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  ba 0e 00 00 00 b9 a0 90  04 08 bb 01 00 00 00 b8  |................|
00000090  04 00 00 00 cd 80 b8 01  00 00 00 cd 80 00 00 00  |................|
000000a0  48 65 6c 6c 6f 2c 20 77  6f 72 6c 64 21 0a 00 2e  |Hello, world!...|
000000b0  73 68 73 74 72 74 61 62  00 2e 74 65 78 74 00 2e  |shstrtab..text..|
000000c0  64 61 74 61 00 00 00 00  00 00 00 00 00 00 00 00  |data............|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000000f0  0b 00 00 00 01 00 00 00  06 00 00 00 80 80 04 08  |................|
00000100  80 00 00 00 1d 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  10 00 00 00 00 00 00 00  11 00 00 00 01 00 00 00  |................|
00000120  03 00 00 00 a0 90 04 08  a0 00 00 00 0e 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  04 00 00 00 00 00 00 00  |................|
00000140  01 00 00 00 03 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  ae 00 00 00 17 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  01 00 00 00 00 00 00 00                           |........|
```

Lenguaje binario

TODA la información que se almacena en un ordenador (programas, videos, música, videojuegos…) se almacena usando el ***sistema de numeración binario.***

Este sistema es posicional, donde el valor de un dígito depende tanto del símbolo utilizado, como de la posición que ese símbolo ocupa en el número***. El número de símbolos permitidos*** en un sistema de numeración posicional se conoce como ***base*** del sistema de numeración.

¿Cuál es la base en el sistema decimal (el que usamos a diario)?

¿Cuál es la base en el sistema octal?

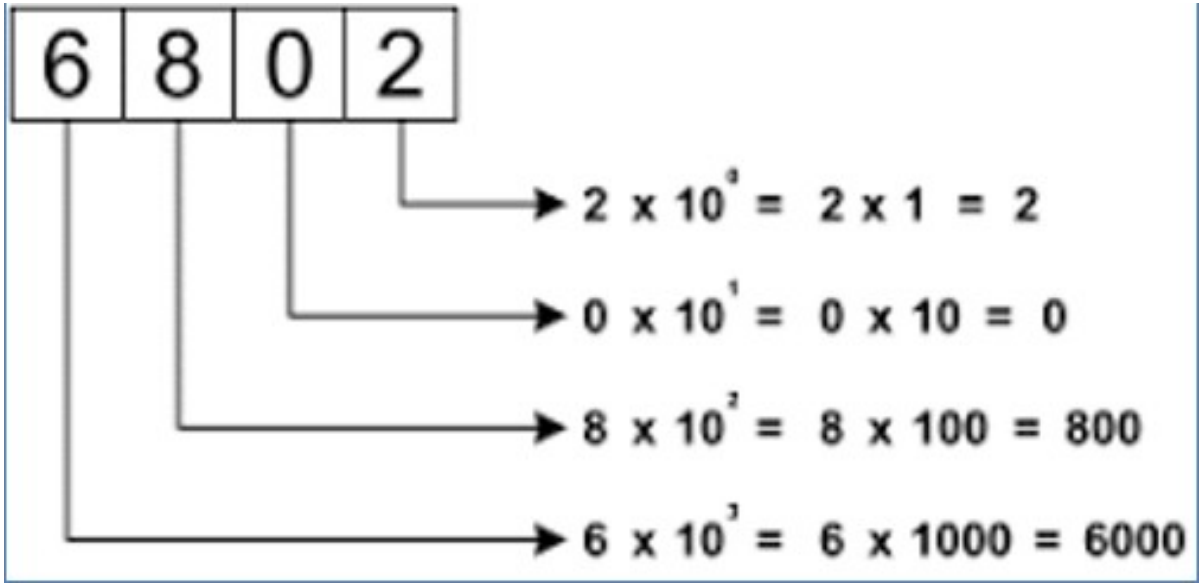
¿Y en el hexadecimal?

¿Y en el binario?

¿Posicional?

Para calcular el valor de un número, tenemos que saber la base en la que se encuentra.

Tras ello, lo que haremos es indicar las posiciones del número (empezando por el de la derecha, con un 0) y multiplicar, en cada posición, el dígito que tenemos, por la base elevada a la posición.

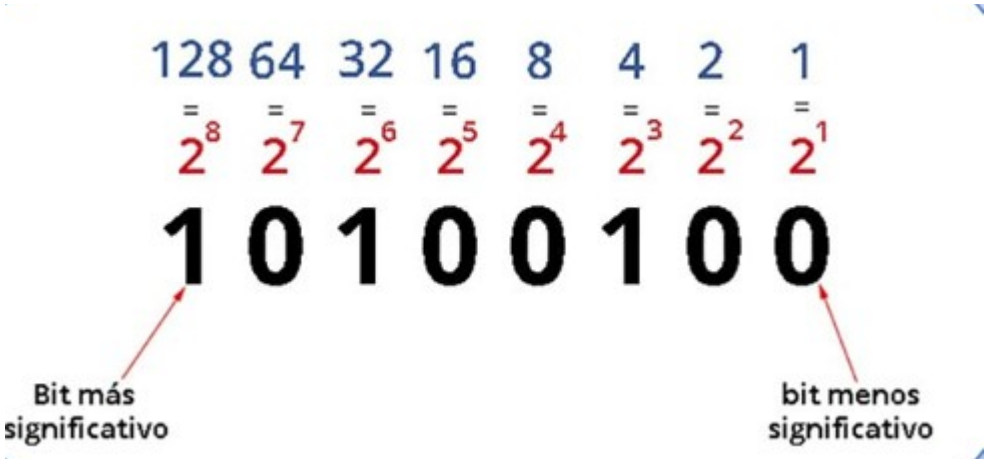


Lenguaje binario

Cada uno de los dígitos que componen un número binario se le denomina bit

Al bit situado más a la derecha en el número se le conoce como bit menos significativo (LSB)

Y el bit que está situado más a la izquierda, recibe el nombre de bit más significativo (MSB)



Lenguaje binario

Para la almacenar la información dentro de un sistema digital, se utiliza el sistema binario.

La unidad mínima de información utilizada es el ***bit***, el cual podrá tomar un valor de 0 o de 1.

Si hablamos de un ***byte***, estamos hablando de 8 bits, y es lo que suele ocupar un carácter. Un número entero, por ejemplo, utilizaría 2 bytes.

La siguiente unidad es el ***kilobyte***, igual a 1024 Bytes. Esta unidad de medida se usa para expresar la cantidades pequeñas de información, como un fichero de texto plano pequeño.

El ***megabyte*** son 1024 KB. Suele ocupar lo que un fichero mp3.

El ***gigabyte*** son 1024 MB. Es la unidad a la que estamos acostumbrados para almacenar grandes cantidades de información: HDD, videoconsolas, Smartphones…

El ***terabyte*** son 1024 GB. Poco a poco nos vamos acercando a este tipo de almacenamientos, sobre todo al hablar de HDD o alguna videoconsola.

Nombre	Abrev.	Factor binario
bytes	B	$2^0 = 1$
kilo	k	$2^{10} = 1024$
mega	M	$2^{20} = 1\,048\,576$
giga	G	$2^{30} = 1\,073\,741\,824$
tera	T	$2^{40} = 1\,099\,511\,627\,776$
peta	P	$2^{50} = 1\,125\,899\,906\,842\,624$
exa	E	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
zetta	Z	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$
yotta	Y	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$

Conversión de binario a decimal

PASO 1 – Numeramos los bits de derecha a izquierda comenzando desde el 0.

PASO 2 – A cada dígito le hacemos corresponder una potencia de base N (dependiendo de la base) y exponente igual a la posición.

PASO 3 – Por último se suman todas las potencias.

7	6	5	4	3	2	1	0	exponentes
1	1	0	0	1	0	1	1	

$$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 128 + 64 + 8 + 2 + 1 = 203$$

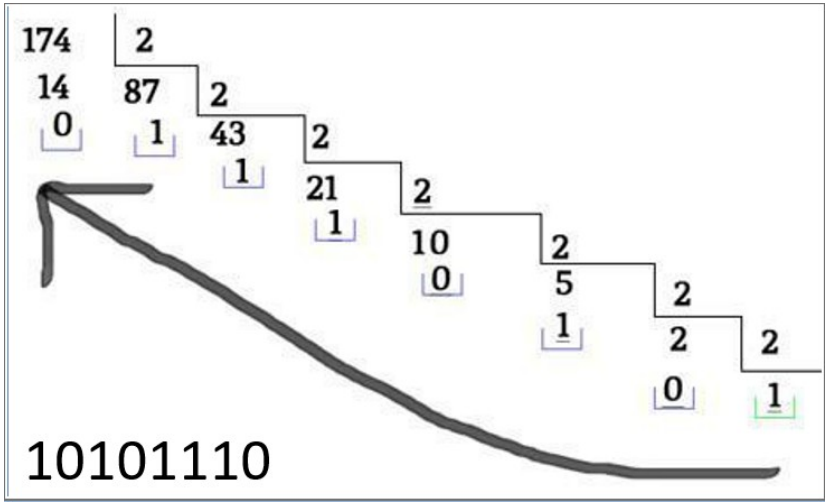
Conversión de decimal a binario

Para ello dividimos(hasta que no se admitan más divisiones enteras) por la base del sistema al que queremos pasar el valor.

Para pasar del decimal al binario, dividimos consecutivamente por 2, obteniendo como restos 0 y 1.

Para pasar del decimal al octal, dividimos consecutivamente por 8, obteniendo como restos símbolos menores que 8 (0..7)

Para pasar del decimal al hexadecimal, dividimos consecutivamente por 16, obteniendo como restos números menores que 16 (0..15) Hay que sustituir si en el resto nos sale un valor superior a 9 (10=A, 11=B..., 15=F)



Vamos a practicar!

1034(10 -> ?(16
66(10 -> ?(2
1896(10 -> ?(2
193(10 -> ?(8
146(10 -> ?(5
3500(10 -> ?(16
18541(10 -> ?(16
234(10 -> ?(8
785(10 -> ?(8
1996(10 -> ?(2
1993(10 -> ?(8
2198504(10 -> ?(16
7632(10 -> ?(2
557(10 -> ?(8

Vamos a practicar!

1000(2
110011(2
10010(2
11111(2
1010101(2
4563(8
3200(8
412(8
2124(8
557(8
AAA(16
A3(16
ABCD(16
39B2C(16

Tipos de lenguajes de programación: bajo nivel vs alto nivel

A lo largo de la historia se han ido desarrollando diferentes generaciones de lenguajes de programación

- ***1 Generación*** -> Código máquina
- ***2 Generación*** -> Lenguaje ensamblador
- ***3 Generación*** -> Son lenguajes más cercanos al lenguaje humano en los cuales la solución al problema se describe paso a paso mediante un conjunto de instrucciones más flexibles y potentes. **C, C++, C#, Java,**
- ***4 Generación*** -> Una mejora de los lenguajes de 4 generación, operan con un conjunto mayor de datos, relacionados con el tratamiento de BBDD, interfaces, web… **SQL, PL/SQL, Unix Shell**…
- ***5 Generación*** -> Son lenguajes cuyos programas están constituidos por hechos, reglas, restricciones, ecuaciones, transformaciones, u otras propiedades que debe cumplir la solución al problema.

Tipos de leng

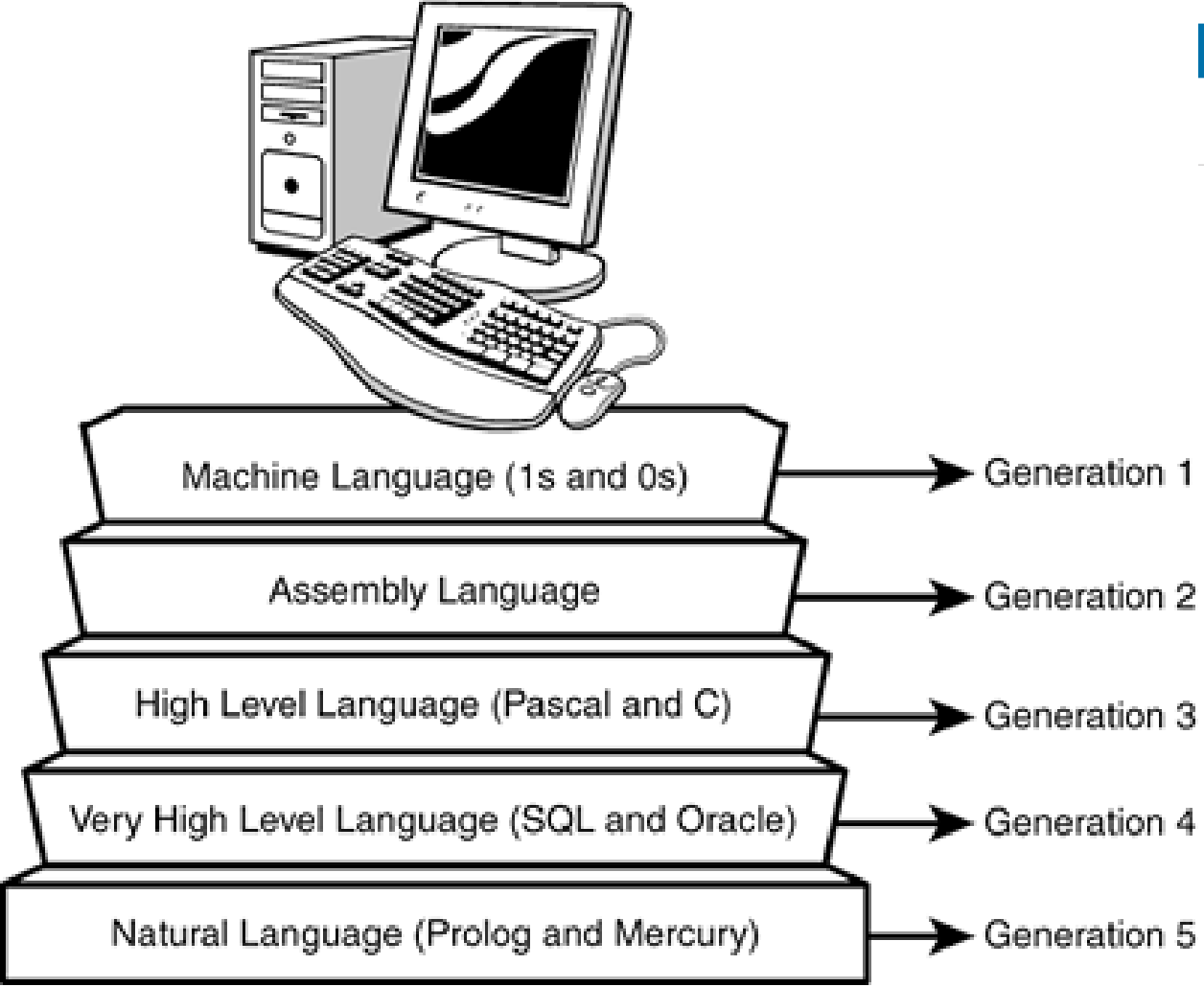
```
1.  DECLAR
2.  text  VA
3.  BEGIN
4.  text:= 'He
5.  dbms_out
6.  END;
7.  /
```

Output:

Hello

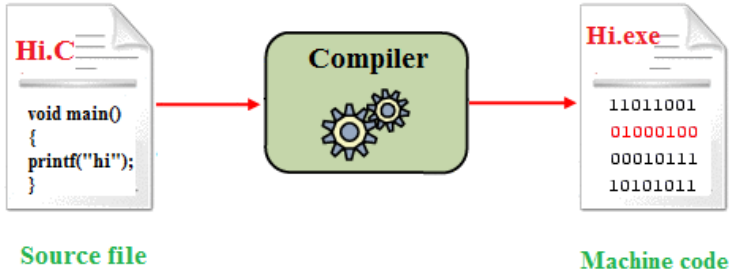
Prolog.pro

```
1  helloWorld :-
2    write('Hello World').
3
4  :- helloWorld.
5
```


Machine Language (1s and 0s)
Generation 1
Assembly Language
Generation 2
High Level Language (Pascal and C)
Generation 3
Very High Level Language (SQL and Oracle)
Generation 4
Natural Language (Prolog and Mercury)
Generation 5

Tipos de lenguajes de programación: ejecución

- Lenguaje **compilado:** Antes de poder utilizarse el programa debe utilizarse un traductor llamado "**compilador**" que se encarga de traducir ("compilar") el programa original ("código fuente") en código objeto y otro programa ("**enlazador**" o "linkador") que unirá el código objeto del programa con el código objeto de las librerías necesarias producir el programa ejecutable. Los programas ejecutables están en **binario** y son los **únicos necesarios** para el funcionamiento del programa. C o C++



Tipos de lenguajes de programación: ejecución

- Lenguaje **interpretados:** Ejecutan las instrucciones directamente, sin que se genere código objeto, para ello es necesario un **programa intérprete** en el sistema operativo o en la propia máquina donde cada instrucción es interpretada y ejecutada de manera independiente y secuencial. Así, cada vez que se usa el programa debe utilizarse un traductor llamado "**intérprete**" que se encarga de traducir ("interpretar"), sin generar código objeto, las instrucciones del programa original ("código fuente") a código máquina según van siendo utilizadas. Para el funcionamiento del programa siempre es necesario disponer del código original y del intérprete. La principal ventaja con respecto al lenguaje compilador es que ser traducen a tiempo real solo las instrucciones que se utilicen en cada ejecución, en vez de interpretar todo el código se vaya a utilizar o no. Python o JS.

Tipos de lengua

- Lenguaje **Virt**

compilados, perc

compilador, sino

por cualquier arc

el código byteco

Tipos de lenguajes de programación: ejecución

- Los lenguajes interpretados se **compilan una vez** y se utilizan cuantas veces se desee sin necesidad de volver a utilizar el compilador. Los lenguajes interpretados **son interpretados, cada vez que se ejecutan y necesitan siempre del intérprete**.
- **Los compiladores** analizan todo el programa y **no generan resultados si no es correcto todo el código**. Los intérpretes analizan las instrucciones según las necesitan y pueden iniciar la ejecución de un programa con errores e incluso terminar correctamente una ejecución de un programa con errores siempre que no haya sido necesario el uso de las instrucciones que contienen dichos errores.
- Un compilador traduce cada instrucción una sola vez. Un intérprete debe traducir una instrucción cada vez que la encuentra.
- **Los binarios son compilados para una arquitectura específica** y no pueden ser utilizados en otras arquitecturas no compatibles (deben existir distintos compiladores para generar binarios para diferentes arquitecturas y Sistemas Operativos). **Un lenguaje interpretado puede ser utilizado en cualquier arquitectura que disponga de un intérprete sin necesidad de cambios**.
- **Los lenguajes compilados son más eficientes que los interpretados y además permiten distribuir el programa en forma confidencial mediante binarios**. Siempre se pueden decompilar. La confidecialidad solo se logra mediante el cifrado criptográfico del código.

Tipos de lenguajes de programación: paradigma

Un paradigma de programación representa un enfoque particular o **filosofía** para la construcción del software, un estilo de programación que facilita la tarea de programación o añade mayor funcionalidad al programa dependiendo del problema que haya que abordar.

Los grandes grupos son:

- **Imperativo**: Las instrucciones y operaciones se diseñan y ejecutan **de forma secuencial**, uno detrás de otro.
- **Declarativo**: Un paradigma específico en el que no se detalla "**cómo**" se hace algo, si no "**qué**" hay que hacer.

Ejemplo -> Tiempo muerto finales de la NBA 1998.

Imperativo -> Quiero que saquéis de banda, hagáis la jugada XY, paséis el balón a Kerr…Michael acabe con un tiro de media distancia.

Declarativo -> Balón a Michael.

Diferencia entre ambos en 4 minutos(en inglés): Link

Tipos de lenguajes de programación: paradigma Imperativo

- **Lenguajes de programación estructurados**: son lenguajes de programación que se basan en la programación estructurada, la cual se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

1) Sentencias secuenciales

2) Sentencias selectivas (condicionales)

3) Sentencias repetitivas (iteraciones o bucles)

Ventajas:

- Los programas son fáciles de leer, sencillos y rápidos.
- El mantenimiento de los programas es sencillo.
- La estructura del programa es sencilla y clara.

Inconvenientes:

- Todo el programa se concentrar en un único bloque.
- No permite reutilización eficaz del código.

Ejemplos: Pascal, C, Fortran


Paradigmas de Programación
Imperativos
Estructurado
Dividido en bloques
Secuencia, selección e iteración
Orientado a Objetos
Abstracción de datos
Encapsulamiento de datos y comportamientos en objetos
Paso de mensajes entre objetos
Declarativos
Funcional
Las funciones son elementos de primer orden
Uso del cálculo lambda
Lógico
Los programas están construidos de hechos, predicados y relaciones

¿Cómo se obtiene el código ejecutable?

Tenemos nuestro código desarrollado en cualquier programa de cualquier paradigma, *¿cómo lo ejecutamos*?

Hay varias fases por las pasará nuestro código:

- **Código fuente**: Es el código que hemos desarrollado en cualquier lenguaje de cualquier paradigma en cualquier máquina, y no se puede ejecutar todavía. Contiene toda la lógica y estructura utilizada, pero esto se tiene que traducir (compiladores, intérpretes…)
- **Código objeto**: Es el resultado de traducir el código fuente por el componente que se encarga de la traducción asociado al lenguaje de programación. Es código máquina, y para obtener un código ejecutable hay que *linkear* (enlazar) todo el código objeto de un programa.
- **Código ejecutable:** El código ejecutable es el resultado de enlazar todo el código objeto con otros elementos, como librerías con otro tipo de código, y es, por lo general un archivo de código binario (0 y 1).Otras veces es código que hace llamadas a otro sistemas o funciones, y no es código máquina, esto es lo que llamamos un **script**.

Pregunta, *¿Todo el código ejecutable funcionará en todas las máquinas?*

Muchas veces los programas utilizan funciones específicas de un procesador o de un sistema operativo, por lo que puede no ir en todas las máquinas. **Portable** vs **no portable**

¿Un programa Java es portable?¿Y mi código hecho en C para un procesador AMD?

Compilación

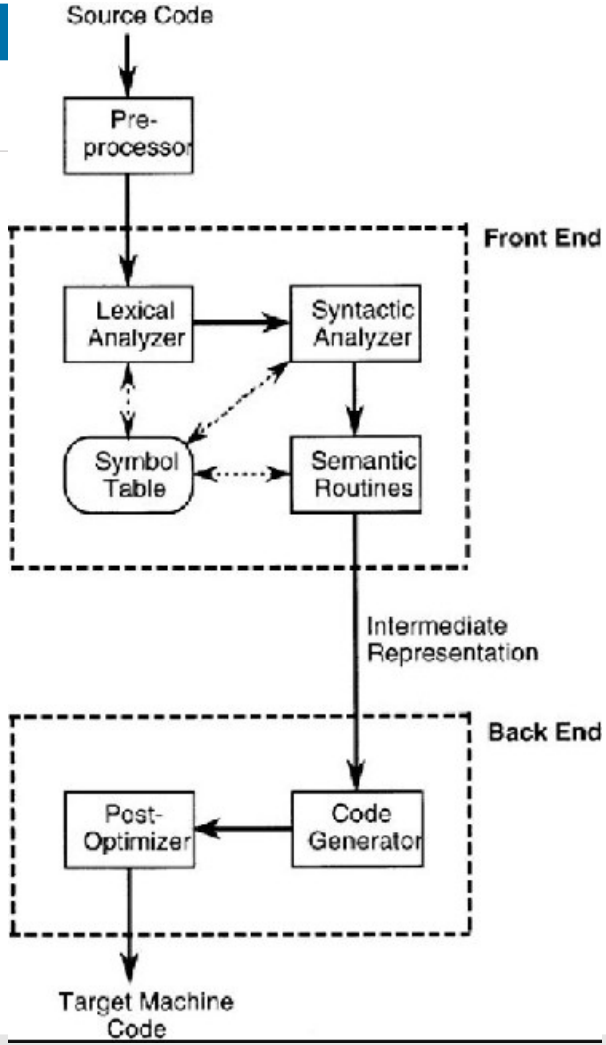
La compilación es el proceso principal mediante el cual obtenemos código ejecutable partiendo de un código fuente. Los programas que se encargan de dicha transformación los llamamos **compiladores.**

Este vídeo (en inglés) es una introducción teórica a los compiladores

Compilación

Fases de un compilador:

1. **Análisis lexicográfico**: se leen de manera secuencial todos los caracteres del código fuente, buscando palabras reservadas, caracteres de puntuación y agrupándolos todos en cadenas de caracteres que se denominan **lexemas**. El compilador crea una **tabla de símbolos**, que es una estructura de datos usada para asociar a cada variable, función y procedimiento que aparece en el programa fuente una representación de sus atributos.
2. **Análisis sintáctico-semántico**: El analizador semántico realiza una comprobación de reglas semánticas dentro del árbol sintáctico, y detecta ***incoherencias*** *en el programa*. Utiliza la tabla de símbolos para implementar las restricciones típicas de los lenguajes de programación, como son: el control de variables no declaradas, código no accesible…
3. **Generación de código intermedio**: una vez finalizado el análisis, se genera una representación intermedia a modo de pseudocódigo con el objetivo de facilitar la tarea de traducir al código objeto.

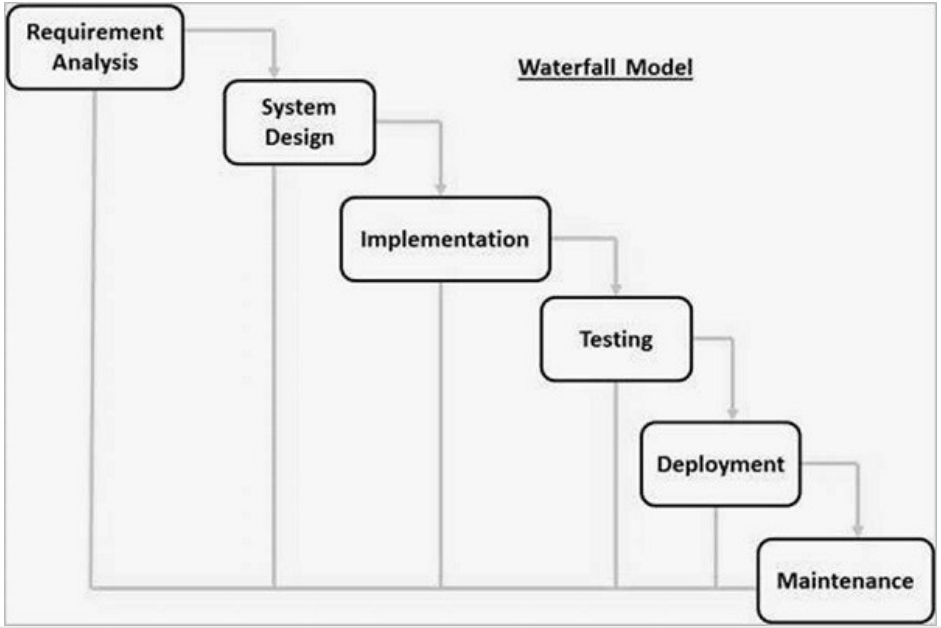


Compilación

4. **Optimización de código**: revisa el código pseudocódigo generado en el paso anterior optimizándolo para que el código resultante sea más fácil y rápido de interpretar por la máquina.
5**. Generación de código**: genera el código objeto de nuestro programa en un código de lenguaje máquina relocalizable, con diversas posiciones de memoria sin establecer, ya que no sabemos en qué parte de la memoria volátil se va a ejecutar nuestro programa.
6. **Enlazador de librerías**: se enlaza el código objeto con las librerías necesarias, produciendo en último término el código ejecutable.

Proceso de desarrollo del Software

El desarrollo de un software o de un conjunto de aplicaciones pasa por diferentes etapas desde que se produce la necesidad de crear un software hasta que se finaliza y está listo para ser usado por un usuario. Ese conjunto de etapas en el desarrollo del software responde al concepto de **ciclo de vida del programa**, siendo uno de los modelos más clásicos el **modelo en cascada**



Proceso de desarrollo del Software

1. **Análisis**. La fase de análisis define los **requisitos del software que hay que desarrollar**. Inicialmente, esta etapa comienza con una entrevista al cliente, que establecerá lo que quiere o lo que cree que necesita, lo cual nos dará una buena idea global de lo que necesita, pero no necesariamente del todo acertada. Aunque el cliente crea que sabe lo que el software tiene que hacer, es necesaria una buena habilidad y experiencia para reconocer requisitos incompletos, ambiguos, contradictorios o incluso necesarios. Es importante que en esta etapa del proceso de desarrollo se mantenga una comunicación bilateral, aunque es frecuente encontrarse con que el cliente pretenda que dicha comunicación sea unilateral, es necesario un contraste y un consenso por ambas partes para llegar a definir los requisitos verdaderos del software.
2. **Diseño**: En esta etapa se pretende **determinar el funcionamiento de una forma global y general**, sin entrar en detalles. Uno de los objetivos principales es establecer las consideraciones de los recursos del sistema, tanto físicos como lógicos. Se define por tanto el entorno que requerirá el sistema, aunque también se puede establecer en sentido contrario, es decir, diseñar el sistema en función de los recursos de los que se dispone.
3. **Codificación:** Gracias a las etapas anteriores, el programador contará con un análisis completo del sistema que hay que codificar y con una especificación de la estructura básica que se necesitará, por lo que en un principio solo habría que traducir el cuaderno de carga en el lenguaje deseado para culminar la etapa de codificación pero esto no es siempre así, las dificultades son recurrentes mientras se modifica

Proceso de desarrollo del Software

4. **Pruebas**: Con una doble funcionalidad, las pruebas buscan confirmar que la codificación ha sido exitosa y el software no contiene errores, a la vez que se comprueba que el software hace lo que debe hacer, que no necesariamente es lo mismo. No es un proceso estático, y es usual realizar pruebas después de otras etapas, como la documentación. En general, las pruebas las realiza, idílicamente, personal diferente al que codificó la aplicación, con una amplia experiencia en programación, personas capaces de saber en qué condiciones un software puede fallar de antemano sin un análisis previo.
5. **Documentación**: Diferenciamos en dos tipos: la documentación disponible para el usuario y la documentación destinada al propio equipo de desarrollo. La documentación para el usuario debe mostrar una información completa y de calidad que ilustre mediante los recursos más adecuados cómo manejar la aplicación. Una buena documentación debería permitir a un usuario cualquiera comprender el propósito y el modo de uso de la aplicación sin información previa o adicional. Por otro lado, tenemos la documentación técnica, destinada a equipos de desarrollo, que explica el funcionamiento interno del programa, haciendo especial hincapié en explicar la codificación del programa. Se pretende con ello permitir a un equipo de desarrollo cualquiera poder entender el programa y modificarlo si fuera necesario.

Proceso de desarrollo del Software

6. **Despliegue**: Una vez que tenemos nuestro software, hay que prepararlo para su distribución. Para ello se implementa el software en el sistema elegido o se prepara para que implemente por si solo de manera automática. Cabe destacar que en caso de que nuestro software sea una versión sustitutiva de un software anterior es recomendable valorar la convivencia de sendas aplicaciones durante un proceso de adaptación.
7. **Mantenimiento**: Son muy escasas las ocasiones en las que un software no va a necesitar de un mantenimiento continuado. En esta fase del desarrollo de un software se arreglan los fallos o errores que suceden cuando el programa ya ha sido implementado en un sistema y se realizan las ampliaciones necesitadas o requeridas. Cuando el mantenimiento que hay que realizar consiste en una ampliación, el modelo en cascada suele volverse cíclico, por lo que, dependiendo de la naturaleza de la ampliación, puede que sea necesario analizar los requisitos, diseñar la ampliación, codificar la ampliación, probarla, documentarla, implementarla y por supuesto, dar soporte de un mantenimiento sobre la misma, por lo que al final, este modelo es recursivo y cíclico para cada aplicación y no es un camino rígido de principio a fin.

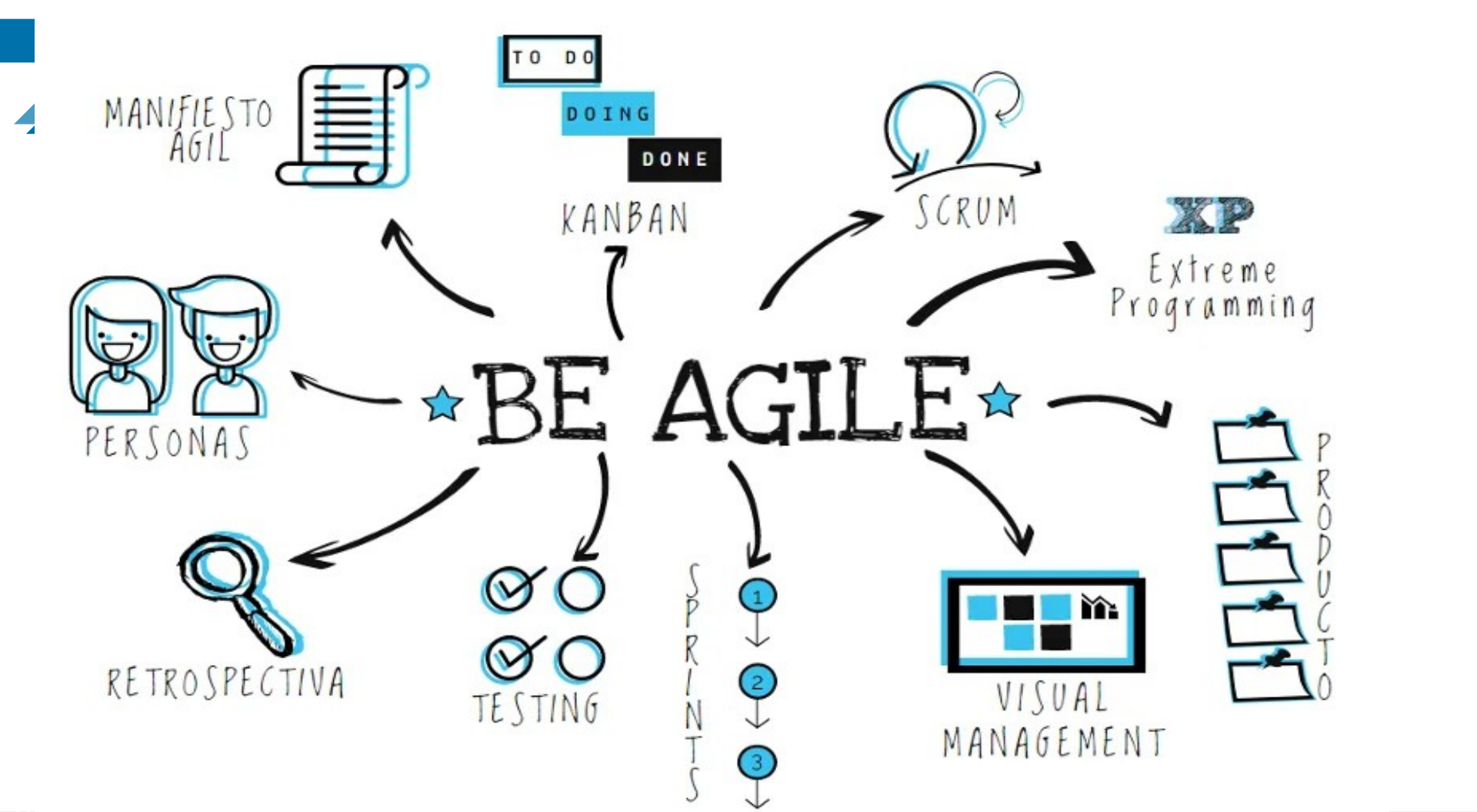
Metodologías ágiles

Como se ha podido observar, el sistema de desarrollo de software en cascada es en su propia naturaleza, inflexible y permite poca adaptación al cambio o imprevisto.

Es por ello que en 2001 se escribió el manifiesto de la metodología Ágil; La metodología Agile consiste en partir del proyecto general e ir desmenuzándolo en pequeñas tareas para poder centrarse de manera concreta en cada una de ellas.

En concreto, las metodologías ágiles de desarrollo de ***software buscan proporcionar en poco tiempo pequeñas piezas de software en funcionamiento para aumentar la satisfacción del cliente***. Estas metodologías utilizan enfoques flexibles y el trabajo en equipo para ofrecer mejoras constantes.

Por lo general, ***el desarrollo ágil de software implica que pequeños equipos autoorganizados de desarrolladores y representantes empresariales se reúnan regularmente en persona durante el ciclo de vida del desarrollo de software.***


MANIFIESTO AGIL
TO DO
DOING
DONE
KANBAN
SCRUM
XP
Extreme Programming
PERSONAS
BE AGILE
PRODUCTO
RETROSPECTIVA
TESTING
SPRINTS
1
2
3
VISUAL MANAGEMENT

Roles

El equipo de desarrollo de software se compone de un grupo de roles, cada uno con unas responsabilidades y tareas diferentes:

- Dueño del producto (*PRODUCT OWNER)*:
  - Es el miembro del equipo que actúa en nombre del cliente; es capaz de analizar y comunicar al resto del equipo las tareas necesarias para llevar a cabo los requisitos establecidos por el cliente
- Project manager:
  - Gestiona el proyecto, asigna tareas al resto del equipo, se encargar del presupuesto y la asignación de horas de trabajo. Se comunican con el cliente, y con el resto de partes de un proyecto
- Diseñadores UX y UI
  - Se encargan de las interfaces del producto y de asegurar un resultado enfocado en el usuario.
- Comercial
  - Recogen las necesidades comerciales del cliente y ofrecen proyectos software viables y eficaces
- Desarrolladores de software
  - Junior vs senior vs back-end vs front-end

Roles

- Team lead
  - Es el desarrollador a cargo del rendimiento del equipo, y que todos los desarrolladores estén trabajando de forma correcta y en conjunto. Son responsables del aprendizaje de las nuevas incorporaciones.
- Tech lead (analista):
  - Es el responsable de supervisar los requerimientos técnicos y de sistema (hardware, software…). Orientan técnicamente al resto del equipo de trabajo

Frameworks

Un framework es un ***conjunto de elementos y pautas que definen la estructura y metodología, sobre cómo desarrollar de un proyecto software***. Actúan de una guía o esquema que nos ayuda a programar de forma sencilla y rápida.

Permite desarrollar ágilmente aplicaciones mediante la aportación de librerías y/o funcionalidades ya desarrolladas.

Principalmente, nos permite centrarnos en el problema, en vez de preocuparnos por implementar funcionalidades que son de *uso común* en muchas aplicaciones. Ejemplos:

JavaScript: ***Angular***, Ember, Vue, React

PHP: Laravel, CodeIgniter, *Symfony*

Java: *Spring MVC*, JSF, Struts

Python: Django

C#: *.NET*

Máquinas virtuales

Una **máquina virtual** es un software que emula a un ordenador y puede ejecutar programas como si fuese una computadora real.

Una característica esencial de las máquinas virtuales es que los procesos que ejecutan están limitados por los recursos y abstracciones proporcionados por ellas. Una máquina virtual emplea los recursos disponibles en la máquina anfitrión o host.

Podemos ejecutar un sistema operativo que queramos probar (GNU/Linux, por ejemplo) desde nuestro sistema operativo habitual (Mac OS X por ejemplo) sin necesidad de instalarlo directamente en nuestra PC y sin miedo a que se desconfigure el sistema operativo primario.

Una máquina puede ser de **sistema** (Una máquina física representada por una máquina virtual) o de **proceso** (como la máquina virtual de Java)