



# UT 0. Nociones básicas

---

ENTORNO DE DESARROLLO

I.E.S. Luis Vives - Desarrollo de Aplicaciones Web

Curso 2022/2023

# Nociones básicas

---

INFORMÁTICA

## ➤ Nociones básicas de informática

1.1 Concepto de informática

1.2 Estructura del ordenador

1.3 Sistemas operativos

1.4 Codificación de la información

## ➤ 1.5 Software

1.6 Lenguajes de programación

1.7 Traductores

# Concepto de informática

---

CONCEPTO

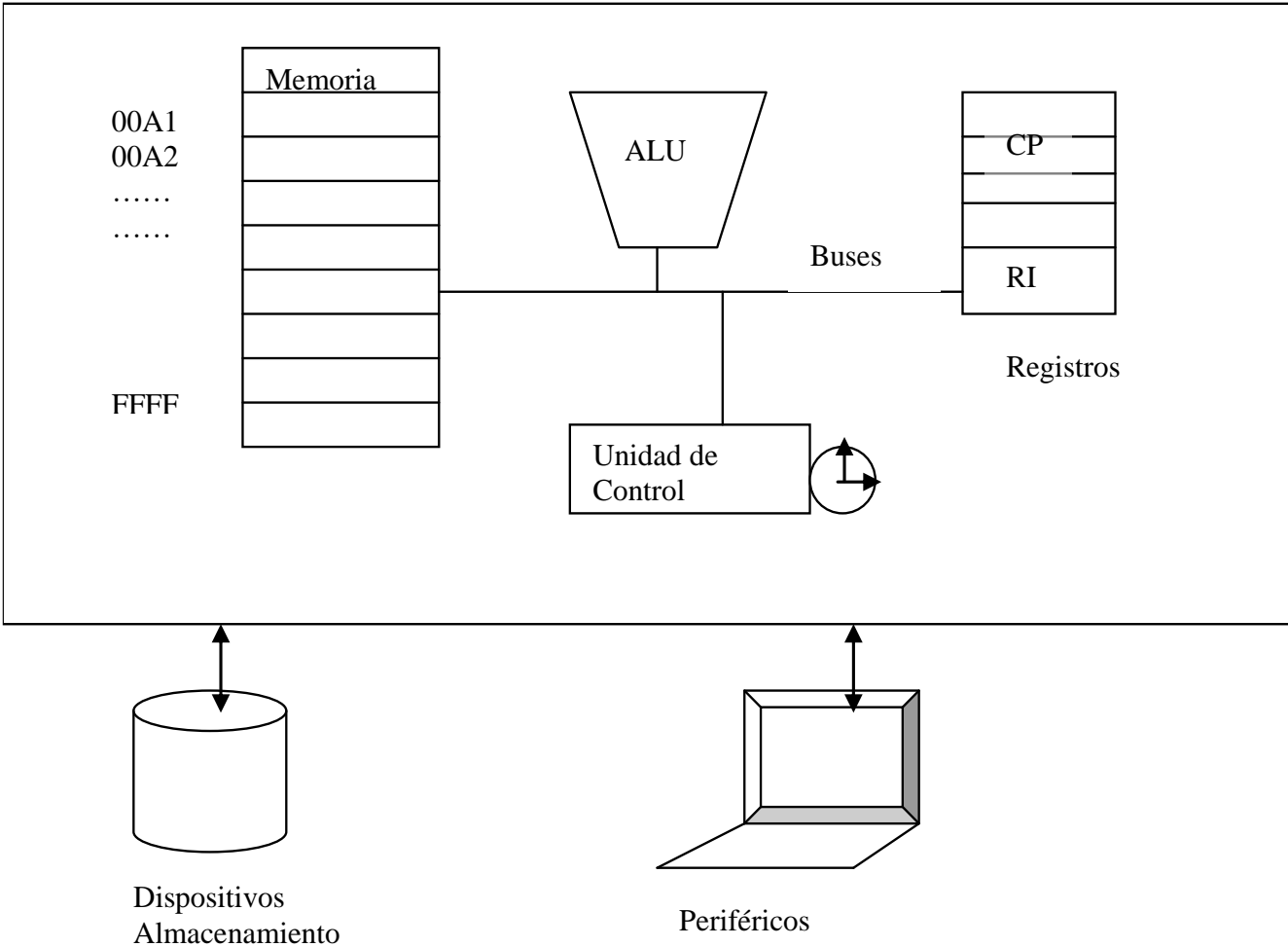
## Definiciones de **Informática**:

Ciencia del **tratamiento** racional, por **medio de máquinas automáticas**, de la **información**, considerada ésta como soporte de los conocimientos humanos y de las comunicaciones, en los campos técnico, económico y social

Conjunto de Técnicas, métodos y máquinas aplicados al **tratamiento lógico y automático de la información**.

Actividad científica dirigida a la investigación de los medios (físicos e intelectuales) que permiten el **tratamiento** y elaboración **automática** de las **informaciones** necesarias para el desarrollo de las actividades humanas.

Estructura del ordenador  
Arquitectura de Von Newman



## Tipos

### Según su naturaleza

- RAM (Random Access Memory)
- ROM (Read Only Memory)
- EPROM (Erasable Programmable Read Only Memory)
- .....

### Según su uso (De menor a mayor tiempo de acceso)

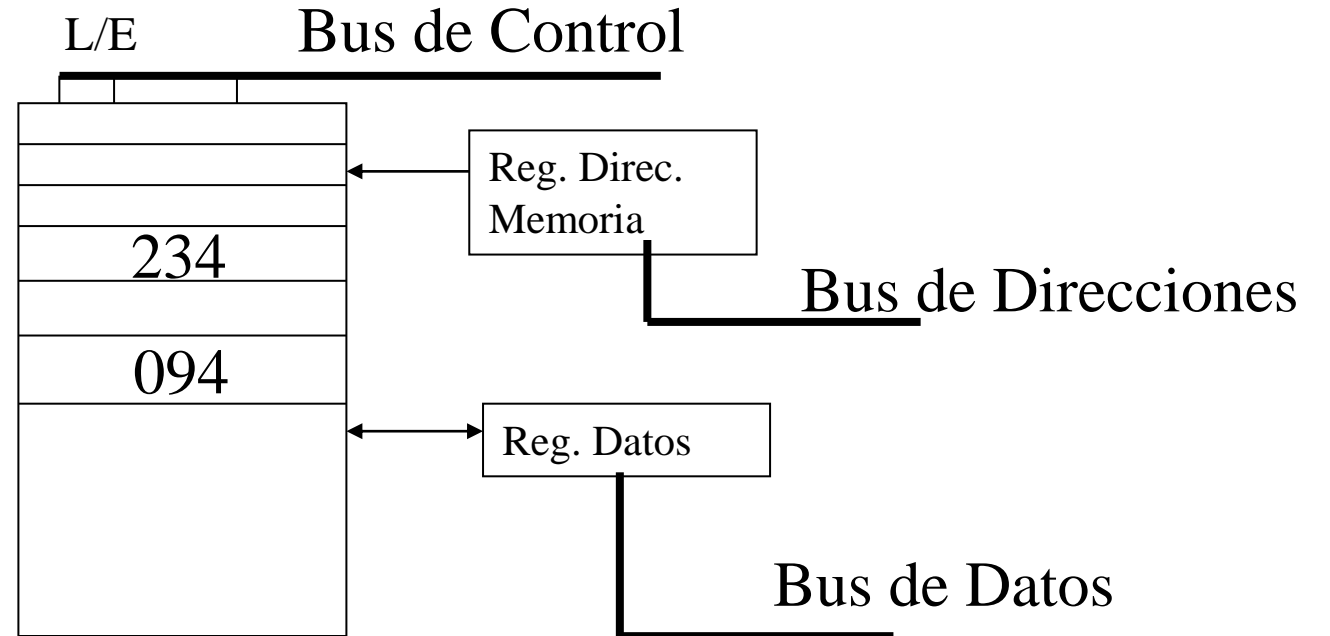
- Registros internos
- Caché interna
- Memoria principal
- Memoria Secundaria

Bus de direcciones -->

Tamaño de la memoria

Bus de Datos -->

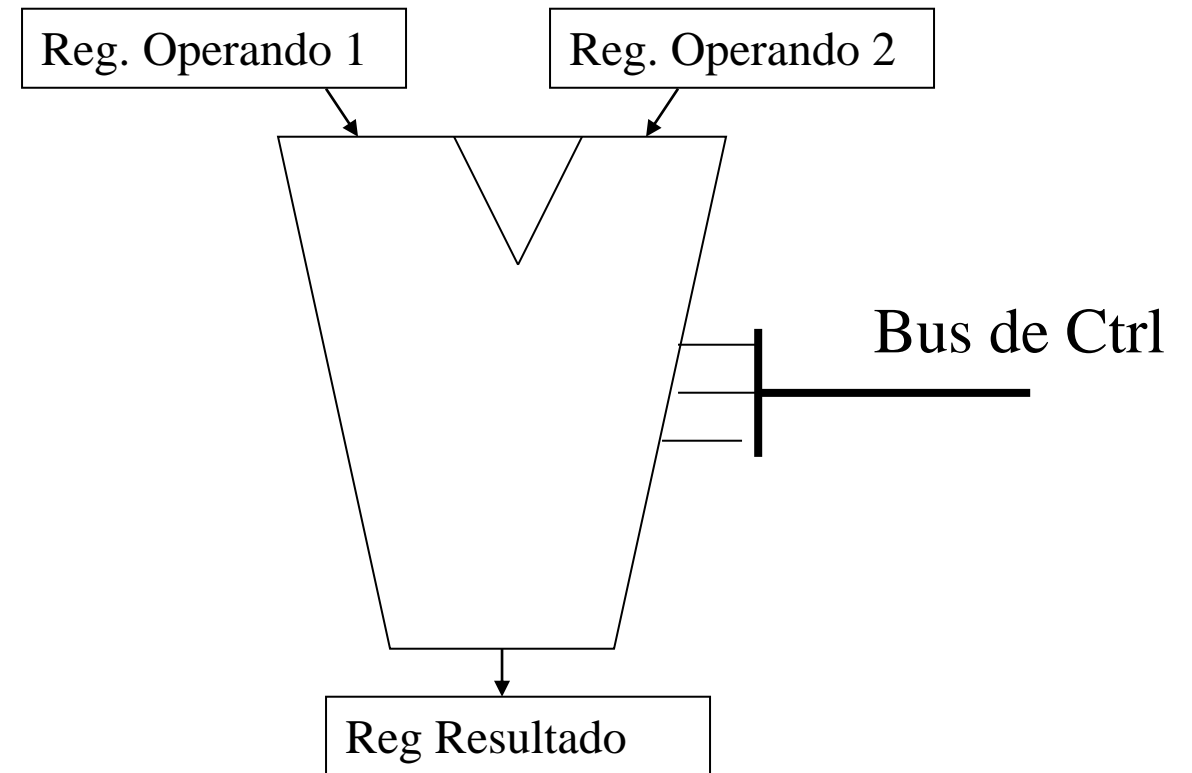
Tamaño datos





Señales de control:

- Operaciones aritméticas
  - Suma
  - Resta
  - Multiplicación
  - ...
- Operaciones lógicas
  - AND
  - OR
  - NOT



Traduce la instrucción (SUMA A, B) en las ordenes precisas para

- Mover los datos,
- Ejecutar las operaciones adecuadas en la ALU,
- Secuenciar las acciones
- ....

Fetch:

- Aumentar el CP
- Mover los datos de la MP al R. Instrucción
- Descodificar el RI

Ejecución

- Operaciones propias de la instrucción

Almacenamiento de datos (discos, CD, DVD, Cintas)

Toma de datos

- Teclado
- Scanner
- Cámaras

Salida de datos

- Impresora
- Pantalla

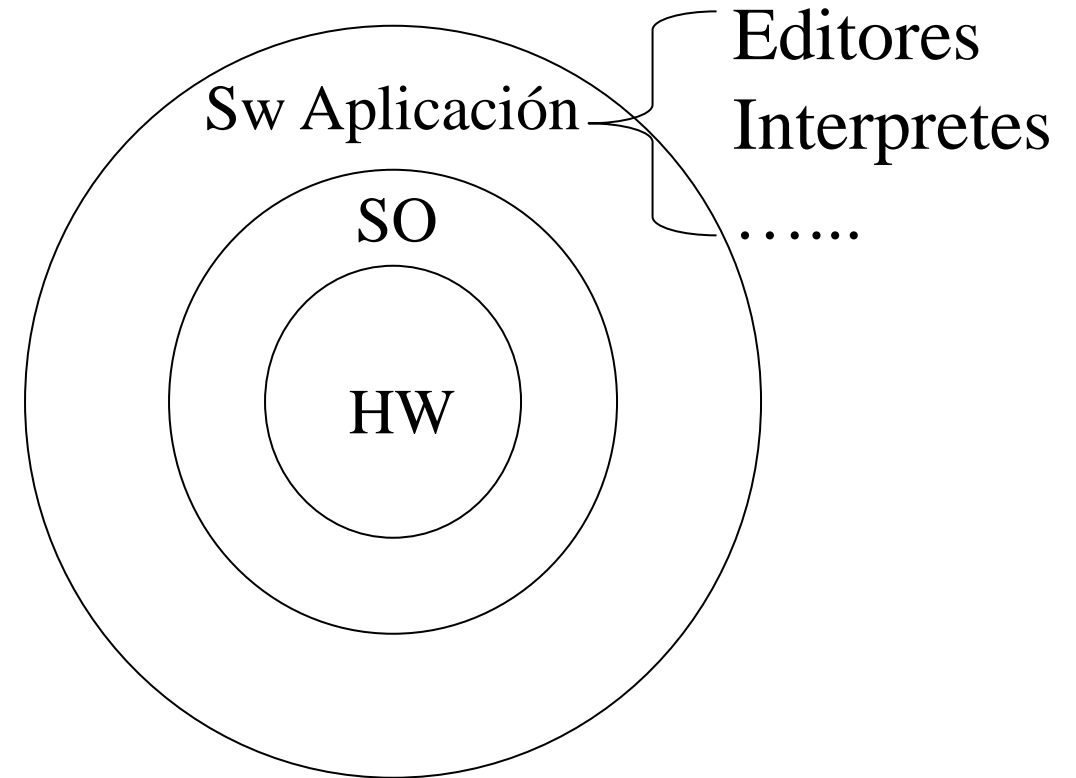
# Sistemas Operativos

---

## INTRODUCCIÓN

## Funciones del SO:

- Planificación, carga, iniciación y supervisión de la ejecución de programas
- Gestión memoria interna (multiusuario), Unidades E/S
- Inicio y Ctrl de operaciones E/S
- Tratamiento de Errores
- Coordinación Usuario<-->Sistema
- Mantenimiento Registro de Estado del sist.
- Ctrl de trabajos por lotes, multiproceso.
- Protección
- Contabilidad recursos....



# Codificación de la información

---

## INTRODUCCIÓN

**bit:** Unidad elemental, representa 0 ó 1

**Byte :** 8 bits

**Palabra:** nº de bytes que el ordenador (UC) es capaz de manejar de una sola vez.

KB= KiloBytes=  $2^{10}B=1024 B = 8192 b$

MB=Mega Bytes=  $2^{20}B=1024 KB$

GB= Giga Bytes =  $2^{30}B=1024 MB$

TB= Tera Bytes =  $2^{40}B=1024 GB$

El ordenador utiliza solamente el **Sistema Binario (0,1)**

$$34_{(10)} = 100010_{(2)}$$

Se suelen utilizar otros por cuestiones de facilidad de uso

$$10100001_{(2)} = 241_{(8)} = A1_{(16)}$$



Utilización de distintos métodos de representación en función del tipo del dato que se vaya a almacenar

- Números
  - Enteros: Modulo y signo, C1, C2,..
  - Reales: Coma Flotante
- Caracteres
  - ASCII, Unicode,....

El ordenador utiliza solamente el **Sistema Binario** (0,1)

$$34_{(10)} = 100010_{(2)}$$

Se suelen utilizar otros por cuestiones de facilidad de uso

$$10100001_{(2)} = 241_{(8)} = A1_{(16)}$$

**MS:** Un bit para el signo y el resto para el modulo

**C1:** Los n° positivos con su representación binaria, los negativos se representan cambiando 0 por 1, El bit de más a la izq. Valdrá 0 para los nums. Positivos y 1 para los negativos

**C2:** similar al C1 pero para evitar la doble representación del 0 se le suma 1 a los nums. Negativos después del cambio 0/1

**En Exceso:** Para 1 byte, el 0 se representaría como el 128, los positivos se sumarían los negativos se restarían.

**BCD:** Se utilizan 4 bits para cada dígito decimal

## Coma Flotante

El tamaño disponible para la representación se divide en 2 partes Mantisa y Exponente

$-324.8125_{(10)} = 101000100.1101_{(2)}$

Normalizado:  $-0.1010001001101 E9_{(2)}$

Representación para 4 Bytes.(exponente en exceso)

Signo

Mantisa

Exponente

Mantisa

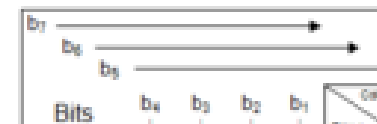
1	10001001	101000100110100000000000
---	----------	--------------------------

Existen códigos definidos que asignan a cada carácter, imprimible o no, un valor numérico preciso

- ASCII 8 bits por carácter (65-A, 90-Z, 97-a, 122-z, 164 - ñ,...)
- Unicode 16 bits por carácter

Se debe saber en todo momento como se debe interpretar la información almacenada (todo es ( Si se almacena un entero, un carácter ,....

**Tabla ASCII**

					0	1	2	3	4	5	6	7
Bits	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>				
0	0	0	0	0	0	0	0	0	NUL	DLE	SP	0
0	0	0	0	1	1	0	0	0	@	P	'	p
0	0	0	1	0	2	STX	DC2	"	2	B	R	b
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	:	K	[	k	{
1	1	0	0	12	FF	FC	,	<	L	\	l	
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

# Software

---

## INTRODUCCIÓN

El software es la parte inmaterial de la informática que hace que el hardware funcione.

El hardware necesita un conjunto de instrucciones escritas en un determinado lenguaje que la máquina entienda denominado **Lenguaje de programación**.

El conjunto de instrucciones que realizan una tarea concreta se denomina **Programa**.

Los programas procesan un conjunto de **datos** o informaciones de entrada convirtiéndolas en informaciones de salida.

Se puede clasificar al software en tres grandes tipos:

**Software de sistema:** Programas, herramientas y utilidades para controlar el funcionamiento interno del ordenador (gestión de memoria, discos, puertos, etc.).

Incluye entre otros:

- Sistemas operativos (Windows, Linux)
- Controladores de dispositivos
- Herramientas de diagnóstico
- Herramientas de Corrección y Optimización, etc.

**Software de programación:** Herramientas que permiten al programador desarrollar programas informáticos. Incluye entre otros:

- **Editores de texto, compiladores, intérpretes, depuradores.**
- **Entornos de Desarrollo Integrados (IDE):** Agrupan las anteriores herramientas, usualmente en un entorno visual, de forma tal que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etc. Habitualmente cuentan con una avanzada interfaz gráfica de usuario (GUI).

Será el software que estudiaremos en este módulo.

**Software de aplicación:** Es aquel que permite a los usuarios llevar a cabo tareas específicas en cualquier actividad. Incluye entre otros:

- Aplicaciones ofimáticas, bases de datos, software educativo, empresarial, diseño asistido por ordenador, videojuegos, etc





# Lenguajes de programación

---

## INTRODUCCIÓN

**DEF:** Lenguaje que permite la **comunicación entre el programador y el ordenador**. Posee alfabeto, sintaxis y semántica.

**DEF:** Lenguaje a través del cual se **describe el proceso que debe realizar el ordenador** de forma inteligible para el sistema informático.

Hay varios tipos, cada uno posee un alfabeto propio y tiene sus propias reglas sintácticas y semánticas

Forman unidades de comunicación llamadas **programas**.

Programa: Conjunto de **instrucciones** o sentencias, en un **lenguaje próximo a la máquina**, que ésta puede entender y ejecutar para realizar un determinado trabajo o proceso.

**Algoritmo:** Conjunto ordenado de pasos que indican la **secuencia de operaciones** que se ha de realizar para resolver un problema. (ejem. receta de cocina)

Todos los programas se desarrollan en algún lenguaje de programación. Nadie (o más bien casi nadie) programa directamente con instrucciones máquina, debido a que son inteligibles para el ser humano.

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

Habitualmente un lenguaje se compone de sintaxis y semántica, y los lenguajes de programación por tanto al ser un lenguaje tiene que tener una sintaxis (conjunto de normas y palabras reservadas) y una semántica (un significado al usar esas palabras y normas).

En la actualidad existen multitud de lenguajes de programación, los más populares según PYPL Popularity of Programming Language (<http://pypl.github.io/PYPL.html>) eran en 2020:

**Worldwide**, Sept 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	31.56 %	+2.9 %
2		Java	16.4 %	-3.1 %
3		Javascript	8.38 %	+0.3 %
4		C#	6.5 %	-0.8 %
5		PHP	5.85 %	-0.5 %
6		C/C++	5.8 %	+0.0 %
7		R	4.08 %	+0.3 %
8		Objective-C	2.79 %	+0.2 %
9		Swift	2.35 %	-0.1 %
10		TypeScript	1.92 %	+0.1 %

## Lenguaje máquina:

- Instrucciones complejas e ininteligibles, combinaciones de unos y ceros.
- No necesita ser traducido.
- Primer lenguaje utilizado.
- Difiere para cada procesador.

Instrucciones divididas en Cod. Operación y Operandos.

Consta de 3 tipos básicos de instrucciones:

- Ejecutivas: Manejan los datos de los regs. , la MP y la UAL
- De control: controlan el flujo de ejecución (inst. condicionales, bucles)
- De E/S : para comunicación con el exterior

Lenguaje de bajo nivel o ensamblador:

- Sustituyó al lenguaje máquina
- Sigue estando cercano al hardware, en lugar de unos y ceros, se programa usando mnemotécnicos.
- Necesita compilación para traducirse al lenguaje máquina.
- Se trabaja con registros del procesador y direcciones físicas de memoria.
- Difícil de comprender y programar

Suelen ser mnemotécnicos de las mismas instrucciones que el L. Máquina:

01000 001 INC CX

Lenguaje de medio nivel:

- Son precisos para ciertas aplicaciones como la creación de sistemas operativos, ya que permiten un manejo abstracto (independiente de la máquina, a diferencia del ensamblador), pero sin perder mucho del poder y eficiencia que tienen los lenguajes de bajo nivel.
- Son **independientes** de la **arquitectura**.
- Necesitan de un programa **traductor**
- Mayor abstracción. Facilitan la codificación
- Reducen el tamaño de los programas.
- Permiten la utilización de **tipos de datos** (enteros, reales, caracteres)

Lenguaje **de alto nivel**, amplían las características de los de medio nivel con:

- Tienen una forma de programar más intuitiva y sencilla.
- Mas cercano al lenguaje humano que al lenguaje máquina.
- Suelen tener librerías y funciones predeterminadas que solucionan algunos problemas comunes que suelen presentarse al programador.
- En ocasiones, ofrecen frameworks para una programación más eficiente y rápida.
- Suelen trabajar con mucha abstracción y orientación a objetos, para facilitar la reutilización y encapsulación de componentes.
- La mayoría de los lenguajes de programación pertenecen a esta categoría.



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

- **Lenguajes compilados.** Necesitan un programa traductor (compilador) para convertir el código fuente en código máquina. Tienen una ejecución rápida. Ej. C o C++
- **Lenguajes interpretados.** No se genera código objeto. El intérprete es un programa que tiene que estar cargado en memoria y se encarga de leer cada una de las instrucciones, interpretarlas y ejecutarlas. Ej. Python, o HTML
- **Lenguajes virtuales.** Son lenguajes más portables que los lenguajes compilados, puesto que el código que se genera tras la compilación es un código intermedio o *bytecode*. Este código puede ser, a su vez, interpretado por una máquina virtual instalada en cualquier equipo. Tienen una ejecución lenta, pero su versatilidad para poder ejecutarse en cualquier entorno los hace muy apreciados. Ej. Java.

## Clasificación según su **propósito**

- De **propósito General** (Basic, Pascal, C)
- **Orientados** al problema (Fortran, Cobol, Lisp, SQL)

## Clasificación según su **estilo de programación**

- Prog. Imperativa: *Pascal, C*
- POO: *Java, C++*
- Prog Funcional : *Lisp*
- Prog. Lógica: *Prolog*

**Actividad 1.** Clasifica los lenguajes siguientes según lo que se expone a continuación.

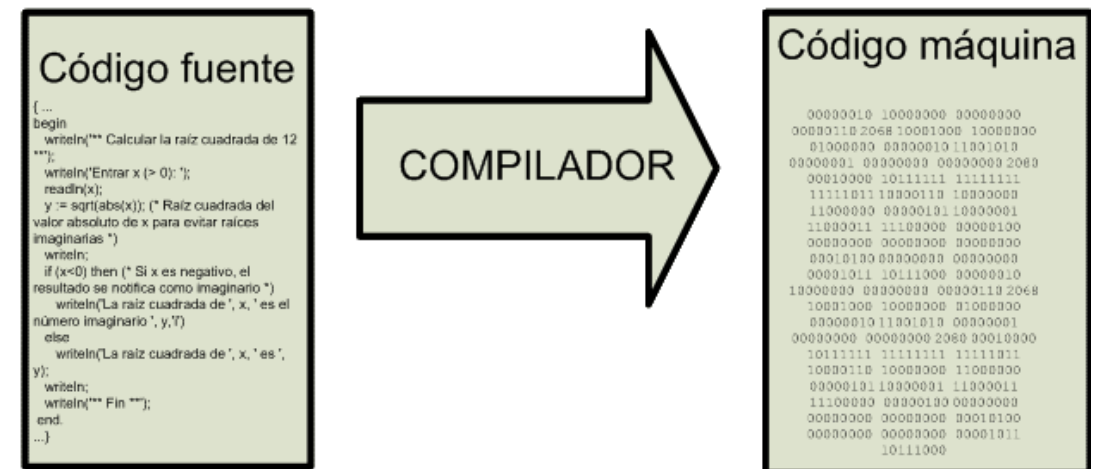
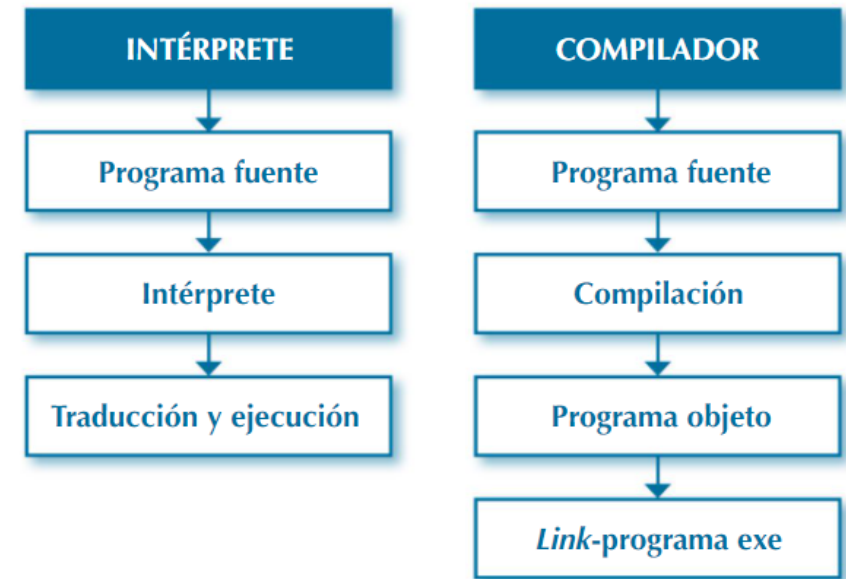
Lenguaje	Compilado/Interpretado/Virtual	Alto/Medio/Bajo Nivel
AngularJS.		
PHP		
Java.		
JavaScript.		
Ruby.		
Python.		
Cobol.		
C.		
Pascal.		
Turbo Pascal.		
C++.		
Objective C.		
Visual Basic.		
Swift.		
Ensamblador.		
Fortran.		
Prolog.		
Scala.		
Node.JS.		

# Proceso de traducción /compilación

---

Los *traductores* son programas cuya finalidad es traducir lenguajes de alto nivel a lenguaje de bajo nivel (código máquina). Existen dos grandes grupos de traductores:

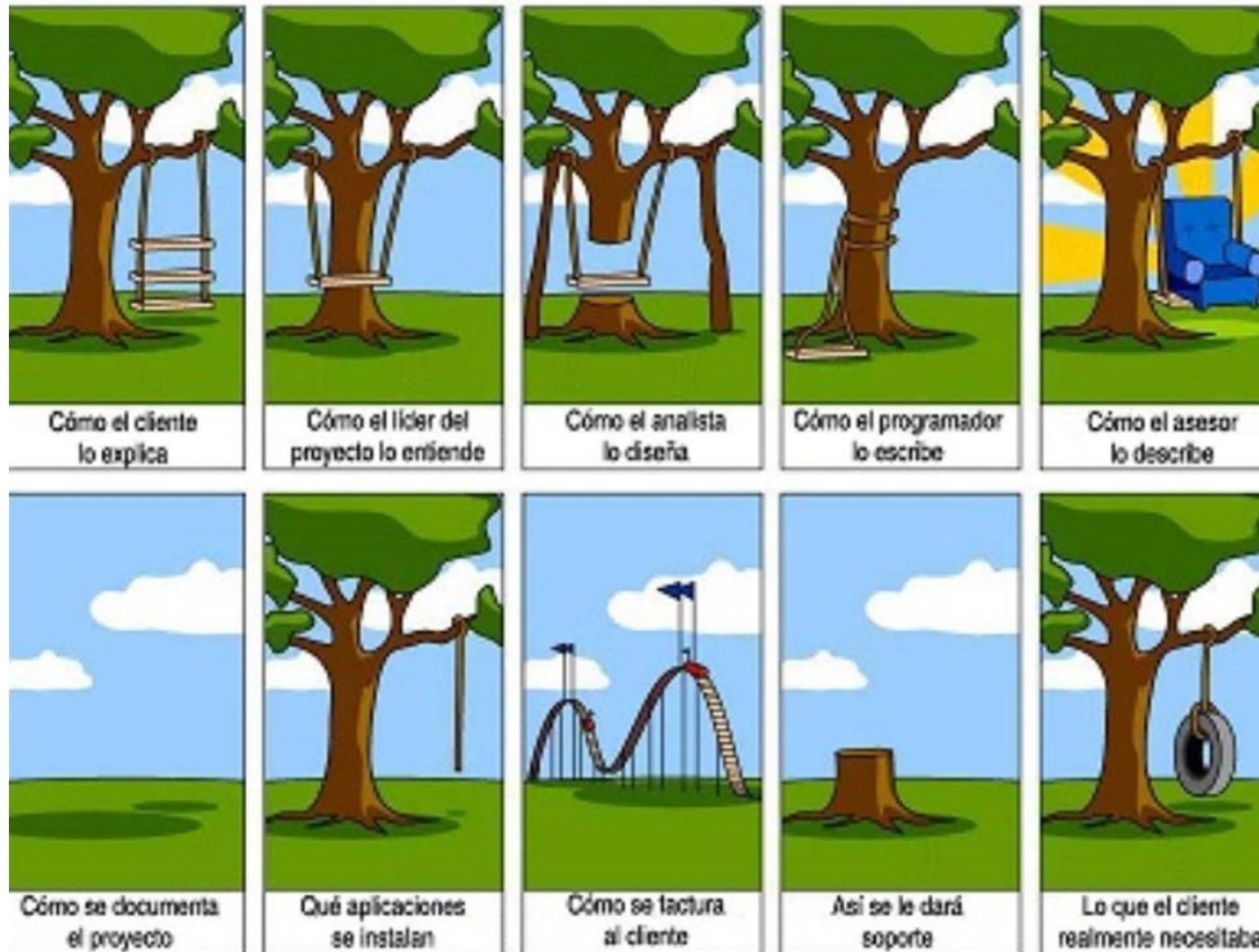
- Interpretes: traduce el código línea a línea. El interprete y el código fuente tiene que estar en memoria para poder ejecutar el programa.
- Compilador traduce el código fuente al código máquina. El compilador solo está en la máquina de desarrollo. El código generado sólo funcionará en una máquina con un Hardware y Software determinados.



- Extracción de las **Tablas de información** (variables, etiquetas, funciones...
  - Análisis **léxico** (escáner): localización y codificación de tokens
  - Análisis **sintáctico** (parser) y semántico
  - Generador de código.
- 
- **Linker: Enlaza** el código objeto de nuestro programa **con** las rutinas de las **librerías** (generales o específicas)

# Desarrollo de una aplicación

---





Existen muchos paradigmas de desarrollo, e incluso muchas metodologías para la gestión de un proyecto, pero en general podemos distinguir siete fases distintas.

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

## Fase inicial

Es dónde se planifica el proyecto, se hacen estimaciones y se decide si el proyecto es o no rentable. Por tanto se establecen las bases de como debe desarrollarse el resto de fases de un proyecto.

Las fases de planificación y estimación son las más complejas de un proyecto y se suele necesitar gente con mucha experiencia tanto en la elaboración de proyectos como en las plataformas y frameworks en los que se va a desarrollar la solución.

En esta fase se debe generar distintos documentos entre los que se incluye, las estimaciones, datos económicos, decisiones técnicas, etc...

## Análisis

En esta fase se analiza el problema, por tanto, consiste en recopilar, examinar y formular los requisitos del cliente y analizar cualquier restricción que pueda aplicarse. Todas las entrevistas con el cliente, por lo tanto, tienen que estar registradas en documentos.

Este documento es un documento formal de requisitos y debe ser consensuado con el cliente.

¿Qué se hace en esta fase? Se especifican y analizan los requisitos funcionales y no funcionales del sistema.

Requisitos:

- Funcionales: Qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
- No funcionales: Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas. La culminación de esta fase es el **documento ERS** (Especificación de Requisitos Software).

En este documento quedan especificados:

- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del usuario cliente y del sistema.
- Relación de los requisitos funcionales y no funcionales del sistema.
- Relación de objetivos prioritarios y temporización.
- Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.

## Diseño

Esta fase consiste en determinar los requisitos generales de la arquitectura de la aplicación y en dar una definición precisa de cada subconjunto de la aplicación. Suelen crearse dos documentos, uno más genérico con una visión general de la aplicación y sus componentes y otro más detallado en el que se profundizará en los detalles técnicos de cada módulo concreto del sistema.

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es **¿Cómo hacerlo?** Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas. Decidir qué hará exactamente cada parte. En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.

En este punto, se deben tomar decisiones importantes, tales como:

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

## Codificación o implementación

Esta fase consiste en la creación del código en un lenguaje de programación para crear las funciones definidas durante la etapa de diseño. Hay que tener en cuenta que un buen código es aquel que:

- Hace lo que tiene que hacer.
- Es legible.
- Está bien comentado y documentado.
- Es sencillo.
- Es flexible.

En definitiva un buen programa, está codificado pensando en la **mantenibilidad** del sistema, por ello hay que hacer hincapié en dos factores muy importantes, la documentación (buenos comentarios, creación y actualización del *javadoc*...) y la calidad del software (Herramientas como *Sonar* nos ayudan en esta tarea).

## Pruebas

Se realizarán pruebas para garantizar que la aplicación se ha programado de acuerdo con las especificaciones originales y los distintos programas que componen la aplicación están perfectamente integrados y preparados para la explotación del sistema. Podrían clasificarse en dos grandes tipos:

- Funcionales. Aquellas en las que se prueba que la aplicación hace lo que tiene que hacer con las funciones acordes a los documentos de especificaciones que se establecieron con el cliente. Dentro de esta destacamos las pruebas unitarias, de integración o de aceptación. Entre todas las pruebas de este tipo que se efectúan sobre el software podemos distinguir básicamente:
  - PRUEBAS UNITARIAS. Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). Junit o Google test es el entorno de pruebas para Java.
  - PRUEBAS DE INTEGRACIÓN. Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.

- No funcionales. Pruebas técnicas que prueban requisitos no funcionales como son la accesibilidad, seguridad o estabilidad de un sistema. Entre todas las pruebas de este tipo que se efectúan sobre el software destacamos:
  - PRUEBAS DE CARGA. Permiten conocer la estabilidad del sistema ante una subida de peticiones. Para llevar a cabo este tipo de pruebas se suele usar programas externos como [Jmeter](#) o [Locust](#).

## Actividad 2. Estudia y analiza las diferencias entre:

- [TDD](#) (Test Driven Development), [BDD](#) (Behavior Driven Development) y [ATDD](#) (Acceptance Test Driven Development)



## Explotación

En esta fase se instala el software en el entorno de producción y se trabaja con él. Suele ser la fase más larga en el tiempo y suelen surgir nuevos requisitos o producirse *Bugs* o *Incidencias*.

## Mantenimiento

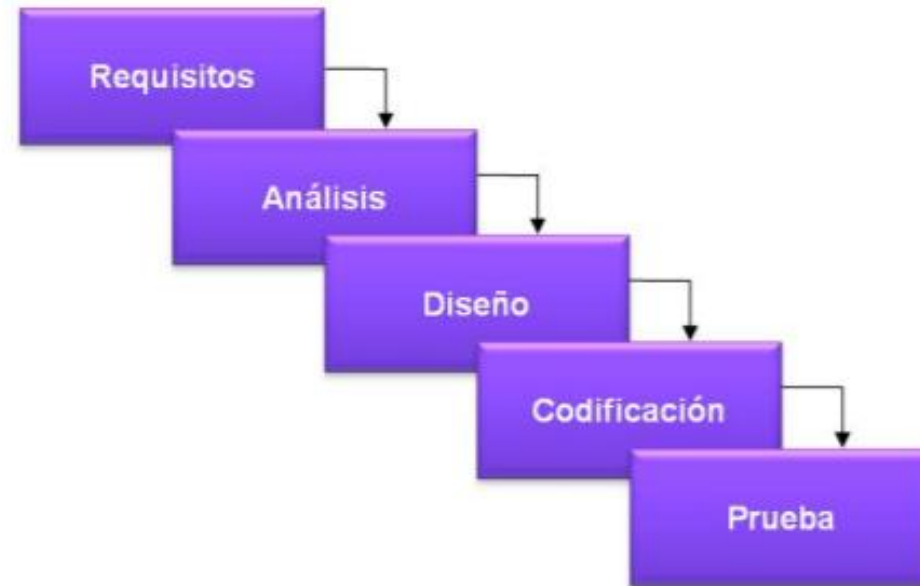
En esta fase se realiza todo tipo de procedimientos correctivos y actualizaciones de software consistente en adaptar y evolucionar la aplicación.

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

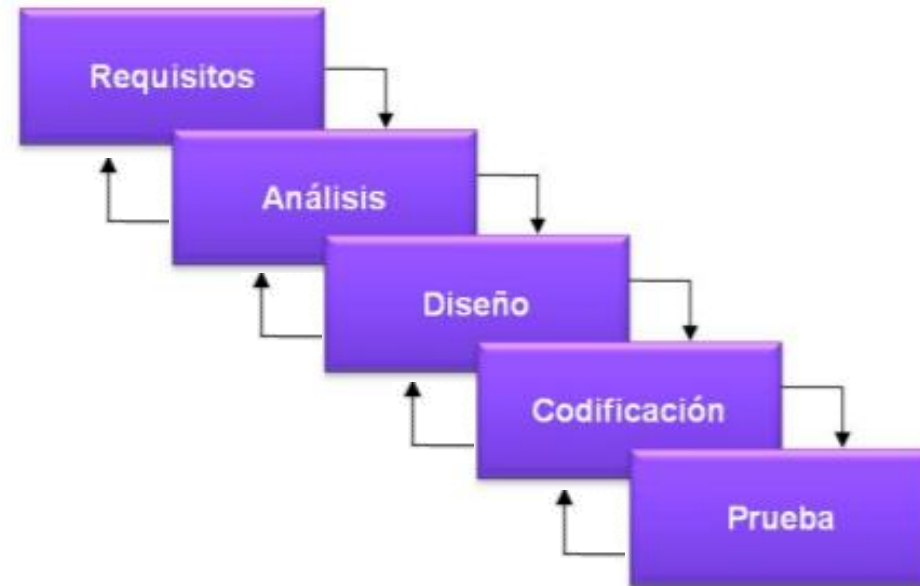
Podemos diferenciar los ciclos de vida en cascada (waterfall) o ágiles dependiendo de la metodología usada:

- Metodología tradicional. Está básicamente enfocada a la creación de productos tangibles. Es decir, barcos, puentes, coches... Su ciclo de Vida es en forma de cascada, ya que la organización de los proyectos se divide en diferentes etapas. Y estas etapas se ejecutan una sola vez y en el orden establecido. De esta forma, no se empieza la siguiente hasta que la anterior está validada y finalizada.
- Metodología Agile. No existe un orden predefinido de las etapas, sino que cada equipo va trabajando de forma interconectada. Esto consigue mejorar el resultado de cada una de las etapas además de permitir una rutina de trabajo más dinámica y participativa

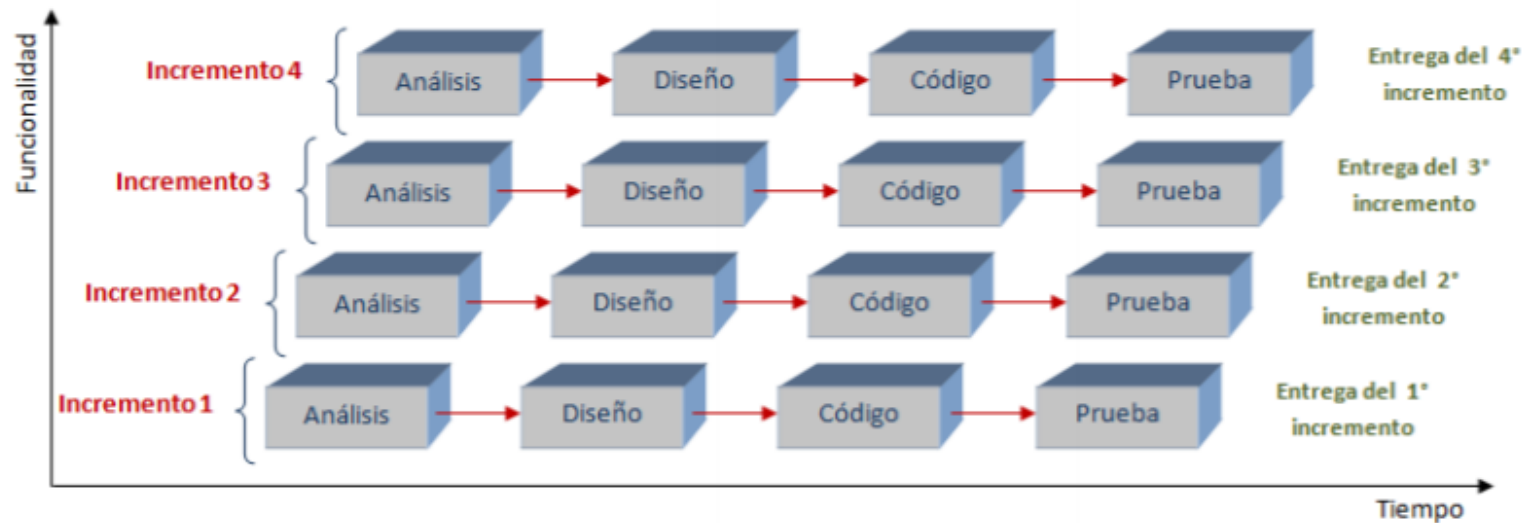
- Modelo en cascada. Propiciado por Winston Royce en 1970, sugiere un enfoque sistemático y secuencial, disciplinado y basado en análisis, diseño, pruebas y mantenimiento. Al final de cada etapa se reúnen y revisan los documentos para garantizar que se cumplen los requerimientos antes de avanzar a la fase siguiente.



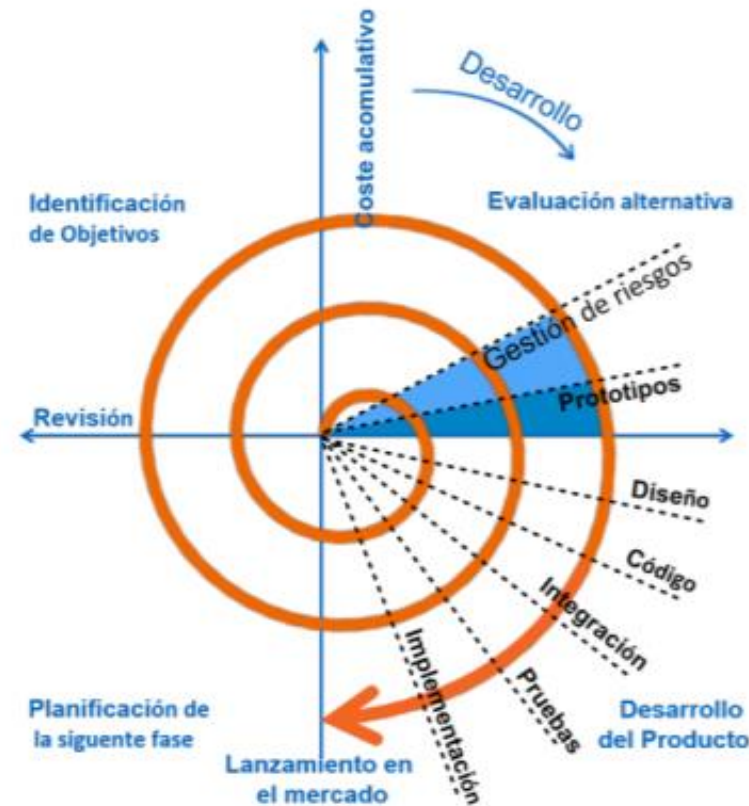
- Modelo en cascada con retroalimentación. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.



- Modelos Evolutivos. Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software. Distinguimos dos variantes:
  - Modelo Iterativo Incremental. Está basado en el modelo en cascada con retroalimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.



- Modelo en Espiral. Es una combinación del modelo anterior con el modelo en cascada. En él, el software se va construyendo repetidamente en forma de versiones que son cada vez mejores, debido a que incrementan la funcionalidad en cada versión. Es un modelo bastante complejo.



- *Jefe de proyecto.* Dirige el proyecto. Tiene que saber liderar un equipo, gestionar los tiempos, tener una relación fluida con el cliente, mantener un presupuesto.
- *Arquitecto.* Es la persona encargada de decidir como se va a realizar el proyecto y como va a cohesionarse. Tiene un conocimiento profundo en tecnologías, los frameworks, las librerías...
- *Analista.* Es la persona encargada de realizar un estudio exhaustivo del problema que ha de analizarse y ejecuta tanto el análisis como el diseño de la aplicación. Al igual que el jefe de proyecto, tiene que tener habilidades sociales para tener una relación fluida con los stakeholders.
- *Analista programador.* Puesto a caballo entre el analista y el programador, es un programador senior. Realiza funciones de análisis y sobre todo de diseño porque su conocimiento así lo permite, pero también realiza tareas de codificación.
- *Programador.* Su función es conocer el profundidad el lenguaje de programación y codificar las tareas que le han sido encomendadas por en analista o analista programador.

\* stakeholders. Personas de interés: empleados, proveedores, clientes, gobierno, etc.

Las metodologías clásicas se muestran, muchas veces, ineficientes en proyectos medianos o con mayores exigencias en tiempos de respuesta y requerimientos imprecisos y cambiantes. Se pasaba más tiempo pensando en el diseño y los controles que en hacer frente a posibles cambios en las especificaciones; estos no eran compatibles con la manera de realizar los análisis y la documentación, haciendo del desarrollo de software un proceso improductivo e ineficiente.

Las metodologías ágiles aparecen en respuesta a esa problemática del desarrollo de software. Son alternativas que procuran un enfoque en el software y no en la arquitectura o la documentación, con un enfoque iterativo se adaptan bien a los requerimientos cambiantes y entregas funcionales desde etapas tempranas con la participación del cliente.



Metodologías ágiles	Metodologías tradicionales
Se basan en heurísticas/ideas provenientes de prácticas de producción de código.	Se basan en normas provenientes de estándares seguidos por el entorno de desarrollo.
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo.	Énfasis en la definición del proceso: roles, actividades y artefactos.
Preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente por el equipo	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso muy controlado, numerosas normas
Contrato flexible e incluso inexistente	Contrato prefijado
El cliente es parte del desarrollo	Cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10) con pocos roles, más genéricos y flexibles.	Grupos grandes con más roles y más específicos.
Orientada a proyectos pequeños, y en el mismo lugar.	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes.
Pocos artefactos El modelo es prescindible, modelos desechables	Más artefactos. El modelo es esencial, mantenimiento en los modelos.
Menor énfasis en la arquitectura del software, se va definiendo y mejorando a lo largo del proyecto.	La arquitectura del software es esencial, se define tempranamente en el proyecto

1. **Satisfacer al cliente** entregándole un producto con valor es la prioridad máxima.
2. **Los requisitos pueden (y casi que deben) cambiar**, independientemente si nos encontramos al principio o al final del proyecto, ya que esos cambios siempre enriquecerán el producto final.
3. La idea es **entregar un producto funcional** en un período corto de tiempo.
4. **Desarrolladores y responsables del negocio** deben trabajar hombro con hombro.
5. La comunicación entre los miembros del equipo ha de ser, preferiblemente, **cara a cara**.
6. **Apreciación, confianza y empoderamiento** deben ser las tres coordinadas básicas del ambiente de un equipo que siga la metodología AGILE.
7. Que **el producto sea funcional** será el indicativo del progreso del proyecto.
8. Un **desarrollo sostenible** siempre será deseable.
9. Es necesario perseguir **la excelencia técnica y la calidad del diseño** para mejorar así la agilidad del proyecto.
10. **Simplicidad**, tu mejor amiga.
11. Hay que permitir **que los equipos se auto-gestionen** para que produzcan mejores productos.
12. Revisar el proyecto y **permitir que este se adapte a los cambios** conducirá a un clima en el que el equipo reflexionará sobre el producto.

Hay una gran variedad de metodologías agile, entre las más conocidas están, Extreme Programming o Scrum, pero todas las metodologías agile comparten como común denominador su alineación al llamado “manifiesto Ágil”, una declaración formal firmada en febrero del 2001 en Utah, USA, por 17 representantes de la industria del software. Desde entonces crean la “Agile Alliance” (<https://www.agilealliance.org>; @agilealliance) una de las más activas congregaciones dedicadas a promover la filosofía del desarrollo ágil, actualmente vigente y con representación mundial.

- [Extreme Programming](#). Propuesta por Kent Beck, en su trabajo fundamental que publicó en 1999 buscando guiar a equipos de desarrollo de software pequeños, entre dos y diez desarrolladores, en ambientes de requerimientos imprecisos o cambiantes. (Cadavid et al. 2013). Cinco valores fundamentan sus principios: Simplicidad, Comunicación, Retroalimentación, Respeto y Coraje. Sus postulados o principios son: Retroalimentación rápida, asumir simplicidad, el cambio incremental, la aceptación del cambio y el trabajo de calidad.

Los llamados Equipos Scrum son autogestionados, multifuncionales y trabajan en iteraciones (Sprints). Define tres roles:

- **Product Owner**: [Representa la voz del cliente](#). Se asegura de que el equipo Scrum trabaje de forma adecuada desde la perspectiva del negocio. El Product Owner escribe historias de usuario, las prioriza, y las coloca en el Product Backlog. Es una persona con experiencia de la empresa de desarrollo
- **ScrumMaster**: (o Facilitador): El Scrum es facilitado por un ScrumMaster, cuyo trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El ScrumMaster no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El ScrumMaster se asegura de que el proceso Scrum se utiliza como es debido. El ScrumMaster es el que hace que las reglas se cumplan.
- **Equipo de desarrollo**: El equipo tiene la responsabilidad de entregar el producto. Un pequeño equipo de 3 a 9 personas con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación).

Scrum es una **metodología ágil y flexible** para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para su empresa (ROI). Se basa en construir **primero la funcionalidad de mayor valor para el cliente** y en los principios de inspección continua, adaptación, autogestión e innovación.

A Scrum se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

La terminología Scrum define un evento temporal conocido como “**Sprint**” con una duración máxima de un mes en el que debe crearse una versión utilizable del producto. Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite. Cada Sprint contiene las siguientes etapas:

- **Reunión de planificación.** El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:
  1. Selección de requisitos. El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
  2. Planificación de la iteración. El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se auto asignan las tareas.

- **Ejecución de la iteración.** Cada día el equipo realiza una reunión de sincronización (15 minutos máximo). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:
  - ¿Qué he hecho desde la última reunión de sincronización?
  - ¿Qué voy a hacer a partir de este momento?
  - ¿Qué impedimentos tengo o voy a tener?

Durante la iteración el Facilitador (Scrum Master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

- Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad



- **Inspección y adaptación:** El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:
1. Demostración. El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
  2. Retrospectiva. El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador (Scrum master) se encargará de ir eliminando los obstáculos identificados.

## BENEFICIOS DE SCRUM

- Flexibilidad a cambios. Gran capacidad de reacción ante los cambiantes requerimientos generados por las necesidades del cliente o la evolución del mercado.
- Reducción del Time to Market. El cliente puede empezar a utilizar las características más importantes del proyecto antes de que esté completamente terminado.
- Mayor calidad del software. El trabajo metódico y la necesidad de obtener una versión de trabajo funcional después de cada iteración, ayuda a la obtención de un software de alta calidad.
- Mayor productividad. Se logra, entre otras razones, debido a la eliminación de la burocracia y la motivación del equipo proporcionado por el hecho de que son de que pueden estructurarse de manera autónoma.
- Maximiza el retorno de la inversión (ROI). Creación de software solamente con las prestaciones que contribuyen a un mayor valor de negocio gracias a la priorización por retorno de inversión.
- Predicciones de tiempos. A través de este marco de trabajo se conoce la velocidad media del equipo por sprint, con lo que es posible estimar de manera fácil cuando se podrá hacer uso de una determinada funcionalidad que todavía está en el Backlog.
- Reducción de riesgos El hecho de llevar a cabo las funcionalidades de mayor valor en primer lugar y de saber la velocidad a la que el equipo avanza en el proyecto, permite despejar riesgos efectivamente de manera anticipada.

# Documentación

---

Como hemos estado viendo durante las fases, en cada una de ellas se puede y se tiene que ir generando cierta documentación. Una vez finalizado cualquier proyecto como mínimo, deben generarse los siguientes documentos:

- Manual de usuario. Es el manual que usará el usuario para desenvolverse con el programa. Deberá ser autoexplicativo y de ayuda para el usuario.
- Manual técnico. Es el manual dirigido a los técnicos. Con esta documentación cualquier técnico que conozca el lenguaje deberá poder continuar con el mantenimiento de la aplicación.
- Manual de instalación. En este manual, se explica paso a paso los requisitos y como se instala y pone en funcionamiento la aplicación.