

# ENTORNOS DE DESARROLLO

---

*1º Grado Superior de Desarrollo de  
aplicaciones web*



Software Development

# BLOQUE 1: RECONOCIMIENTO DE ELEMENTOS DEL DESARROLLO DE SOFTWARE

# 1. SOFTWARE Y PROGRAMA. TIPOS DE SOFTWARE

EL ordenador se compone por dos partes:

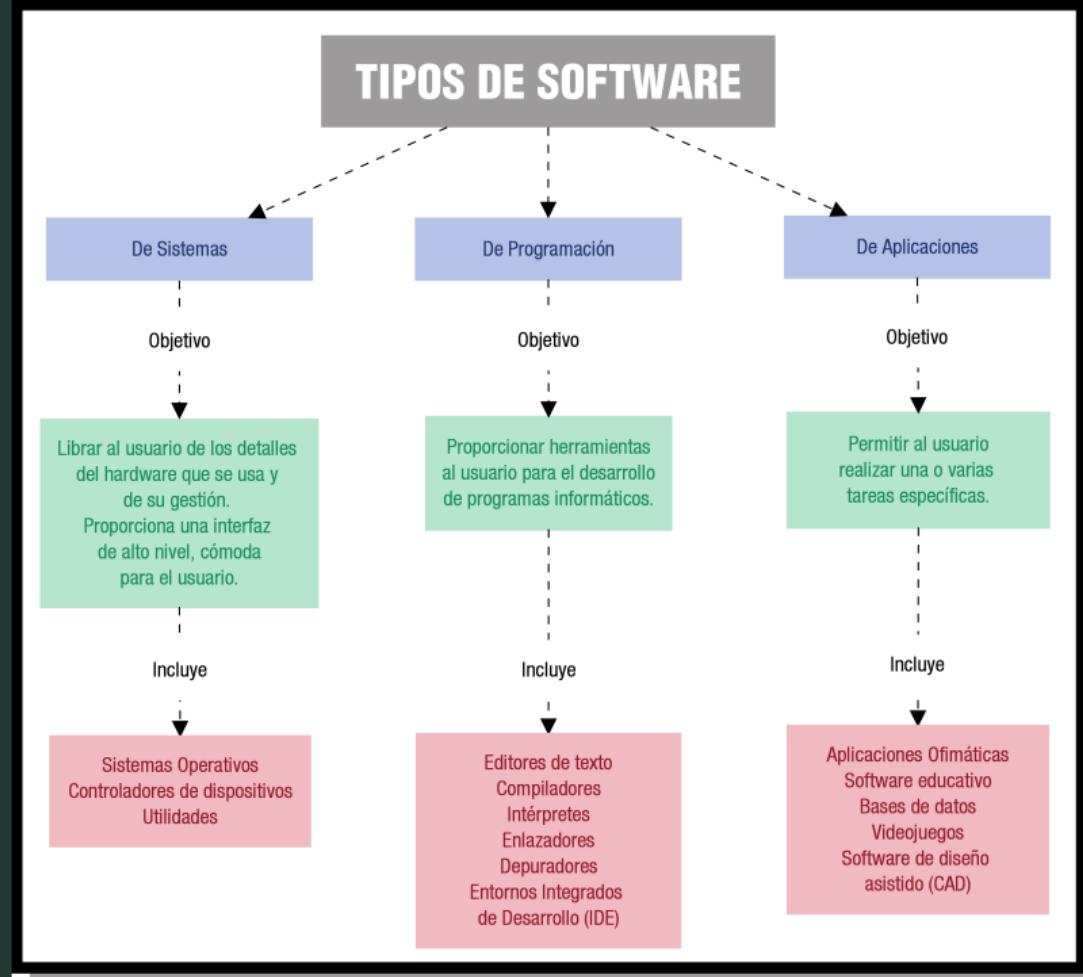
- Hardware
- Software

El Software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar algo que el usuario desee.

Sistema operativo

Software de programación

Aplicaciones

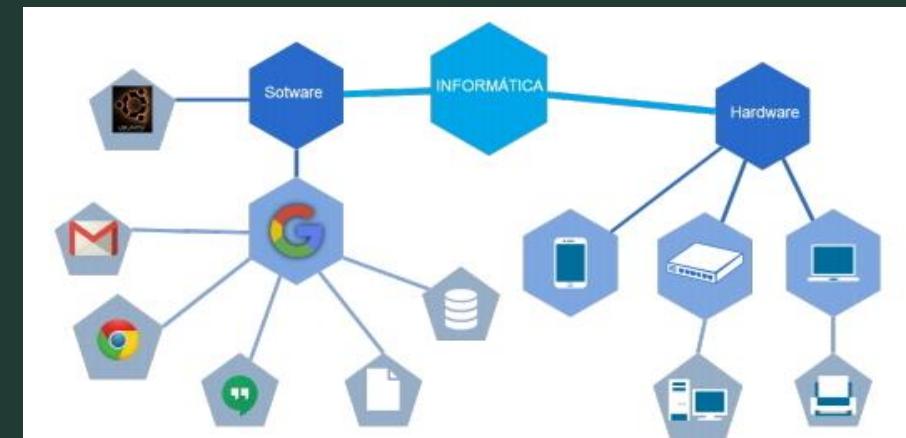


## PREGUNTAS

- ¿Cuáles son los sistemas operativos más conocidos?
- ¿Por qué se utiliza tanto Linux?

## 2. RELACIÓN HARDWARE- SOFTWARE

- Hardware: conjunto de dispositivos físicos que forman un ordenador.
- Relación: El software se ejecutará sobre los dispositivos físicos.
- Dos puntos de vista:
  - *Desde el punto de vista del sistema operativo.*
  - *Desde el punto de vista de las aplicaciones*



### 3. DESARROLLO DE SOFTWARE

- Proceso por el que hay que pasar desde que se concibe la idea de un programa hasta que dicho programa está implementado en el ordenador y funcionando.
- CICLOS DE VIDA DEL SOFTWARE
  - *Modelo en Cascada.*
  - *Modelo en Cascada con Retroalimentación.*
  - *Modelos evolutivos:*
    - + *Modelo Iterativo Incremental.*
    - + *Modelo en Espiral.*

### 3. DESARROLLO DEL SOFTWARE

- HERRAMIENTAS DE APOYO AL DESARROLLO DEL SOFTWARE

Las herramientas CASE son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo de proceso, mejorando por tanto la productividad del proceso.

Principalmente nos ayudan en el diseño de proyectos, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores,...

El desarrollo rápido de aplicaciones o RAD es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE.

### 3. DESARROLLO DEL SOFTWARE

La tecnología CASE trata de automatizar las fases del desarrollo de software para que mejore la calidad del proceso y del producto final.

En concreto estas herramientas permiten:

- Mejorar la planificación del proyecto.
- Darle agilidad al proceso.
- Poder reutilizar partes del software en proyectos futuros.
- Hacer que las aplicaciones respondan a estándares.
- Mejorar la tarea del mantenimiento de los programas.
- Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

### 3. DESARROLLO DEL SOFTWARE

CLASIFICACIÓN: se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- U-CASE: ofrece ayuda en las fases de planificación y análisis de requisitos.

- M- CASE: ofrece ayuda en el análisis y diseño.

- L- CASE: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker,...

### 3. DESARROLLO DEL SOFTWARE

- Otras clasificaciones que podemos encontrar son:

Según el tipo de integración:

Juegos de herramientas (Toolkits) consisten en una serie de herramientas independientes entre sí, donde cada una de ellas sirve para realizar una función. Por ejemplo, podría haber un compilador/depurador específico (gcc), un IDE para proporcionar una interface gráfica (Eclipse) y un sistema para gestionar las versiones (SVN).

Bancos de trabajo (Workbenches) se trata de un conjunto de herramientas integradas que comparten una base de datos de soporte y una interface única. Están orientados a una metodología de desarrollo, y ayudan al usuario a seguir la misma. La salida de una de las fases metodológicas se puede utilizar directamente de entrada para la fase siguiente.

### 3. DESARROLLO DEL SOFTWARE

Según la metodología utilizada:

**Metodología estructurada:** fue la primera que se usó de forma extendida, y a la que las primeras herramientas CASE daban soporte. Las herramientas de este tipo permiten gestionar proyectos de desarrollo mediante la construcción y gestión de DFDs, modelos Entidad/Relación, Diagramas de módulos, etc..

**Metodología orientada a objetos:** surgió con posterioridad, cuando ya las herramientas CASE estaban más maduras. Las técnicas de la MOO son muy diferentes, y están basadas sobre todo en el lenguaje UML y en el desarrollo iterativo basado en casos de uso. Hoy en día es más común desarrollar con esta metodología y con herramientas que la soportan.

También existen herramientas mixtas, que permiten soportar ambas metodologías.

## 4 . LENGUAJES DE PROGRAMACIÓN

¿Qué es un lenguaje de programación?

- Se trata de un lenguaje artificial, creado por programadores
- Permiten traducir las instrucciones de un programa a código comprensible por el ordenador
- Son lenguajes mucho más complejos y precisos que el lenguaje máquina (01010101)
- Cada instrucción puede dar lugar a muchas instrucciones de lenguaje máquina
- Se componen de un conjunto de símbolos (sintaxis) y reglas (semántica)

## 4 . LENGUAJES DE PROGRAMACIÓN

- Algunas definiciones importantes:
  1. *Lenguaje máquina: lenguaje que sólo es comprensible por el ordenador (0101101)*
  2. *Programa: conjunto de instrucciones escritas en un lenguaje de programación*
  3. *Instrucción: cada una de las sentencias u órdenes que forman parte de un programa.*
  4. *Palabra reservada: cada uno de los símbolos(léxico) que componen la sintaxis de un lenguaje.*
  5. *Semántica: reglas que definen la combinación de los símbolos de un lenguaje de programación.*

# 4 . LENGUAJES DE PROGRAMACIÓN

## 4.1. Tipos de lenguajes de programación

- Existen diferentes tipos de lenguajes de programación
- Los lenguajes de programación han evolucionado a lo largo de los años
- Actualmente, los lenguajes son más amigables y facilitan mucho la tarea del programador
- A veces, la facilidad a la hora de programar implica crear programas más lentos y pesados
- En la actualidad existen infinidad de lenguajes de programación:  
*Fortran, Cobol, Pascal, C, C++, Basic, Visual Basic, C#, Java, JavaScript, PHP, Python...*

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

Analicemos ahora los tipos de lenguajes de programación más importantes

### Lenguaje máquina

- *Posee instrucciones complejas e ininteligibles (01010101)*
- *Necesita ser traducido (es el único lenguaje que entiende directamente el ordenador)*
- *Fue el primer lenguaje usado (muy dependiente del hardware)*
- *Difiere para cada procesador (las instrucciones son diferentes de un ordenador a otro)*
- *En la actualidad sólo se usa para determinados módulos de un sistema operativo, drivers...*

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguaje de medio nivel (ensamblador)*

- *Facilita la labor de programación*
- *Se centra en el hardware, pero usa mnemotécnicos (ADD...) comprensibles por el programador*
- *Hay que compilarlos (traducirlos) para que sean comprensibles por el ordenador*
- *Trabaja con los registros del procesador y direcciones de memoria físicas*
- *A pesar de estas mejoras, es difícil de entender y de usar*

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguajes de alto nivel*

- *Poseen una forma de programar intuitiva y sencilla*
- *Son más cercanos a los lenguajes humanos (IF, WHILE, DO...)*
- *Incorporan librerías y funciones predeterminadas que ayudan al programador en su labor*
- *Suelen ofrecer frameworks, que incorporan funciones y componentes que facilitan el desarrollo*
- *La mayoría de los lenguajes de programación actuales se engloban en esta categoría*
- *Ahora analizaremos las características de algunos de estos lenguajes*

## 4.1 TIPOS DE LENGUAJES DE PROGRAMACIÓN

*Lenguajes de alto nivel*

*Lenguaje C/C++*

- *Uno de los más potentes y utilizados, ligado a Unix y a Linux*
- *Estructurados y ligados a funciones*
- *Incluyen el concepto de puntero (necesario para gestionar la memoria, pero un concepto complejo)*
- *Combinan comandos de alto nivel*
- *Programas eficientes, rápidos y pequeños*
- *Necesidad de compilar los programas*

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

Lenguaje C/C++

```
using namespace std;

int main (int argc, char *argv[])
{
    cout << "Hola mundo" << endl;
    return 0;
}
```

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguaje Java*

- *Simplificación de C++ (no admite sobrecarga de operadores ni herencia múltiple)*
- *Manejo más eficiente de cadenas de caracteres (String)*
- *Lenguaje 100% Orientado a objetos*
- *Pensado para trabajar con redes y protocolos de red (TCP/IP, HTTP, HTTPS...)*
- *Portable, seguro y permite la programación concurrente*
- *Es un lenguaje virtual interpretado*

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguaje Java*

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

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguaje Python*

- *Lenguaje de alto nivel muy portable*
- *Lenguaje interpretado*
- *Orientado a objetos*
- *Permite incrustarse en otros lenguajes como C/C++*
- *Uno de los lenguajes más simples y sencillos de aprender*

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

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguaje JavaScript*

- *Se utiliza mucho en programación web (Angular, React...)*
- *Se parece mucho a Java, C y C++*
- *Es un lenguaje de scripting, seguro y fiable*
- *Se ejecuta en el lado del cliente (Frontend)*
- *Su código es visible (hay que tener en cuenta aspectos de seguridad)*

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

## 4.1. TIPOS DE LENGUAJES DE PROGRAMACIÓN

### *Lenguaje PHP*

- *Lenguaje web de propósito general usado en el servidor (Backend)*
- *Se emplea en aplicaciones como Prestashop, WordPress...*
- *Es un lenguaje multiplataforma*
- *Orientado al desarrollo de webs dinámicas*
- *Buena integración con MySQL*
- *Orientado a objetos*
- *Lenguaje interpretado*

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

## 4.2. PARADIGMAS DE PROGRAMACIÓN

- Los lenguajes de programación pueden usarse de formas diferentes
- Un paradigma de programación define la forma en la que se desarrolla el programa
- Existen distintos tipos de paradigmas:
  1. Programación estructurada o imperativa
  2. Programación funcional
  3. Programación orientada a objetos
  4. Programación lógica

## 4.2. PARADIGMAS DE PROGRAMACIÓN

- Programación estructurada
  - *Consiste en una secuencia ordenada y organizada de instrucciones*
  - *Es fácil de entender*
  - *Programas sencillos y rápidos*
  - *Posee tres estructuras básicas: secuencia, selección e iteración*
  - *Inconveniente:* un único bloque de programa (*es inmanejable si crece*)

## 4.2. PARADIGMAS DE PROGRAMACIÓN

- Programación funcional
  - *Consiste en descomponer el programa en funciones o módulos.*
  - *Programa modular y estructurado.*
  - *Inconveniente: El programa puede crecer en gran medida y ser complejo*

## 4.2. PARADIGMAS DE PROGRAMACIÓN

- Programación orientada a objetos
  - *Consiste en representar entidades del mundo real*
  - *Permite reutilizar código*
  - *Principio de separación de responsabilidades*
  - *Patrones de diseño y paradigmas avanzados (MVC...)*
  - *Polimorfismo, herencia y encapsulación*

## 4.2. PARADIGMAS DE PROGRAMACIÓN

### Programación lógica

- *Consiste en representar predicados y relaciones.*
- *Uso de lógica de predicados de primer orden.*
- *Muy utilizado en aplicaciones de inteligencia artificial*

# 5. PROCESO DE TRADUCCIÓN/COMPILEACIÓN

## 5.1. Traductores de código

- Un traductor traduce un lenguaje de alto nivel a ensamblador o lenguaje máquina
- Existen dos tipos de traductores: intérpretes y compiladores
- Asimismo, en función de la forma de ejecutar un lenguaje existen los siguientes tipos:
  1. Lenguajes compilados
  2. Lenguajes interpretados
  3. Lenguajes virtuales

## 5.1. TRADUCTORES DE CÓDIGO

### LENGUAJES COMPILADOS

- Necesitan un compilador para traducir el código fuente a código máquina
- Se ejecutan más rápida que los programas interpretados o virtuales
- Precisan de un programa enlazador que permite unir el código objeto con el código objeto de librerías
- Aunque el código es más seguro, no son tan flexibles para modificarlos como los lenguajes interpretados
- Ejemplo: C/C++

## 5.1. TRADUCTORES DE CÓDIGO

### LENGUAJES INTERPRETADOS

- *No generan código objeto.*
- *El intérprete es un programa que tiene que estar cargado en memoria*
- *El intérprete "interpreta" cada sentencia del programa y lo ejecuta*
- *Las instrucciones se traducen "on the fly" (al vuelo), a medida que van ejecutándose*
- *Ejemplo: PHP*

## 5.1. TRADUCTORES DE CÓDIGO

### LENGUAJES VIRTUALES

- *Son más portables que los lenguajes compilados*
- *El código se genera tras la compilación en un código intermedio o bytecode*
- *El código puede ser interpretado por una máquina virtual instalada en cualquier equipo*
- *Son más lentos, pero más versátiles*
- *Ejemplos: Java*

## 5.1. TRADUCTORES DE CÓDIGO

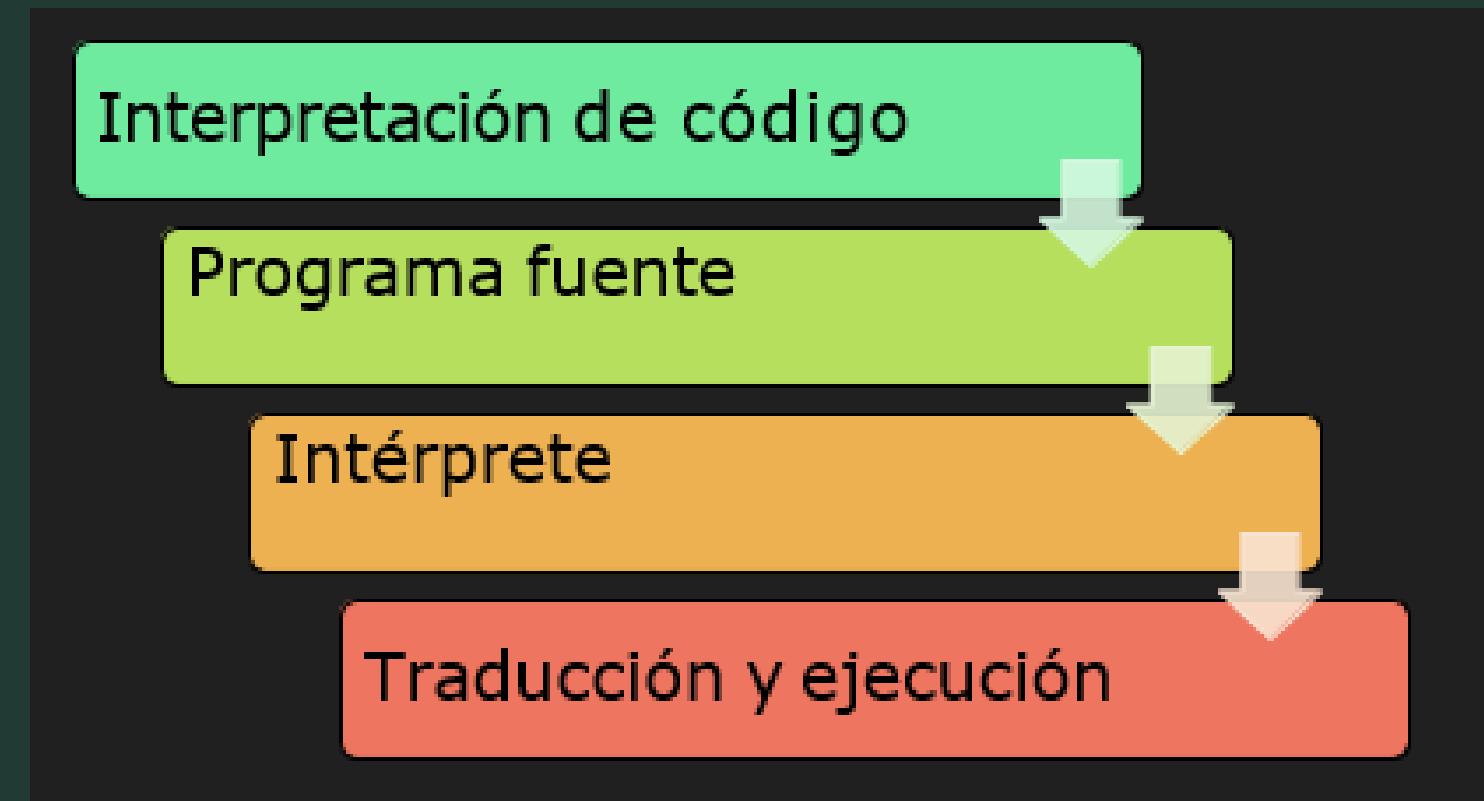
Algunas definiciones importantes:

- Código fuente: código de un programa, escrito por un programador en un lenguaje de programación
- Compilar: proceso que traduce el código fuente en código objeto (comprendible por un ordenador)
- Código objeto: código máquina generado tras la compilación del código fuente
- Librería: código externo que se incluye al programa para proporcionar funcionalidad adicional
- Archivo ejecutable: programa ejecutable que puede ser ejecutado en el ordenador  
*Incluye librerías (tras ser enlazadas), complementos, módulos...*

## 5.2. INTERPRETACIÓN DEL CÓDIGO

- Un intérprete traduce el código fuente línea a línea.
- El intérprete tiene que estar en memoria ejecutándose para ejecutar el programa.
- El código fuente del programa también debe ser carga en memoria para poder ser interpretado.

## 5.2. INTERPRETACIÓN DEL CÓDIGO



## 5.3. COMPILEACIÓN DEL CÓDIGO

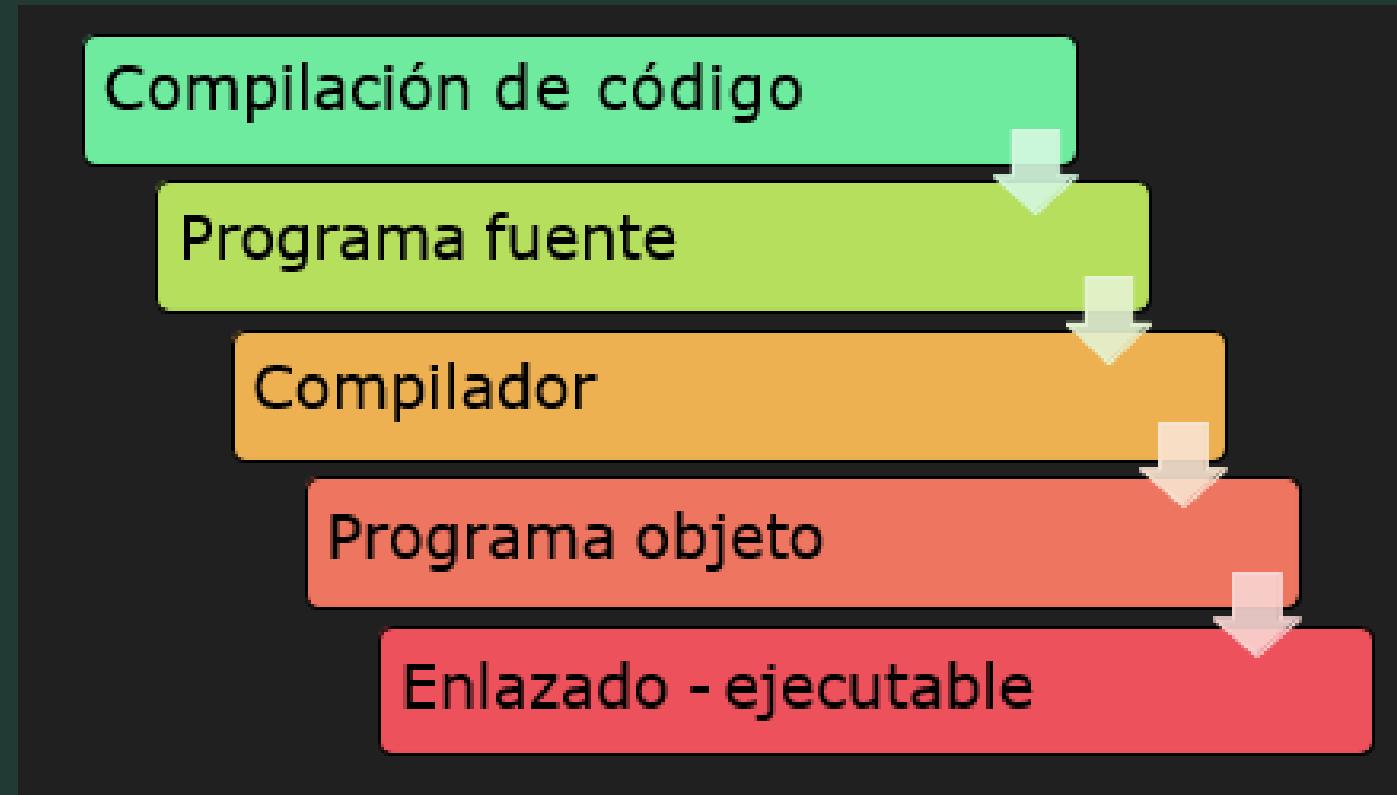
- Un compilador traduce código fuente a código máquina
- El compilador se instala en cada máquina en la que queramos compilar el programa
- El compilador depende de la arquitectura hardware de la máquina
- El proceso de compilación supone las siguientes fases:

*Preprocesado: se traducen y se ejecutan los comandos de preprocessamiento*

*Generación de código intermedio: generación de código máquina*

*Enlazado: se enlazan el código objeto con librerías externas*

## 5.3. COMPILACIÓN DE CÓDIGO



## 5.3. COMPILACIÓN DE CÓDIGO

### FASES DE UN COMPILADOR



# 6. DESARROLLO DE UNA APLICACIÓN

## 6.1. Fases del desarrollo de una aplicación

- El software se desarrolla siguiendo un paradigma o una metodología de desarrollo.
- Generalmente, los paradigmas suelen tener varias fases en común.
- Las fases más comunes son:
  1. Fase inicial
  2. Análisis.
  3. Diseño.
  4. Implementación.
  5. Pruebas.
  6. Explotación
  7. Mantenimiento
  8. Retirada

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN

### FASE INICIAL

- Planificación del proyecto
- Estimación de costes (viabilidad del proyecto)
- Es la fase más compleja
- Precisa de expertos en planificación de proyectos
- Se desarrollan documentos muy importantes para el proyecto (viabilidad, estimación...)

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN

### FASE DE ANÁLISIS

- Consiste en analizar el problema
- Recopilar, examinar y formular los requisitos del cliente (especificación de requisitos)
- Análisis de restricciones, entrevistas con el cliente y los usuarios finales
- Se genera un documento vinculante (a modo de contrato) entre el cliente y el desarrollador

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN

### FASE DE DISEÑO

- Consiste en determinar los requisitos generales de la arquitectura de la aplicación
- Se define cada subconjunto de la aplicación
- Los documentos son mucho más técnicos
- Esta fase involucra a los jefes de proyecto, arquitectos de software y analistas
- Los programadores aún no intervienen en esta fase

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN

### FASE DE IMPLEMENTACIÓN

- Consiste en implementar el software usando lenguajes de programación, librerías, frameworks...
- Se crea documentación muy detallada en la que se incluye y documenta el código fuente
- Parte del código suele comentarse sobre el propio código fuente generado
- Se detallan las entradas, salidas, parámetros... de cada uno de los módulos del programa.
- El detalle es máximo, pensando en el mantenimiento y soporte futuro que tendrá el programa

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN

### PRUEBAS

- Se realizan pruebas para garantizar que la aplicación se ha programado según las especificaciones
- A modo preliminar, se pueden considerar dos categorías de pruebas:
  - Pruebas funcionales: se prueba que a la aplicación hace lo que tiene que hacer (con el cliente)*
  - Pruebas estructurales: se efectúan pruebas técnicas sobre el sistema (estrés, carga, integración...)*
- En temas posteriores se abordará la tipología y el desarrollo de las pruebas de software

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN EXPLOTACIÓN

- Se instala el software en el entorno real de uso
- Se trabaja con el software de forma cotidiana (nuevas necesidades, incidencias...)
- Se recogen los errores y las correcciones en un nuevo documento de mantenimiento
- Los programadores y analistas revisan esos fallos para mejorar el software y aprender de los errores

## 6.1.FASES DEL DESARROLLO DE UNA APLICACIÓN MANTENIMIENTO

- Se realizan procedimientos correctivos sobre el programa
- Siempre hay que tener delante la documentación técnica de la aplicación
- Las operaciones de mantenimiento se deben documentar para dejar constancia de los cambios

## 6.1. FASES DEL DESARROLLO DE UNA APLICACIÓN RETIRADA

- El software ha llegado al final de su vida útil
- No resulta rentable seguir ampliándolo o manteniéndolo
- Llegados a este punto, el ciclo puede comenzar de nuevo:

*Comprando un nuevo software*

*Desarrollando un nuevo software (a medida)*

## 6.2. DOCUMENTACIÓN

- Como hemos explicado en la sección anterior, la documentación es vital
- En cada fase se generan uno o más documentos
- La documentación es vital para saber cómo usar la aplicación final
- Como mínimo, cada aplicación debe tener estos documentos:

*Manual de usuario: explica cómo usará el usuario la aplicación*

*Manual técnico: documentación dirigida a los técnicos (administradores y programadores)*

*Manual de instalación: detalla el proceso de instalación de la aplicación*

## 6.3. ROLES DEL DESARROLLO DE SOFTWARE

Los roles del proceso de desarrollo de software son:

### ARQUITECTO DE SOFTWARE

- Decide cómo se realiza el proyecto y cómo va a estructurarse
- Tiene un amplio conocimiento de las tecnologías, los frameworks y las librerías
- Decide y forma los recursos del desarrollo de un proyecto

## 6.3. ROLES DEL DESARROLLO DE SOFTWARE

### JEFE DE PROYECTO

- Dirige el curso del proyecto
- Puede ser un analista con experiencia, un arquitecto o una persona dedicada a ese puesto en exclusividad
- Debe saber gestionar un equipo y lidiar con los tiempos
- Trata de manera continua y fluida con el cliente

## 6.3. ROLES DEL DESARROLLO DE SOFTWARE

### ANALISTA DE SISTEMAS

- Esta persona realiza un estudio exhaustivo del problema que va a llevarse a cabo
- Efectúa el análisis y el diseño de todo el sistema
- Este perfil requiere mucha experiencia, y también suele involucrarse en reuniones con el cliente

## 6.3. ROLES DEL DESARROLLO DE SOFTWARE

### ANALISTA PROGRAMADOR

- Gran manejo de la programación.
- Investiga, analiza, desarrolla e implementa soluciones.
- Realiza el mantenimiento y actualizaciones de software y aplicaciones.

## 6.3. ROLES DEL DESARROLLO DE SOFTWARE

### PROGRAMADOR

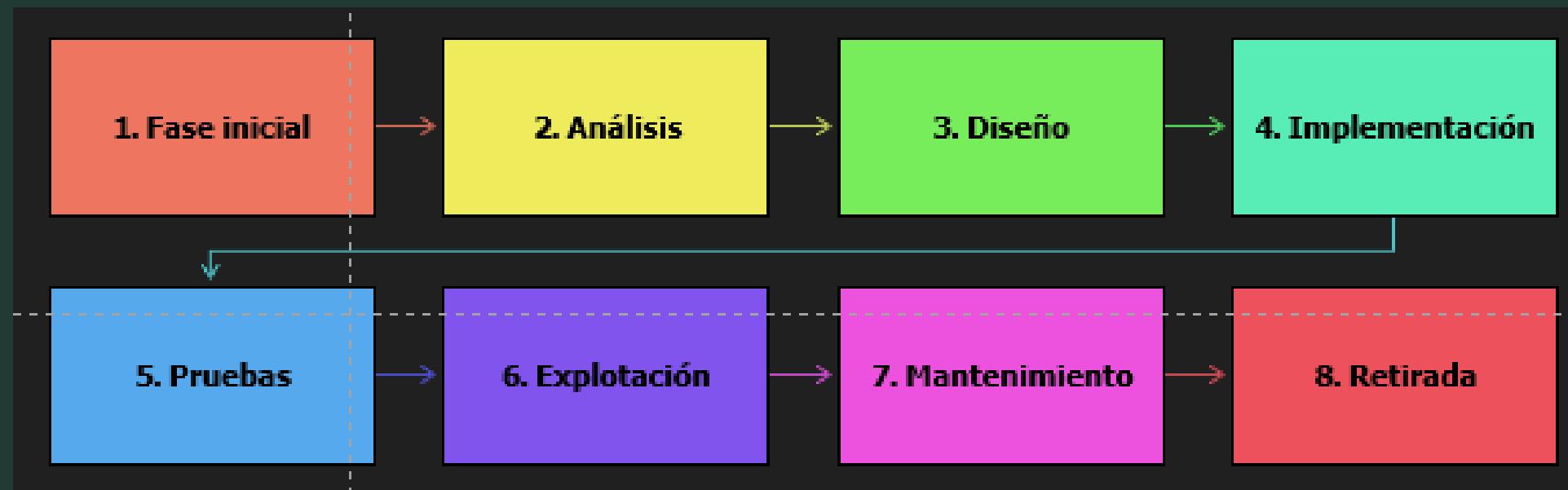
- Conoce en profundidad el lenguaje de programación
- Se encarga de codificar las tareas encomendadas por el analista o el analista programador
- Su misión es la de codificar y probar los diferentes módulos de la aplicación

## 6.4. PARADIGMAS DE DESARROLLO CLÁSICOS

- Los paradigmas de desarrollo clásicos son inflexibles
- Cada etapa debe finalizar para pasar a la siguiente
- ¿Qué pasa si detectamos un error grave de diseño durante la implementación?
- El paradigma más clásico es el modelo en cascada
- En esta asignatura trabajaremos con metodologías ágiles (SCRUM)

## 6.4. PARADIGMAS DE DESARROLLO CLÁSICOS

- El modelo en cascada es muy inflexible ya que cualquier error en una fase avanzada implica un sobrecoste y desperdicios.



## 6.5. METODOLOGÍAS ÁGILES

- En esta asignatura usaremos metodologías ágiles
- Estas metodologías responden mejor ante un cambio de especificaciones o un error
- Las metodologías ágiles se basan en el manifiesto ágil, que valora:
  - A los individuos y su interacción, por encima de los procesos y las herramientas*
  - Al software que funciona, por encima de la documentación exhaustiva*
  - A la colaboración con el cliente, por encima de la negociación contractual*
  - A la respuesta al cambio, por encima del seguimiento de un plan*

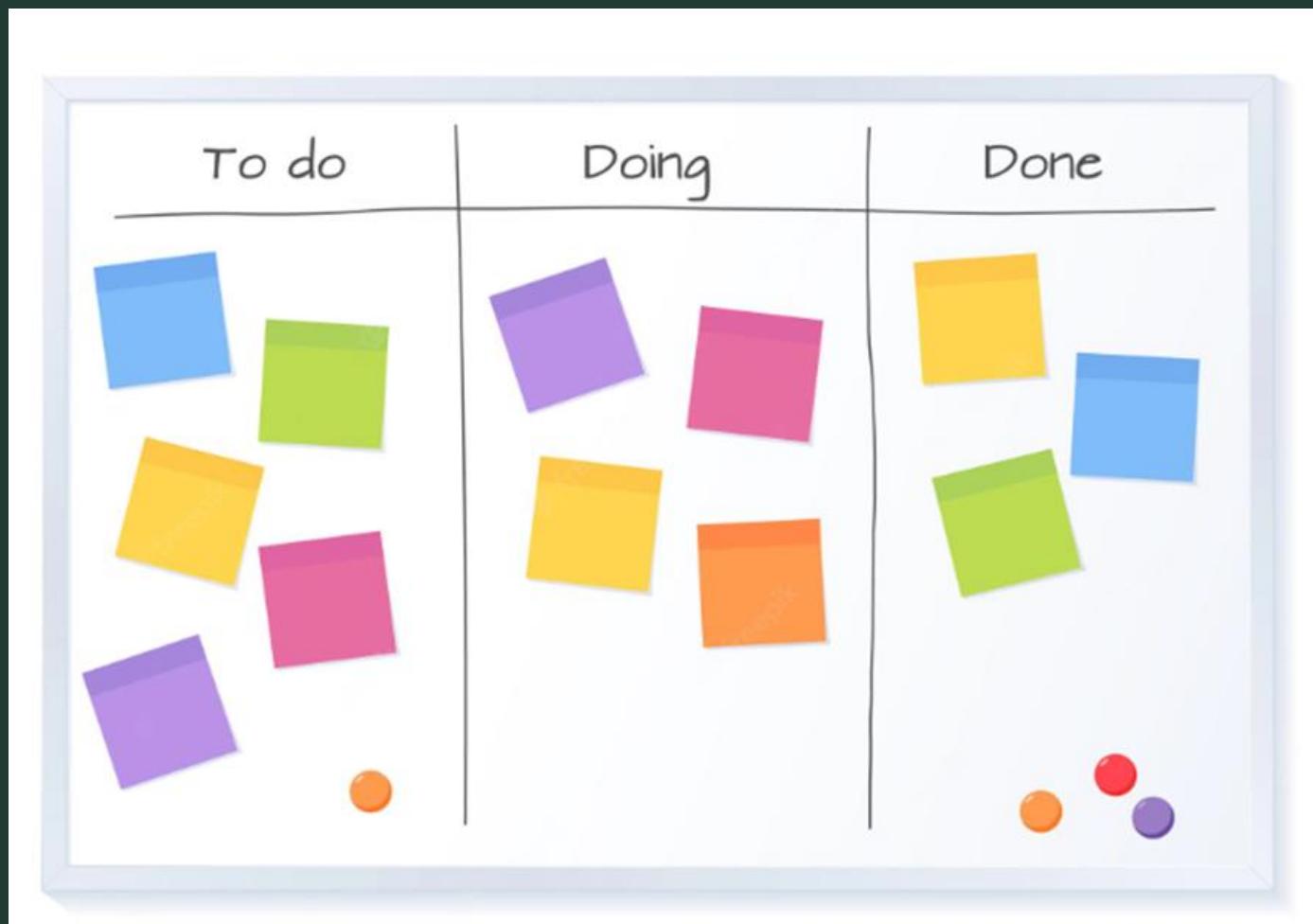
## 6.5. METODOLOGÍAS ÁGILES

Vamos a ver algunas de las metodologías más conocidas:

### KANBAN

- También se denomina "sistema de tarjetas".
- Desarrollado inicialmente por Toyota para la industria de fabricación de productos.
- Controla por demanda la fabricación de los productos necesarios en la cantidad y tiempo necesarios.
- Enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos.
- Kanban en desarrollo software. Pizarra kanban.

## 6.5 METODOLOGÍAS ÁGILES



## 6.5. METODOLOGÍAS ÁGILES

### *SCRUM*

- *Modelo de desarrollo incremental.*
- *Iteraciones (sprints) regulares cada 2 a 4 semanas.*
- *Al principio de cada iteración se establecen sus objetivos priorizados (sprint backlog).*
- *Al finalizar cada iteración se obtiene una entrega parcial utilizable por el cliente.*
- *Existen reuniones diarias para tratar la marcha del sprint.*

## 6.5. METODOLOGÍAS ÁGILES



## 6.6 SCRUM

### ¿Qué es SCRUM?

- El término SCRUM significa melé (jugada inicial de rugby)
- SCRUM es un marco de trabajo ágil basado en la iteración y entrega incrementales
- Se basa en el manifiesto ágil , la cohesión, gestión del equipo y la comunicación
- Supone una alternativa a las metodologías tradicionales de desarrollo (modelo en cascada)

## 6.6 SCRUM

### OBJETIVOS DE SCRUM

- Aportar el máximo valor al cliente
- Reducir el "time to market" (tiempo que tarda en llegar el producto al mercado)
- Reducir el tiempo de respuesta ante un cambio (aumentar la flexibilidad)
- Fiabilidad del producto terminado (el producto o incremento debe funcionar)

## 6.6 SCRUM

### EL MANIFIESTO ÁGIL

- En 2001 se reúne un comité de expertos para analizar las técnicas de desarrollo tradicionales
- Este comité crea el manifiesto ágil , cuyas premisas son valorar:
  - A los individuos y su interacción, por encima de los procesos y las herramientas
  - Al software que funciona, por encima de la documentación exhaustiva
  - A la colaboración con el cliente, por encima de la negociación contractual
  - A la respuesta al cambio, por encima del seguimiento de un plan

## 6.6 SCRUM

### CICLO DE TRABAJO

El ciclo de trabajo puede resumirse en cuatro etapas diferenciadas:

1. Se toman los requisitos y para cada requisito se crea un bloque de trabajo (historia de cliente)
2. El cliente ordena los bloques de trabajo en una pila de producto , según su prioridad de entrega
3. El equipo de trabajo toma un grupo de historias y trabaja con ellas durante una iteración (sprint)
- 4.Cuando se finaliza el sprint se entrega al cliente el resultado

## 6.6 SCRUM

### CICLO DE TRABAJO

- El ciclo de trabajo continúa mientras haya trabajo en la pila de producto
- En esencia el desarrollo se inicia con dos premisas iniciales: tiempo y coste
- Si se supera alguna de estas, se entrega el resultado final (siempre guiado por el cliente)
- Este marco de trabajo permite controlar los costes y el tiempo de desarrollo al máximo

## 6.6 SCRUM

### ROLES

Durante el desarrollo en SCRUM encontramos los siguientes roles:

1. **Cliente** : es el cliente del producto (el que espera recibir el producto o servicio)
2. **Dueño del producto (product owner)**: representa al cliente final del producto
3. **SCRUM master** : es el líder del equipo SCRUM, lidera el proyecto y gestiona al equipo
4. **Equipo SCRUM** : se encargan de llevar a cabo el proyecto SCRUM

## 6.6 SCRUM

### ARTEFACTOS

Podemos obtener los siguientes artefactos:

1. Pila de producto (product backlog): contiene las historias de cliente pendientes de desarrollar
2. Dueño del producto (product owner): contiene las historias que se desarrollarán en el sprint
3. Gráfica de avance (burn down ): representa el avance (en tanto %) del proyecto
4. Incremento : es la entrega que se obtiene tras un sprint (el incremento debe ser viable!)

## 6.6 SCRUM

### REUNIONES

En SCRUM se realizan las siguientes reuniones:

1. Planificación del sprint : reunión de planificación previa a la ejecución del sprint
2. Reunión diaria : reunión de progreso diario (se identifican problemas y se mide el avance)
3. Revisión del sprint : reunión de revisión del avance del sprint (suele ser semanal)
4. Retrospectiva del sprint : reunión de revisión final del sprint (se realiza al finalizar el sprint)

## 6.6 SCRUM

### ESTIMACIONES

- Para estimar el esfuerzo se usa un póker de planificación (sucesión de Fibonacci)
- El póker permite medir el esfuerzo de realizar un punto de la historia del cliente
- Cada integrante del equipo tiene una baraja de planificación
- Usando la baraja se asigna el esfuerzo estimado de cada tarea
- Cuanta más experiencia tiene el equipo, mejor será la estimación

## 6.6 SCRUM

### SCRUM Board

- Es un tablero que permite medir el avance de las historias del cliente que han finalizado
- Se utilizan colores, pegatinas, incluso fotos de los participantes
- Suele contener cuatro estados:

*Pendiente: trabajo pendiente para este sprint (o para el siguiente)*

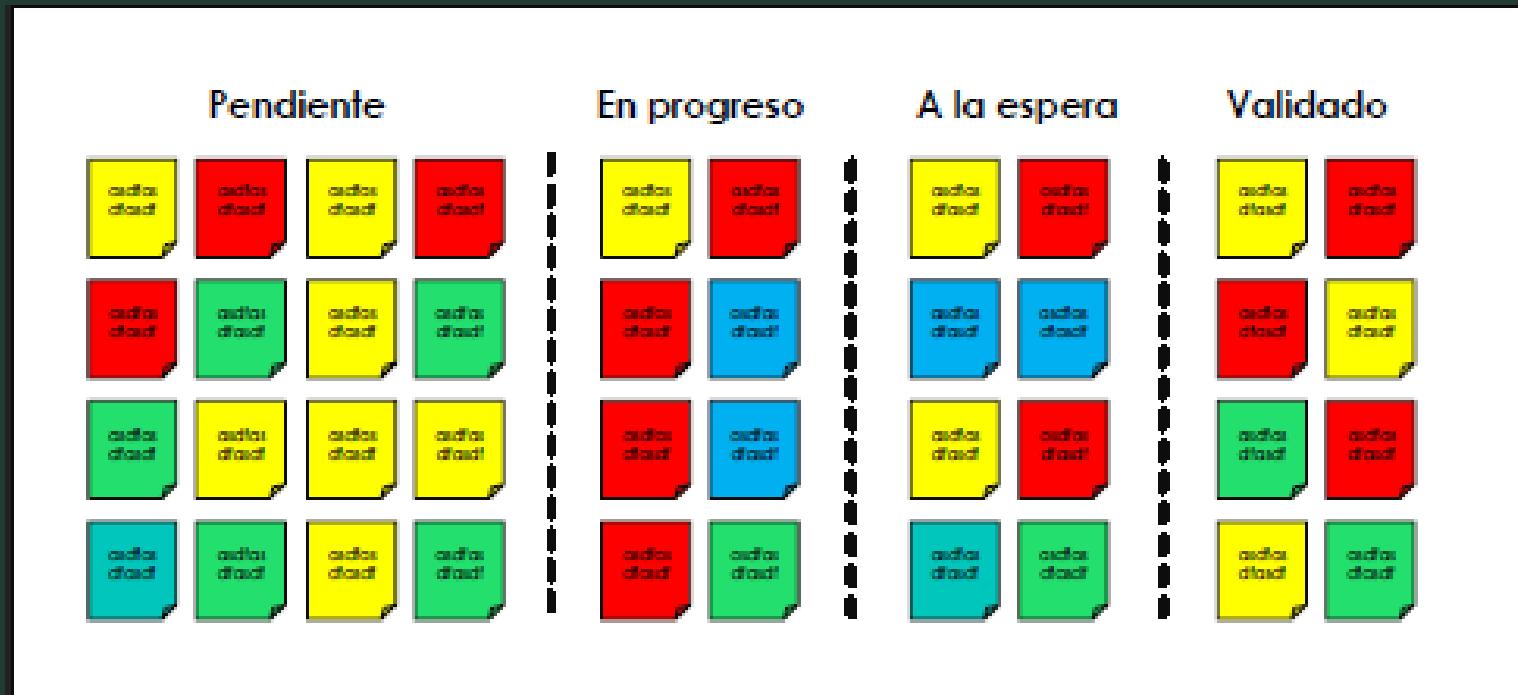
*En progreso: trabajo en progreso en este sprint.*

*A la espera: historias que deben ser validadas por el cliente.*

*Validado: historias que ya han sido validadas por el cliente*

## 6.6 SCRUM

### SCRUM Board



## 6.6. SCRUM

### EL CONTRATO ÁGIL

Se debe lograr un acuerdo que contenga:

- *Un backlog de producto bien definido*
- *Definición de “completado” consensuada*
- *Demos de producto y feedback continuo*
- *Cumplir el triángulo de hierro*



## 6.6 SCRUM

Clausulas más comunes sobre los cambios de requisitos:

- La nueva priorización no debe implicar cambios
- Los nuevos requisitos (o modificación de requisitos) no implican cambios si quitamos otros requisitos
- Los nuevos requisitos pueden sustituirse por otros con el mismo esfuerzo (en horas o coste)
- La resolución de errores por parte del desarrollador no se considera un cambio

## 6.6. SCRUM

Cláusulas más comunes sobre la finalización anticipada:

- El cliente puede finalizar en cualquier momento (previo abono del 20% de las horas restantes)
- Los requisitos aprobados cuya entrega fuese anterior a la finalización serán entregados sin coste
- El desarrollador se compromete a entregar el 80% de los requisitos en la fecha de entrega

## 6.6 SCRUM

### EJEMPLO

Caso práctico:

- Un cliente se pone en contacto con una fábrica de animales robóticos
- El cliente quiere un pato robótico para su estanque artificial

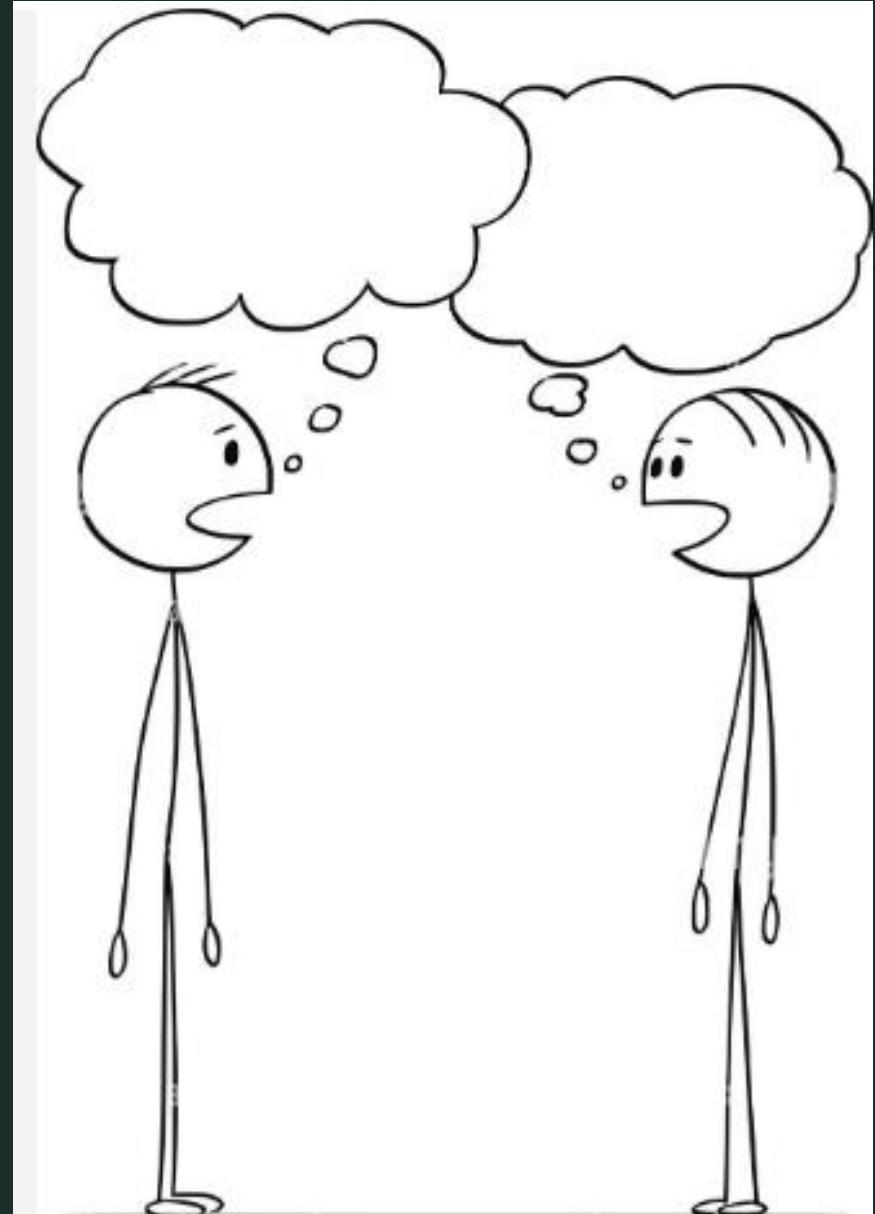
## 6.6. SCRUM

---

### EJEMPLO PRÁCTICO

1º) El cliente se reúne con el dueño del producto

El cliente indica al dueño del producto sus necesidades, es decir, que quiere un pato robótico.



## 6.6 SCRUM

### EJEMPLO PRÁCTICO

2º) El dueño del producto divide el proyecto en historias

Estas historias componen la pila del producto



## 6.6 SCRUM

### EJEMPLO PRÁCTICO

3º) El dueño del producto entrega la pila del producto al SCRUM Master

El SCRUM Master realiza una estimación del coste con el equipo

## 6.6 SCRUM

### EJEMPLO PRÁCTICO



4º) El equipo se reúne para estimar el coste de cada historia del cliente

Se emplea el poker de planificación



## 6.6 SCRUM

### EJEMPLO PRÁCTICO

5º) Una vez aprobado el presupuesto, el cliente reordena la pila de producto

La pila se reordena según la necesidad del cliente



## 6.6 SCRUM

### EJEMPLO PRÁCTICO

6º) El equipo comienza su trabajo

Se divide cada historia de la pila del producto en tareas menores y comienza el sprint



## 6.6 SCRUM

### EJEMPLO PRÁCTICO

7º) Al final de cada sprint...

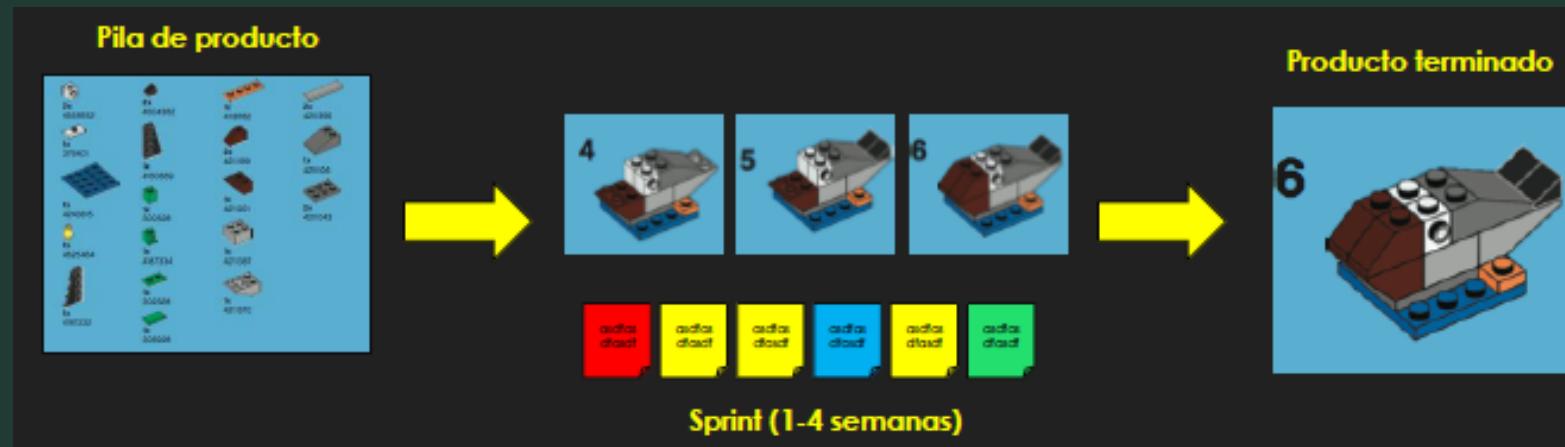
- El cliente tiene una toma de contacto con el producto terminado (incremento)
- Se prioriza de nuevo la pila de producto antes del siguiente sprint
- El cliente puede valorar el avance del proyecto
- Se mide el coste (¡siempre se mide!) en horas, tareas finalizadas, horas...

## 6.6 SCRUM

### EJEMPLO PRÁCTICO

8º) Los sprints se suceden en el tiempo

Se realizan reuniones, retrospectivas, nuevas planificaciones....



## 6.6 SCRUM

### EJEMPLO PRÁCTICO

9º) Finalmente

- El cliente recibe el producto final
- El cliente ha monitorizado cada avance y cada incremento
- Los costes se han controlado durante todo el proyecto
- El cliente está satisfecho

