

# UD1: Desarrollo del software

Entornos de desarrollo  
M<sup>a</sup> Carmen Safont Richarte

# INDICE

## 1. Introducción

- 1.1. ¿Qué es un ordenador?
- 1.2. ¿Qué es el software?
- 1.3. ¿Qué es una aplicación informática?

## 2. Lenguajes de programación

- 2.1. ¿Qué es un lenguaje de programación
- 2.2. Tipos de lenguajes de programación
- 2.3. Paradigmas de programación
- 2.4. Generación de código: proceso de traducción/compilación

## 3. Desarrollo del software

- 3.1. Ciclo de vida de una aplicación
- 3.2. Documentación
- 3.3. Equipos o roles de trabajo
- 3.4. Paradigmas del desarrollo clásicos
- 3.5. Metodologías ágiles



```
31 self.file = None
32 self.fingerprints = set()
33 self.logdupes = True
34 self.debug = debug
35 self.logger = logging.getLogger(__name__)
36
37 if path:
38     self.file = open(os.path.join(path, 'requests.log'),
39                     'a')
40     self.fingerprints.update([fp])
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('debug', False)
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)
```

# 1. INTRODUCCIÓN

## CONTENIDOS

- 1.1. ¿Qué es un ordenador?
  - 1.1.1. Evolución
- 1.2. ¿Qué es el software?
  - 1.2.1. Tipos de software
- 1.3. ¿Qué es una aplicación informática?

# 1. INTRODUCCIÓN

## 1.1. ¿Qué es un ordenador?

Los primeros ordenadores aparecen a partir de la década de los 40.

Un ordenador acepta datos de entrada, los procesa y produce unas salidas (resultados).

Se componen de elementos físicos (hardware) y lógicos (programas, software).

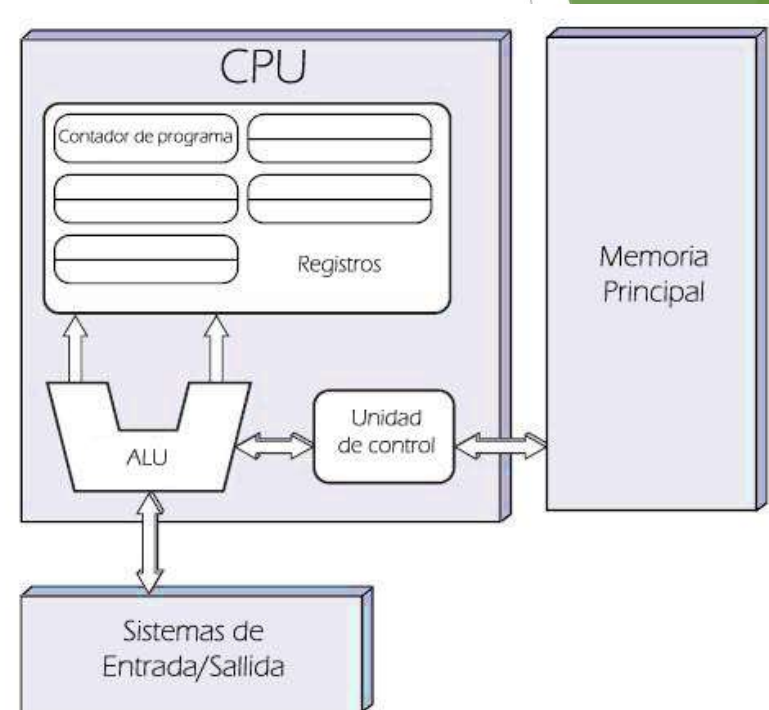
Las órdenes de los usuarios deben ser traducidas para que las entienda el ordenador.



# 1. INTRODUCCIÓN

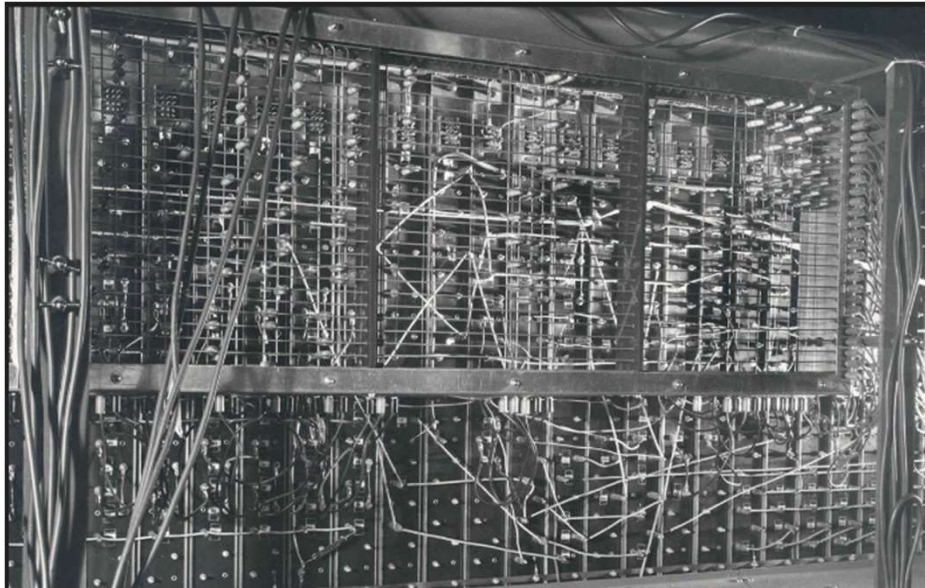
## Arquitectura Von Neumann

- ▶ Sistemas de entrada/salida: para recibir y comunicar información
- ▶ Memoria Principal: almacenan las instrucciones a ejecutar y los datos
- ▶ Unidad de Control: decodifica las instrucciones y su orden de ejecución
- ▶ ALU: Ejecuta las instrucciones

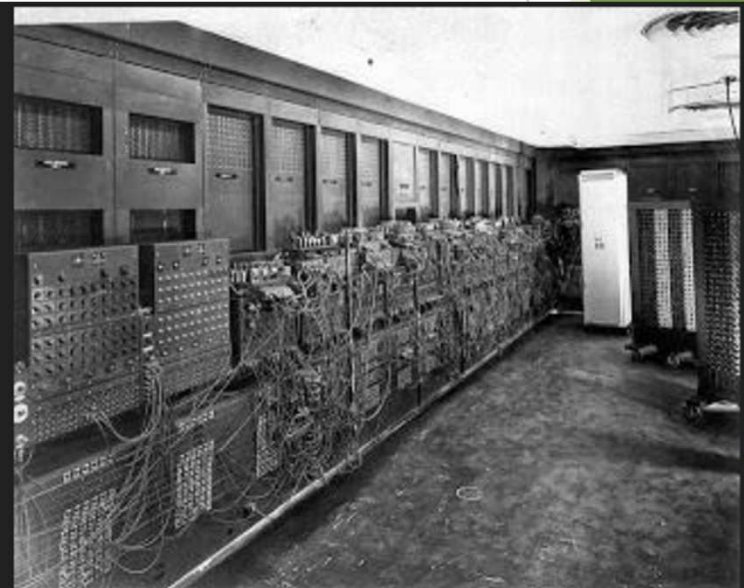


# 1. INTRODUCCIÓN

## 1.1.1. Evolución



Computadora de Alan Turing (1940s)  
Fuente: [Autodesk](#)



ENIAC (1940s)  
Fuente: [Helisulbaran](#)



# 1. INTRODUCCIÓN

## 1.1.1. Evolución



IBM Mainframe 701 (1950s)  
Fuente: [Time.graphics](#)



IBM PC (1980s)  
Fuente: [Wikipedia](#)

# 1. INTRODUCCIÓN

## 1.1.1. Evolución



IBM Mainframe 701 (1990s)  
Fuente: [NeXTstation](#)



HP Pavilion x2 (2010-2015)  
Fuente: [El País](#)



# 1. INTRODUCCIÓN

## 1.2. ¿Qué es el software?

### Definición formal:

*Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación.*

### Definición coloquial:

*Entendemos por Software al conjunto de **componentes lógicos** de un ordenador que permiten la realización de tareas específicas.*

# 1. INTRODUCCIÓN

## 1.2. ¿Qué es el software?

- ▶ Es la parte **intangible** (o lógica) de un **sistema informático**
- ▶ Se **desarrolla** para llevar a cabo una **tarea determinada**
- ▶ Se **comunica** con el **hardware** y le dice **qué tiene que hacer**
- ▶ Se encarga de **traducir** las **instrucciones de los usuarios**
- ▶ El término fue usado por **Charles Babbage** en el siglo XIX
- ▶ **Alan Turing** coge el relevo para descifrar la máquina **Enigma**

# 1. INTRODUCCIÓN

## Conceptos clave a tener en cuenta:

1. El software se **desarrolla**, no se fabrica.
2. El software es **lógico** (intangible), no es un elemento físico.
3. El software **no se estropea** (por lo general), y una copia produce un clon del software original.
4. El software puede desarrollarse a **medida** (aplicaciones desarrolladas a medida)
5. El software **posibilita el uso** del ordenador
6. El software se desarrolla usando un **lenguaje de programación** y una **metodología de programación**.

# 1. INTRODUCCIÓN

## 1.2.1. Tipos de software

- ▶ Software de Sistema: SSOO, controladores, servidores, herramientas de diagnóstico, etc.
- ▶ Software de programación: editores de texto, compiladores, intérpretes, depuradores, IDEs, etc.
- ▶ Software de aplicación: aplicaciones ofimáticas, bases de datos, videojuegos, software empresarial, software de diseño, etc.

# 1. INTRODUCCIÓN

## 1.2.1. Tipos de software

El SO actúa de intermediario, coordinando el HW y las aplicaciones en funcionamiento.

Las aplicaciones consumen recursos (hardware): Tiempo de CPU para su ejecución, espacio de almacenamiento en memoria RAM, gestión de interrupciones, gestión de dispositivos de entrada/salida.

Las aplicaciones están escritas en lenguaje humano pero los ordenadores solo saben interpretar señales eléctricas o lógicas (0 y 1). Por tanto, es necesario un proceso de traducción:



# 1. INTRODUCCIÓN

## 1.2.2. Tipos de código

### Código Fuente

- ▶ Es el conjunto de sentencias (entendibles por el programador) que indica los pasos (instrucciones) a seguir por un ordenador para realizar una tarea (programa o una parte del mismo).
- ▶ Está escrito en un lenguaje de programación específico, elegido por el programador (C, C++, C#, Java, Perl, Python, PHP, ...).
- ▶ NO es directamente ejecutable por el ordenador DEBE ser traducido (compilado) a lenguaje máquina o código objeto.
- ▶ Suele estar almacenado en un fichero de texto por lo que se puede abrir con cualquier editor de texto.



# 1. INTRODUCCIÓN

## 1.2.2. Tipos de código

### Código Objeto

- ▶ Conjunto de instrucciones y datos escritos en un lenguaje que entiende directamente el ordenador (pero no por el ser humano)
- ▶ Puede estar escrito en lenguaje máquina o bytecode.
- ▶ Es el código resultado de una compilación/interpretación del código fuente.
- ▶ Puede estar distribuido en diversos archivos.
- ▶ Un enlazador (Linker) se encarga de juntar el código objeto y obtener el programa ejecutable

# 1. INTRODUCCIÓN

## 1.2.2. Tipos de código

### Código Ejecutable

- ▶ Es el conjunto de instrucciones compiladas y enlazadas, listas para ser ejecutadas por una máquina (código máquina).
- ▶ Es el código que ejecutan los usuarios del sistema y es específico para una plataforma concreta: Windows, Linux, MacOS, ...
- ▶ El SO será el encargado de cargar este código en la memoria RAM y proceder a ejecutarlo.

# 1. INTRODUCCIÓN

## 1.3. ¿Qué es una aplicación informática/programa?

- ▶ Un **programa**, o **aplicación informática**, se compone de un **conjunto de instrucciones**
- ▶ Una **aplicación** se **desarrolla** usando un **lenguaje de programación**
- ▶ Las **instrucciones** le indican al ordenador el **programa** o **pasos** que debe ejecutar
- ▶ Si el ordenador **no entiende** alguna instrucción, lo notificará mediante un **mensaje**
- ▶ En este ciclo pasaremos de ser **usuarios** a **programadores de aplicaciones**

# 1. INTRODUCCIÓN

## 1.3. ¿Qué es una aplicación informática/programa?

### Ejemplos:

1. Sistemas operativos (Windows, Linux, MacOS)
2. Aplicaciones de contabilidad y ofimática
3. Aplicaciones de gestión de bases de datos
4. Aplicaciones de diseño gráfico
5. Aplicaciones de correo electrónico
6. Sistemas de mensajería
7. Videojuegos

# 1. INTRODUCCIÓN

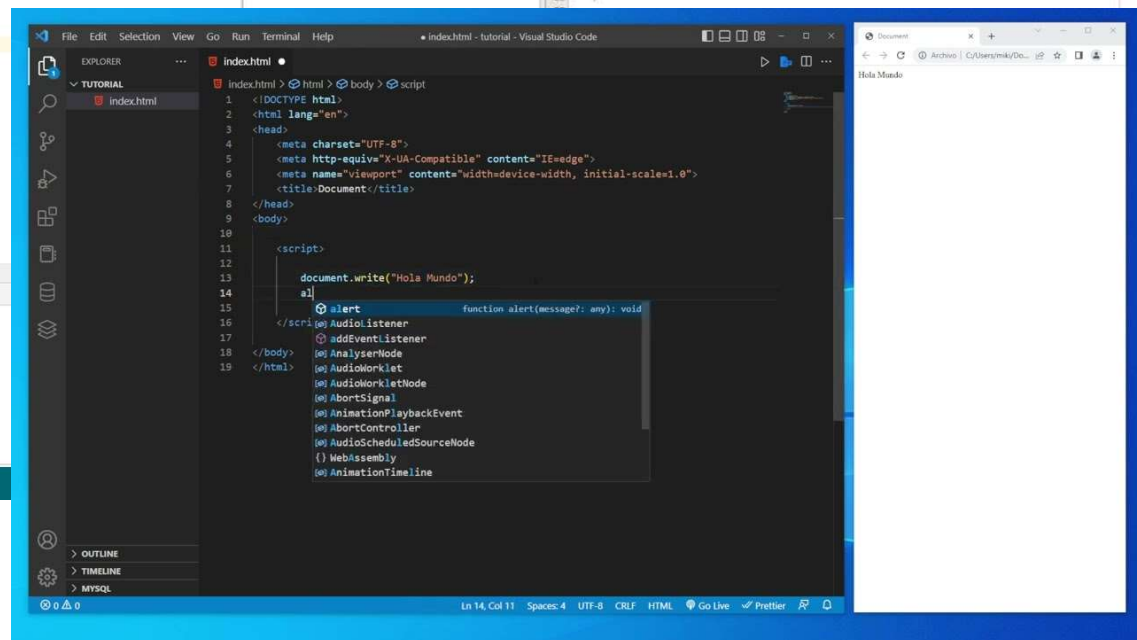
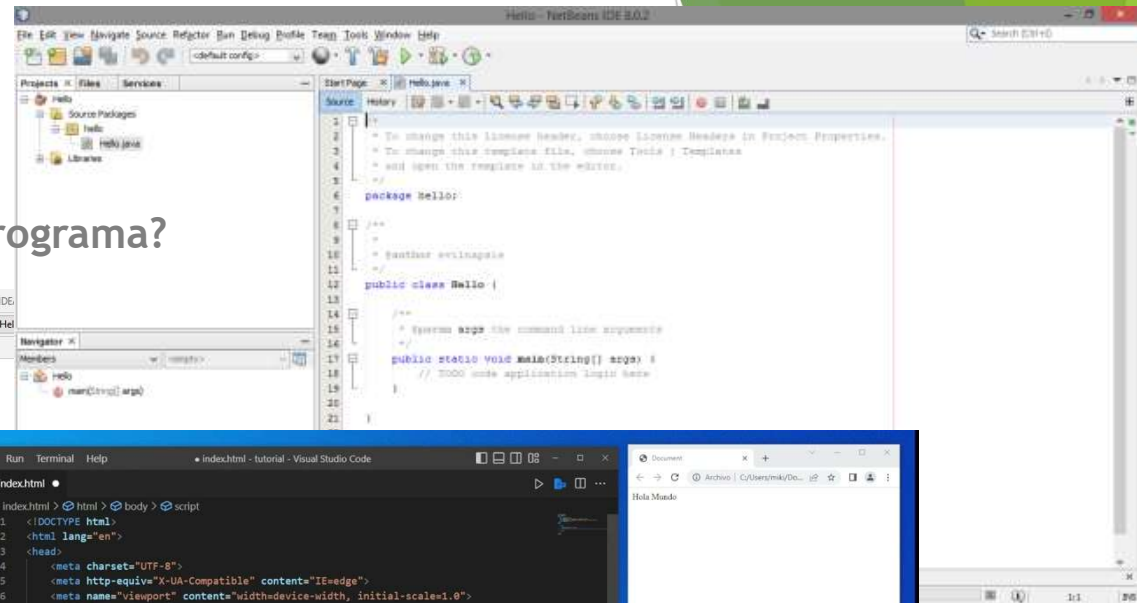
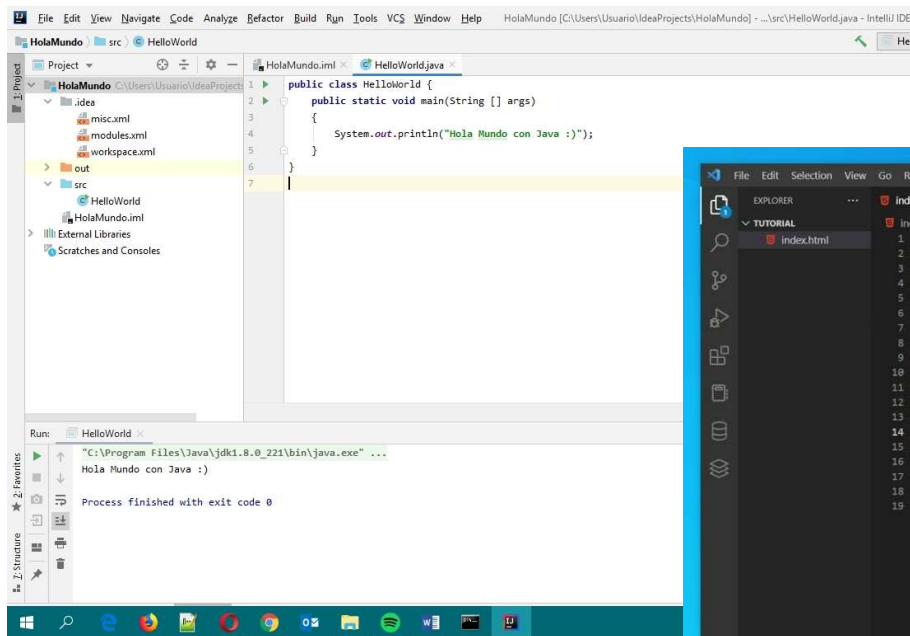
## 1.3. ¿Qué es una aplicación informática/programa?

### Conceptos importantes:

1. **Programa:** conjunto de pasos o instrucciones que indican al ordenador cómo realizar un proceso
2. **Ejecutar:** consiste en iniciar un programa y ponerlo en ejecución
3. **Librería:** conjunto de programas y funciones que realizan tareas concretas (BBDD, informes...)
4. **Entorno de desarrollo (IDE):** herramienta que facilita y posibilita el desarrollo de software

# 1. INTRODUCCIÓN

## 1.3. ¿Qué es una aplicación informática/programa?





## 2. LENGUAJES DE PROGRAMACIÓN

### CONTENIDOS

- 2.1. ¿Qué es un lenguaje de programación?
- 2.2. Tipos de lenguajes de programación
- 2.3. Paradigmas de programación
- 2.4. Generación de código: Proceso de traducción/compilación

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.1. ¿Qué es un lenguaje de programación?

#### Definición:

*Un lenguaje de programación es un **lenguaje formal** que proporciona al programador la capacidad de escribir una serie de instrucciones en forma de **algoritmos**, con el fin de controlar el comportamiento lógico de un sistema informático.*

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.1. ¿Qué es un lenguaje de programación?

#### Conceptos a tener en cuenta:

1. **Lenguaje máquina:** lenguaje que sólo es comprensible por el ordenador (01010101)
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

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.1. ¿Qué es un lenguaje de programación?

- ▶ Existen diferentes tipos de lenguajes de programación y diferentes formas de clasificarlos.
- ▶ 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...

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.2. Tipos de lenguajes de programación

#### Clasificación según el nivel



LENGUAJE MÁQUINA



LENGUAJE  
ENSAMBLADOR  
(o de bajo nivel)



LENGUAJE DE ALTO  
NIVEL

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.2. Tipos de lenguajes de programación

#### 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...



## 2. LENGUAJES DE PROGRAMACIÓN

### 2.2. Tipos de lenguajes de programación

#### LENGUAJE BAJO 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

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.2. Tipos de lenguajes de programación

#### LENGUAJE 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.

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.2. Tipos de lenguajes de programación

Ahora analizaremos las características de algunos de estos lenguajes

#### Lenguaje C/C++

```
using namespace std;

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

- Uno de los más potentes y utilizados, ligado a Unix y a Linux.
- Estructura y ligado 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.

#### Java

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

- 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

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.2. Tipos de lenguajes de programación

Ahora analizaremos las características de algunos de estos lenguajes

#### Python

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

- 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

#### JavaScript

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

- 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)

#### PHP

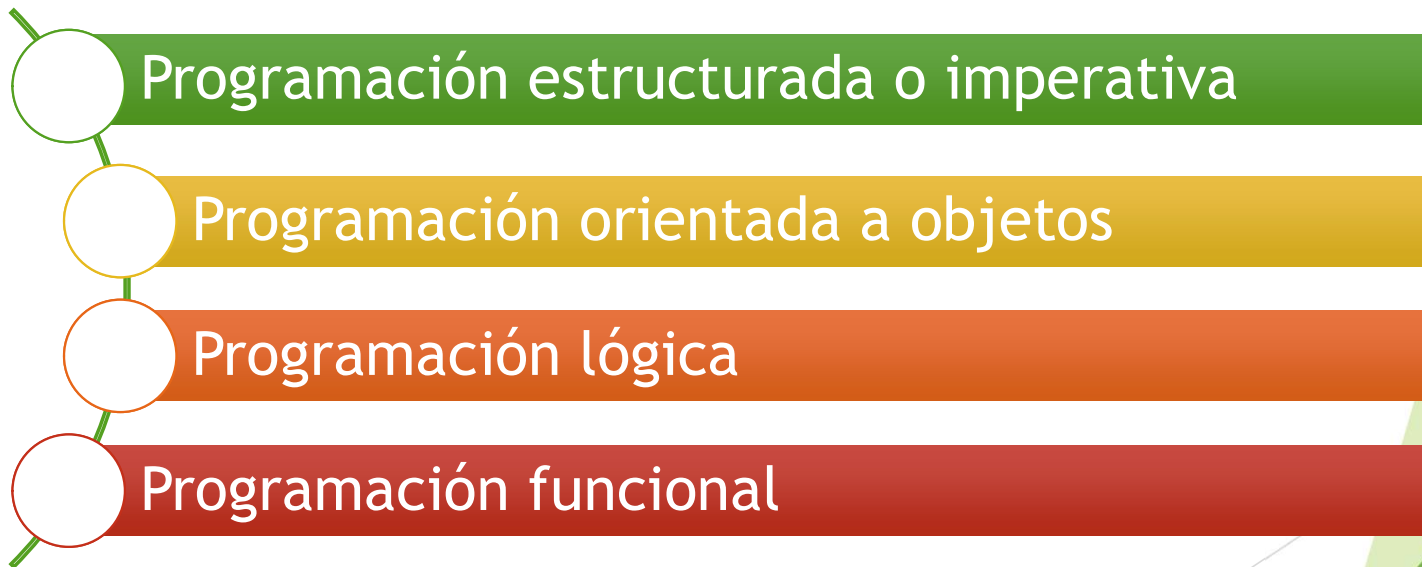
```
<?php  
echo '<p>Hola mundo</p>';  
?>
```

- 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
- Lenguaje interpretado.

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.3. 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



## 2. LENGUAJES DE PROGRAMACIÓN

### 2.3. Paradigmas de programación



#### Programación estructurada o imperativa

- 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)



## 2. LENGUAJES DE PROGRAMACIÓN

### 2.3. 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

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.3. 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**

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.3. 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.

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4. Generación de código: proceso de traducción/compilación

Como hemos mencionado, la tarea de programar implica pasar de un código fuente entendible por humanos a un código máquina.

Esta transformación se realiza mediante un traductor, que puede ser un **compilador** o un **intérprete**, según el lenguaje de programación empleado.

- ▶ En un lenguaje compilado, el código fuente es transformado a lenguaje máquina y posteriormente ejecutado en la máquina.
- ▶ En un lenguaje interpretado el código fuente es transformado a un lenguaje máquina a medida que es ejecutado.

Así mismo, en función de la forma de ejecutar un lenguaje existen los lenguajes compilados, interpretados o virtuales.

## 2. LENGUAJES DE PROGRAMACIÓN

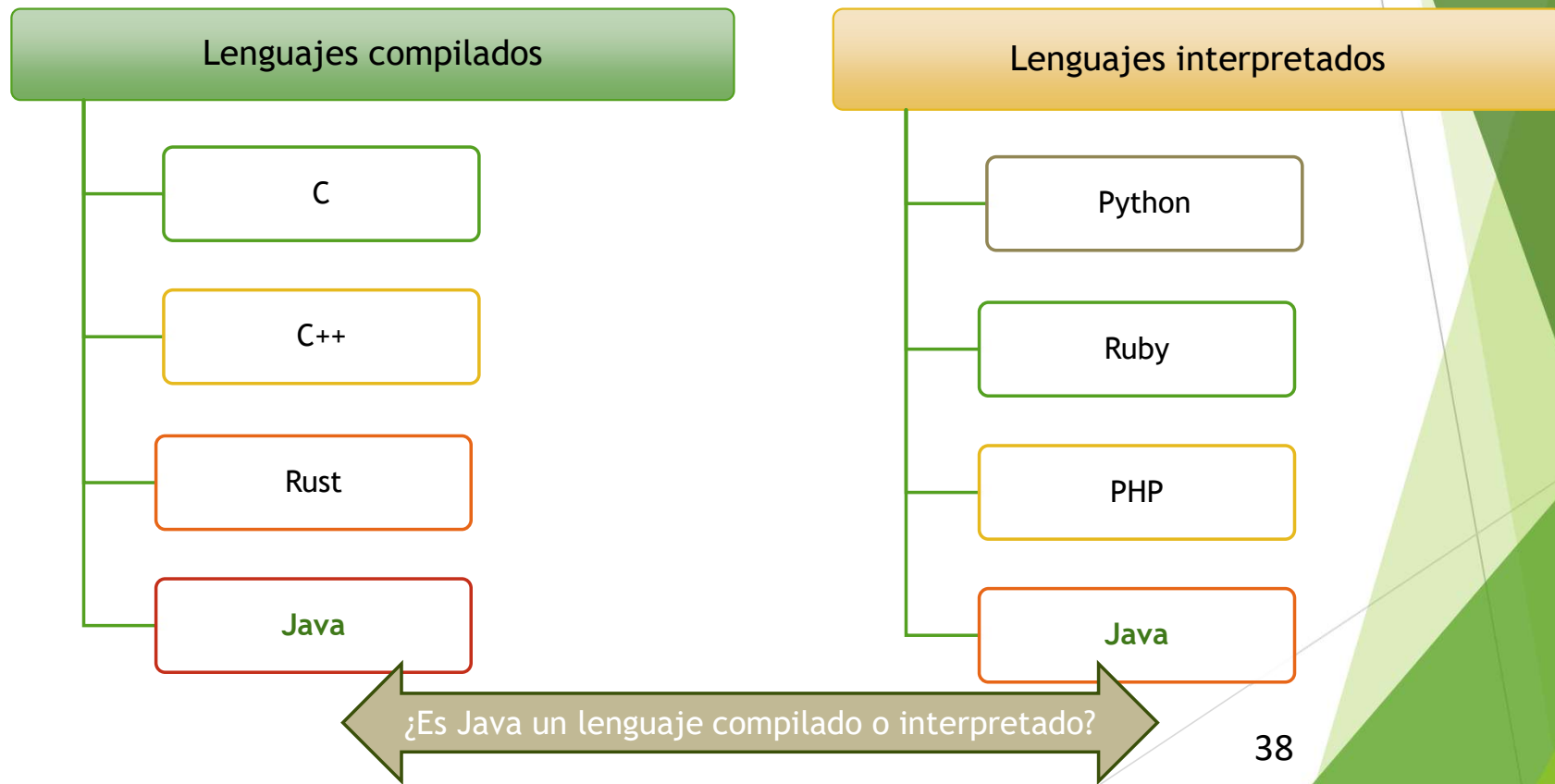
### 2.4. Generación de código: proceso de traducción/compilación

#### 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 (comprensible 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...

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4. Generación de código: proceso de traducción/compilación



## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4.1. 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++

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4.1. Lenguajes compilados

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:

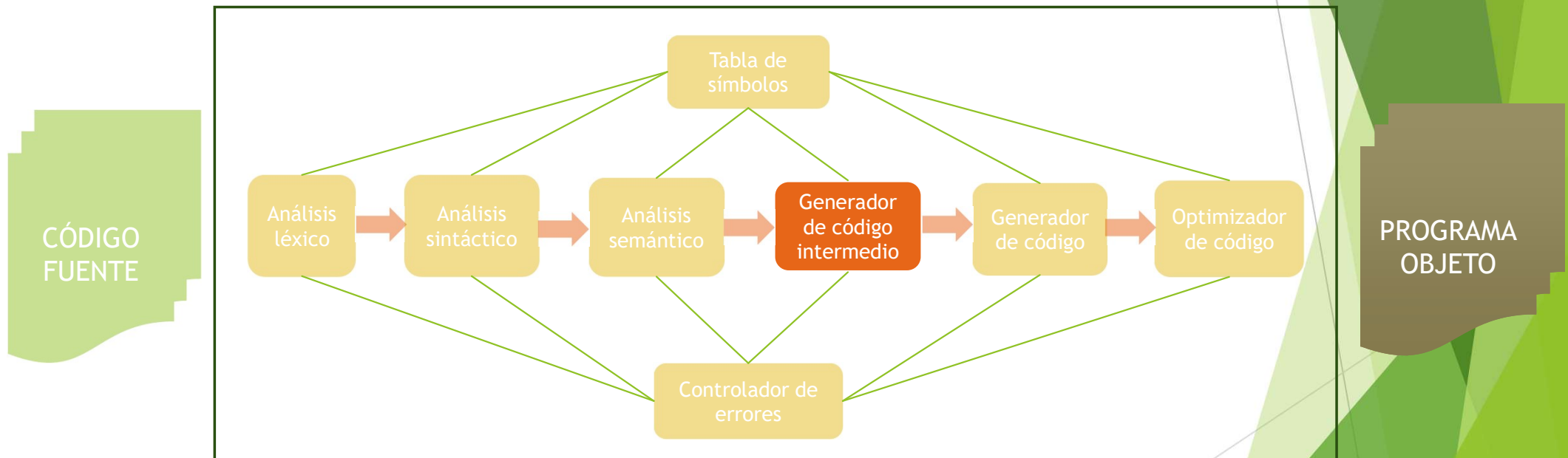
- **Preprocesado**: se comprueban el léxico, la sintaxis y la semántica del código.
- **Generación de código intermedio**: generación de código máquina
- **Enlazado**: se enlazan el código objeto con librerías externas





## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4.1. Lenguajes compilados



## 2. LENGUAJES DE PROGRAMACIÓN

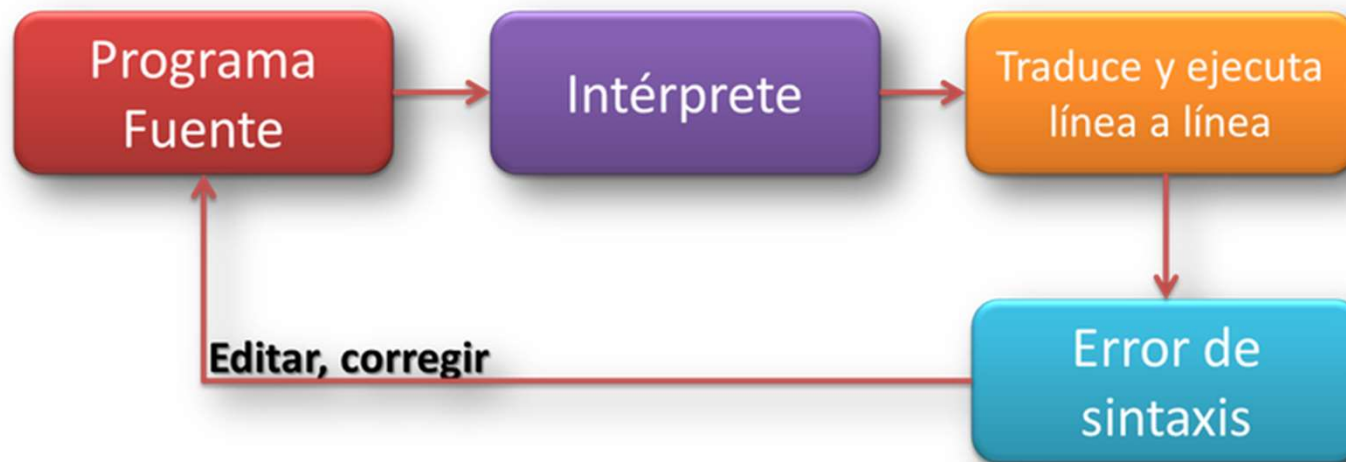
### 2.4.1. Lenguajes interpretados

- El intérprete tiene que estar en memoria ejecutándose para ejecutar el programa.
- El código fuente del programa también se carga en memoria para poder ser interpretado.
- Un intérprete **traduce** el código fuente **línea a línea**, sin generar código objeto, es decir, “interpreta” cada sentencia del programa y lo ejecuta.
- Ejemplo: PHP



## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4.1. Lenguajes interpretados



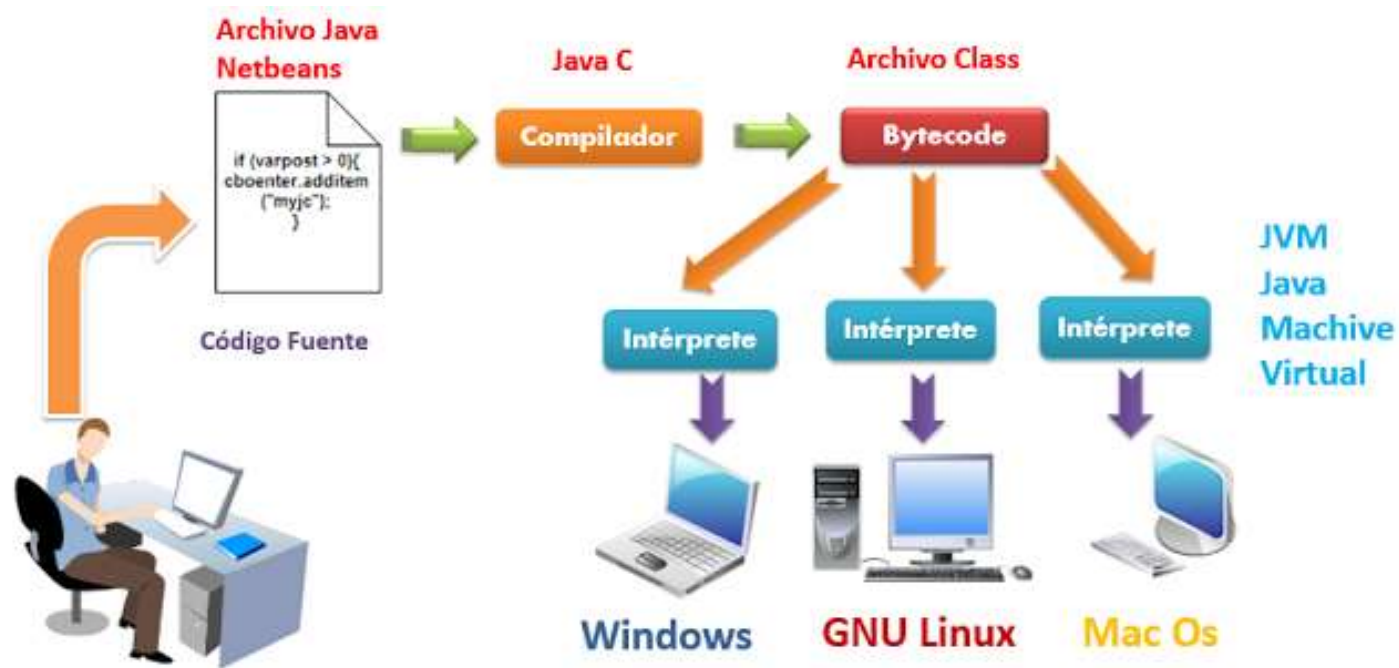
## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4.1. 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

## 2. LENGUAJES DE PROGRAMACIÓN

### 2.4.1. Lenguajes virtuales



# 3. INGENIERIA DEL SW

## CONTENIDOS

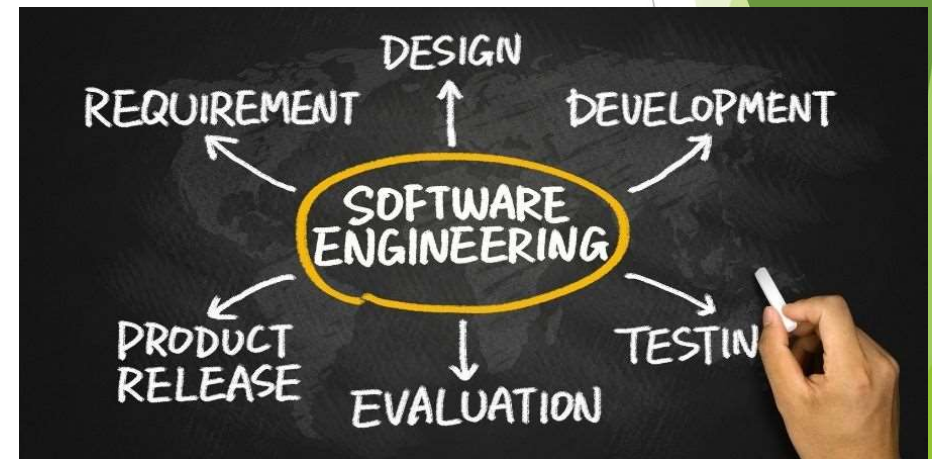
- 3.1. Ciclo de vida del desarrollo del SW
- 3.2. Documentación
- 3.3. Roles o equipos de trabajo
- 3.4. Ciclos de vida de desarrollo clásico
- 3.5. Metodologías ágiles

### 3. INGENIERIA DEL SW

La Ingeniería del Software es la rama de la ciencia de la computación que estudia la creación de software fiable y de calidad.

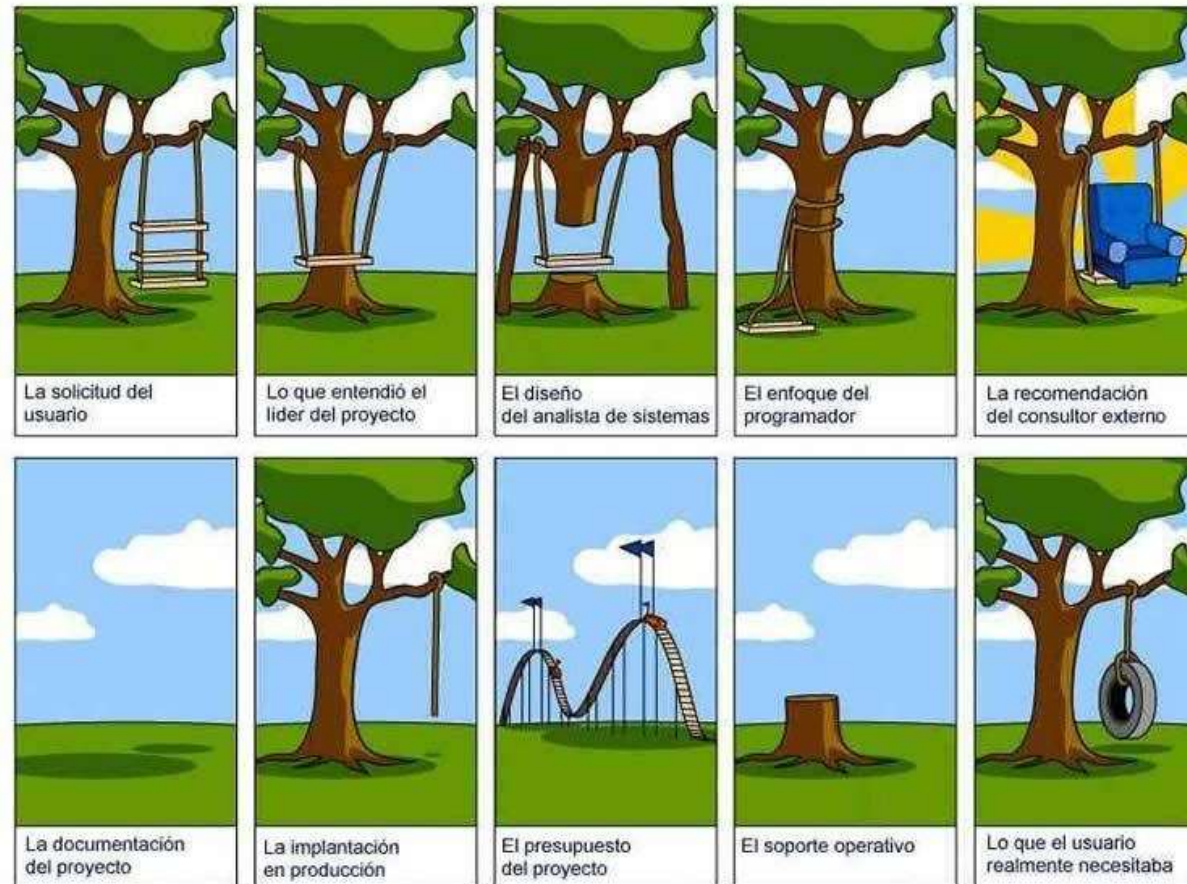
En el proceso de creación de un programa informático se pasa por diferentes etapas, desde que se presenta una idea y se planifica, hasta que llega al usuario final.

Es lo que se conoce como Ciclo de Vida del Desarrollo del Software.





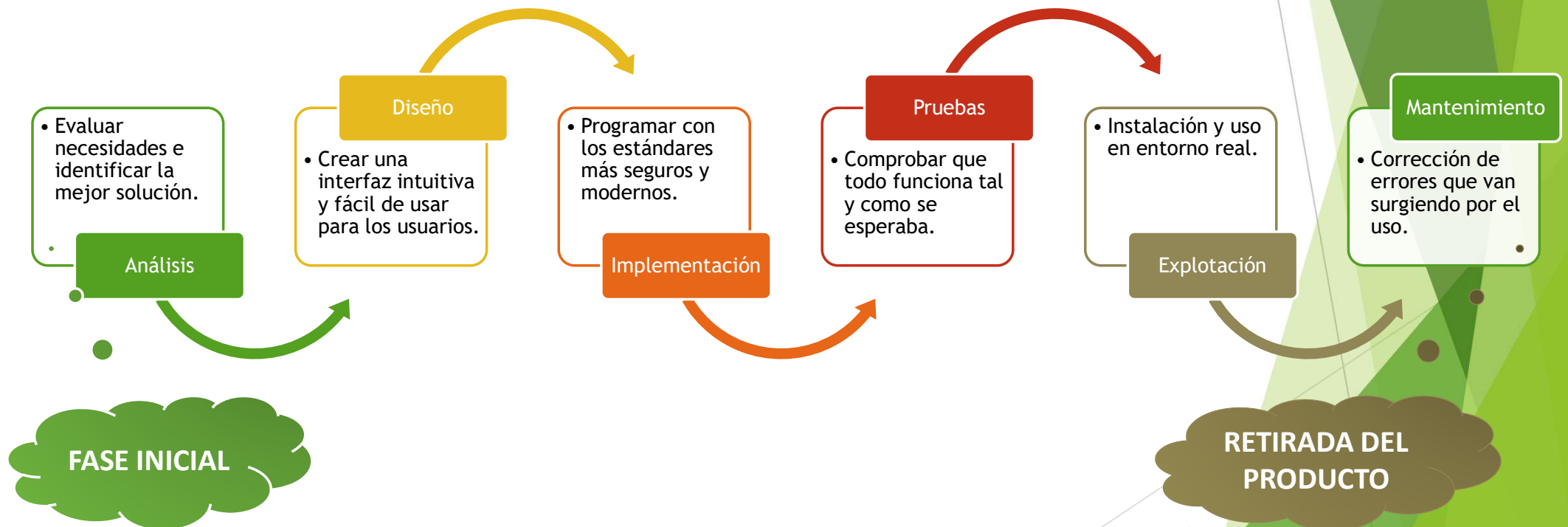
### 3. INGENIERIA DEL SW





# 3. INGENIERIA DEL SW

## 3.1. Ciclo de Vida del Desarrollo del Software



## 3. INGENIERIA DEL SW

### 3.1. Ciclo de Vida del Desarrollo del Software

#### 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...)



## 3. INGENIERIA DEL SW

### 3.1. Ciclo de Vida del Desarrollo del Software

#### Fase de Análisis de requisitos

- ▶ Consiste en analizar el problema: Recopilar, examinar y formular los requisitos del cliente (especificación de requisitos).
  - ▶ Estudiar las necesidades del cliente.
  - ▶ Definir las funcionalidades de la aplicación final.
  - ▶ Describir las interfaces de usuarios necesarias.
  - ▶ Definir los casos de uso.
- ▶ 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.

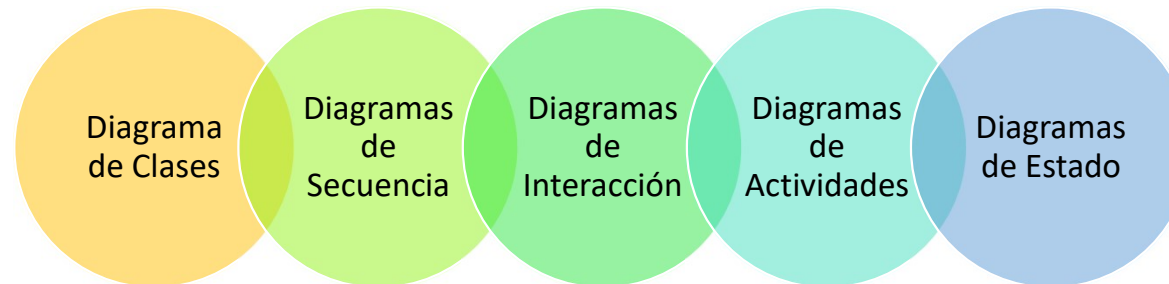


## 3. INGENIERIA DEL SW

### 3.1. Ciclo de Vida del Desarrollo del Software

#### Fase de Diseño

- ▶ Consiste en determinar los requisitos generales de la arquitectura de la aplicación: tecnologías (software y hardware) a utilizar, diseñar los componentes funcionales del sistema, describir el funcionamiento del sistema.
- ▶ Los documentos son mucho más técnicos (uno más genérico y otro más detallado):



- ▶ Esta fase involucra a los jefes de proyecto, arquitectos de software y analistas.
- ▶ Los programadores aún no intervienen en esta fase.

# 3. INGENIERIA DEL SW

## 3.1. Ciclo de Vida del Desarrollo del Software

### Fase de Codificación o implementación

- Consiste en **implementar** el software usando lenguajes de programación, librerías, frameworks...
- Debe cumplir los requisitos definidos en la fase de análisis y las funcionalidades y diagramas descritos en la fase de diseño.



- 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.

# 3. INGENIERIA DEL SW

## 3.1. Ciclo de Vida del Desarrollo del Software

### Fase de 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 la aplicación hace lo que tiene que hacer (con el cliente) → Documentar!!!
  - ▶ **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

## 3. INGENIERIA DEL SW

### 3.1. Ciclo de Vida del Desarrollo del Software

#### Fase de 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.
- ▶ También se anotan las nuevas funcionalidades detectadas.



## 3. INGENIERIA DEL SW

### 3.1. Ciclo de Vida del Desarrollo del Software

#### Fase de 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.
- ▶ Adaptación y evolución del software a las nuevas funcionalidades que vayan surgiendo con el uso (**mantenimiento continuo**).



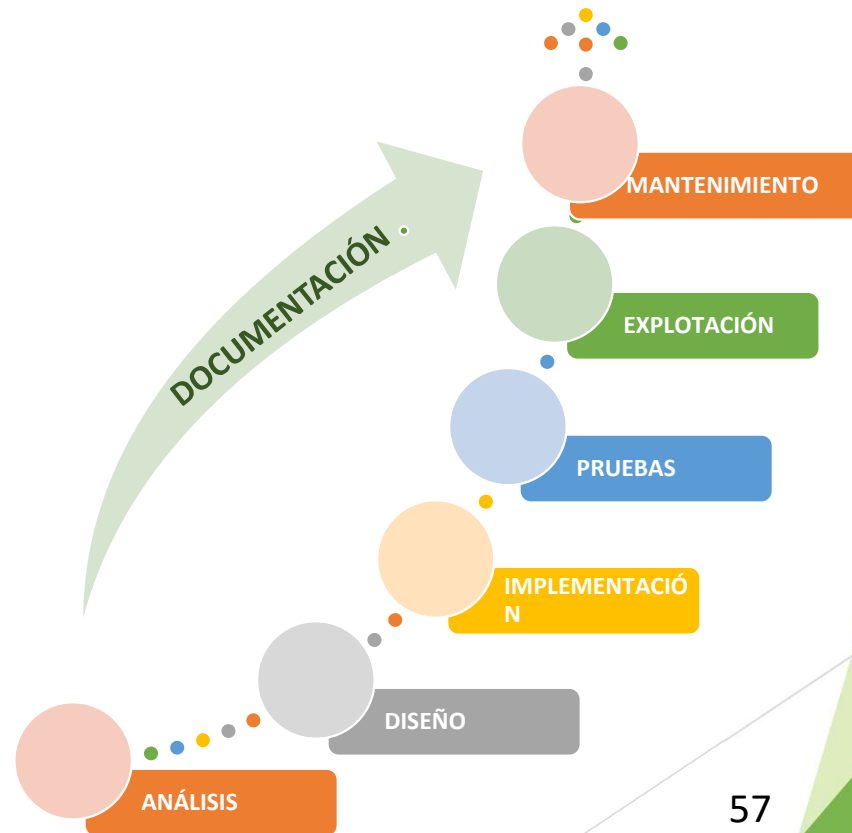


## 3. INGENIERIA DEL SW

### 3.1. Ciclo de Vida del Desarrollo del Software

El principio y el final de cada etapa no están claros. Todas las fases se solapan y, una vez que concluyen, el resultado es la aplicación.

La documentación, es un proceso transversal a lo largo de todo el proyecto.



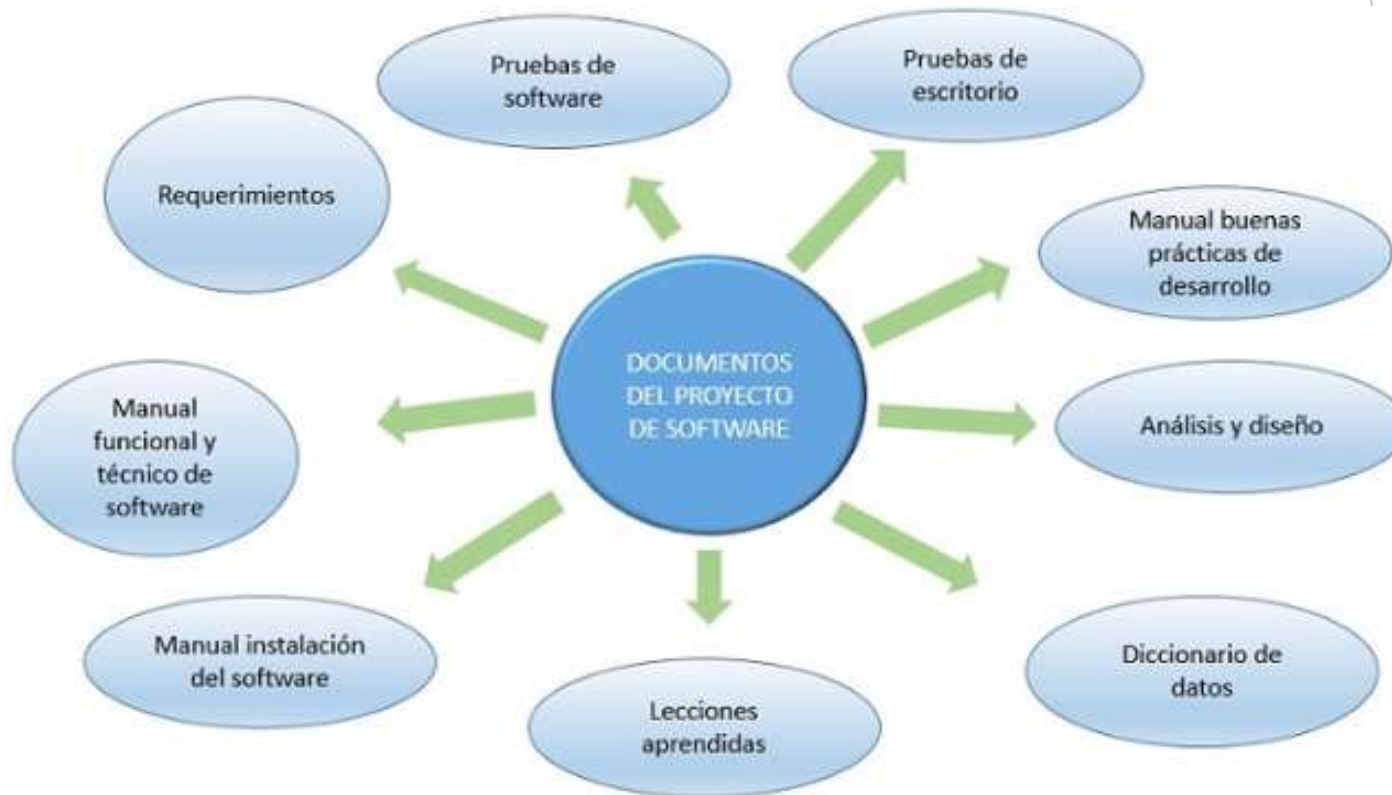
## 3. INGENIERIA DEL SW

### 3.2. Documentación

- ▶ Como hemos explicado en la sección anterior, la **documentación es vital.**
- ▶ En cada fase se generan uno o más documentos, por lo que la documentación debe llevarse a cabo a lo largo de toda la vida del proyecto
- ▶ Como mínimo, cada aplicación debe tener estos documentos:
  - ▶ **Manual de usuario:** explica cómo usará el usuario la aplicación
  - ▶ **Manual de instalación:** detalla el proceso de instalación de la aplicación
  - ▶ **Manual técnico:** documentación dirigida a los técnicos (administradores y programadores)

## 3. INGENIERIA DEL SW

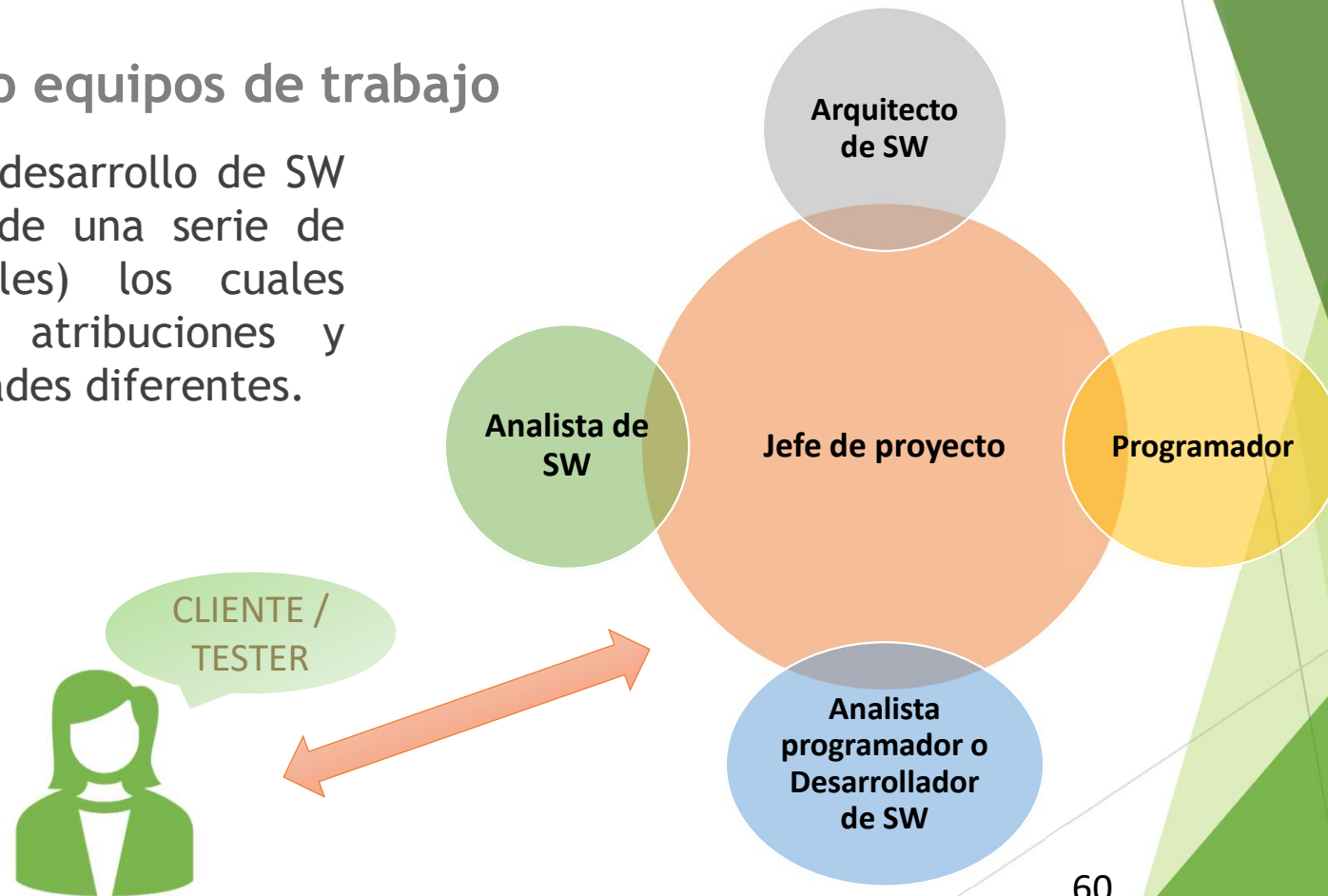
### 3.2. Documentación



## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo

El equipo de desarrollo de SW se compone de una serie de personas (roles) los cuales tienen unas atribuciones y responsabilidades diferentes.



## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo

#### Jefe de Proyecto

Funciones	Habilidades	Interviene	Formación
<ul style="list-style-type: none"><li>• Planificación del proyecto y definición de objetivos.</li><li>• Reporte de viabilidad.</li><li>• Administración de recursos (económicos y personales).</li><li>• Coordinación de equipos.</li><li>• Contacto continuo con el cliente.</li></ul>	<ul style="list-style-type: none"><li>• Experiencia en gestión de proyectos</li><li>• Experiencia en gestión de equipos.</li><li>• Capacidad de análisis.</li><li>• Buen comunicador.</li><li>• Liderazgo</li></ul>	<ul style="list-style-type: none"><li>• Fase inicial</li><li>• Análisis</li><li>• Diseño</li><li>• Implementación</li><li>• Pruebas</li><li>• Explotación</li><li>• Mantenimiento.</li><li>• <i>Documentación</i></li></ul>	<ul style="list-style-type: none"><li>• Ingeniero TIC.</li><li>• Certificación ITIL</li><li>• Certificación PMP</li><li>• Certificación MBA</li></ul>

## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo

#### Arquitecto SW

Funciones	Habilidades	Interviene	Formación
<ul style="list-style-type: none"><li>• Diseñar la arquitectura y cada componente del sistema.</li><li>• Selección de tecnología a usar.</li><li>• Seguimiento tras la implementación.</li><li>• Documentar los requisitos.</li><li>• Toma de decisiones.</li></ul>	<ul style="list-style-type: none"><li>• Conocimiento técnico:<ul style="list-style-type: none"><li>• UML</li><li>• Tecnologías</li><li>• Frameworks, ...</li></ul></li><li>• Liderazgo.</li><li>• Capacidad analítica y de organización.</li><li>• Dotes de facilitador y formador.</li></ul>	<ul style="list-style-type: none"><li>• Análisis</li><li>• Diseño</li><li>• Explotación</li><li>• <i>Documentación</i></li></ul>	<ul style="list-style-type: none"><li>• Grado de Ingeniería Informática o de Sistemas.</li></ul>

**ROL MÁS  
IMPORTANTE PERO  
MENOS CONOCIDO**

## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo

#### Analista de SW

Funciones	Habilidades	Interviene	Formación
<ul style="list-style-type: none"><li>• Estudio exhaustivo del problema</li><li>• Definir las especificaciones técnicas del producto.</li><li>• Traducción al equipo de Desarrollo.</li><li>• Optimización.</li></ul>	<ul style="list-style-type: none"><li>• Conocimientos en seguridad TI, Arquitectura de SW, BD, etc.</li><li>• Análisis y modelado de procesos y datos (UML).</li><li>• Capacidad de comunicación.</li><li>• Trabajo con el cliente.</li></ul>	<ul style="list-style-type: none"><li>• Análisis</li><li>• <i>Documentación</i></li></ul>	<ul style="list-style-type: none"><li>• Ingeniero TI.</li><li>• Experiencia previa en TI</li></ul>

## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo

#### Analista Programador o Desarrollador de SW

Funciones	Habilidades	Interviene	Formación
<ul style="list-style-type: none"><li>• Puesto a caballo entre el analista y el programador.</li><li>• Es un programador “senior”.</li><li>• Realiza funciones de análisis porque sus conocimientos lo permiten y también codifica.</li><li>• En proyecto pequeños puede realizar ambas funciones (analista y programador).</li></ul>	<ul style="list-style-type: none"><li>• Conocimientos técnicos: LP, BD, Web, protocolos, herramientas, metodologías ágiles, ...</li><li>• Iniciativa</li><li>• Capacidad de autoaprendizaje.</li><li>• Analista</li><li>• Detallista</li><li>• Trabajo en equipo y colaboración</li></ul>	<ul style="list-style-type: none"><li>• Diseño</li><li>• Implementación</li><li>• <i>Documentación</i></li></ul>	<ul style="list-style-type: none"><li>• Ingeniería Informática o Técnico superior (DAW, DAM, ...)</li></ul>



## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo

#### Programador

Funciones	Habilidades	Interviene	Formación
<ul style="list-style-type: none"><li>• Codificar las tareas encomendadas por el analista o el desarrollador ("picar código")</li><li>• Depurar errores.</li><li>• Pruebas unitarias.</li><li>• Mantener la aplicación.</li><li>• Implementar nuevas funcionalidades.</li></ul>	<ul style="list-style-type: none"><li>• Conocimiento en profundidad de LP.</li><li>• Autoaprendizaje.</li><li>• Proactivo</li><li>• Detallista y calidad en el código que genera.</li><li>• Trabajo en equipo.</li><li>• Flexibilidad.</li></ul>	<ul style="list-style-type: none"><li>• Implementación</li><li>• <i>Documentación</i></li></ul>	<ul style="list-style-type: none"><li>• Técnico superior (DAW, DAM, ...)</li></ul>

## 3. INGENIERIA DEL SW

### 3.3. Roles o equipos de trabajo Tester

#### Funciones

- Asegurar que los requisitos definidos por el **arquitecto de SW** son cumplidos en la implementación
- Junto a programadores y desarrolladores de SW aplicará diferentes métodos de testeo e informará sobre todos los errores encontrados durante la fase de prueba.
- Detectar puntos que puedan ocasionar futuros fallos (QA)

#### Habilidades

- Metodologías y modelos de calidad de SW.
- Herramientas de testeo.
- Automatización y scripting.
- Detallista.
- Trabajo en equipo.

#### Interviene

- Prueba
- *Documentación*

(El QA participa en todas las fases del desarrollo)

#### Formación

- Ingeniería Informática o de Sistemas.
- Certificación Internacional en Testing (ISTQB)

## 3. INGENIERIA DEL SW

### 3.4. Ciclos de vida de desarrollo clásicos

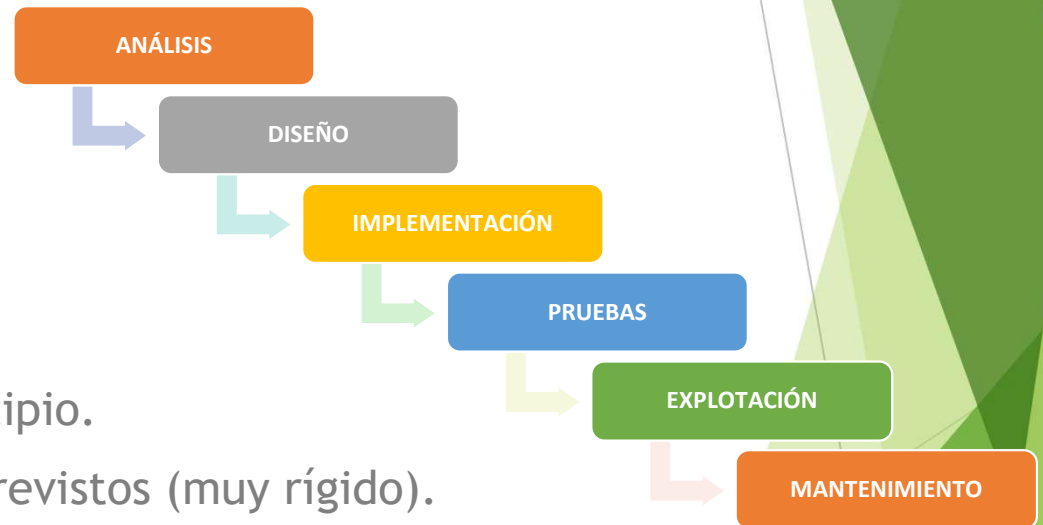
- ▶ Las fases que hemos visto en apartados anteriores se pueden distribuir de diferentes formas, dando lugar a distintos tipos de ciclo de vida.
- ▶ Estos se diferencian básicamente en:
  - ▶ La estructura de las fases (cascada, V, espirar, ....)
  - ▶ El alcance del ciclo (estudio de viabilidad, desarrollo completo, desarrollo y mantenimiento, ...)
  - ▶ Las características de cada fase (organización, división del trabajo, ...)

## 3. INGENIERIA DEL SW

### 3.4. Ciclos de vida de desarrollo clásicos

#### CICLO DE VIDA EN CASACADA

- ▶ Es el más simple.
- ▶ Se realiza cada etapa por separado y de forma secuencial.
- ▶ Para que comience una fase debe haber finalizado la anterior.
- ▶ Todo está planificado desde el principio.
- ▶ Poco margen de maniobra ante imprevistos (muy rígido).
- ▶ Útil en proyectos claros y sin cambios posibles.



# 3. INGENIERIA DEL SW

## 3.4. Ciclos de vida de desarrollo clásicos

### CICLO DE VIDA EN V

- Gestión secuencial y lineal.
- Consta de:
  - Fase descendente = necesidades del proyecto;
  - Fase ascendente = verificaciones de las necesidades.
- Verificación de cada etapa para prevenir errores.
- Muy rígido. Útil en proyectos claros y sin cambios posibles.
- Requiere más recursos.

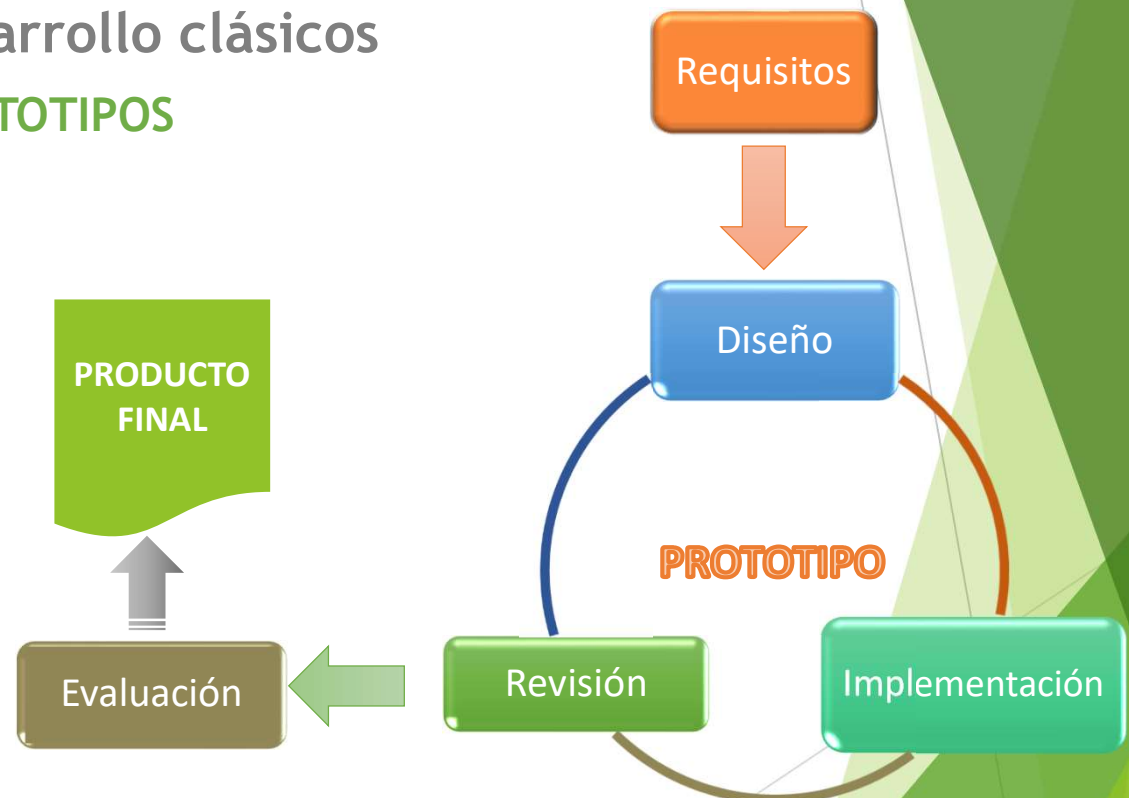


## 3. INGENIERIA DEL SW

### 3.4. Ciclos de vida de desarrollo clásicos

#### CICLO DE VIDA BASADO EN PROTOTIPOS

- ▶ Se realiza un prototipo del proyecto antes de comenzar a desarrollar por completo.
- ▶ Permite comprobar si el prototipo se ajusta a las necesidades del cliente/usuario.
- ▶ Iterativo las veces necesarias hasta conseguir producto deseado.

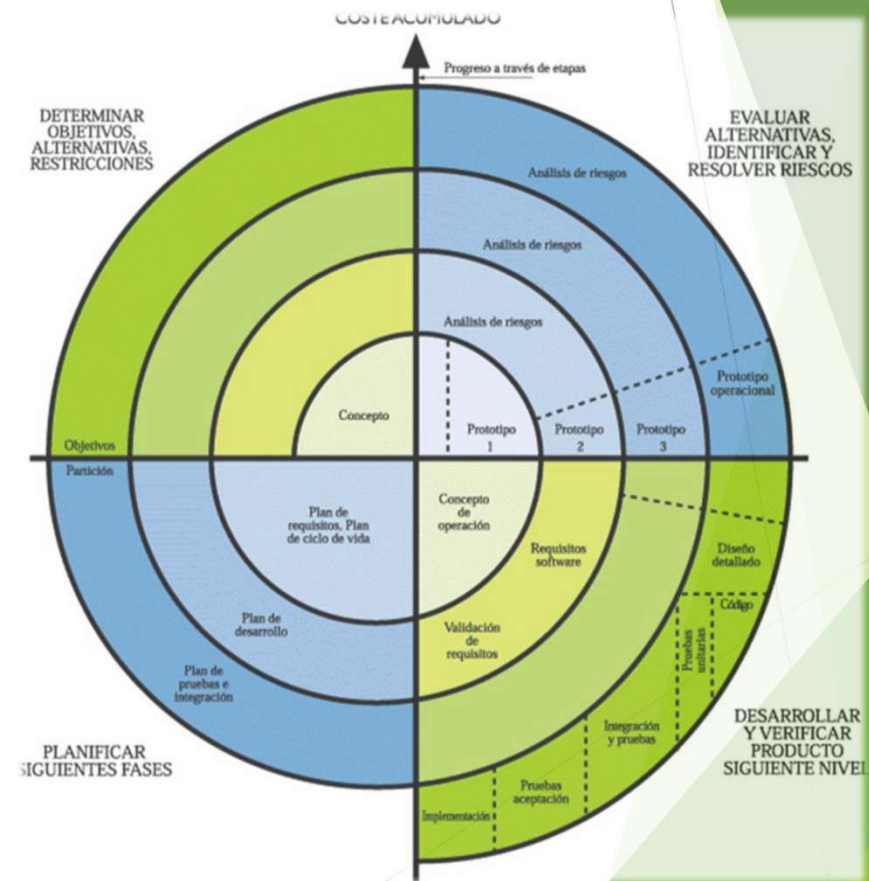


## 3. INGENIERIA DEL SW

### 3.4. Ciclos de vida de desarrollo clásicos

#### CICLO DE VIDA EN ESPIRAL

- ▶ Se van creando pequeñas partes del proyecto y se van ampliando a medida que avanza el proyecto.
- ▶ Normalmente los incrementos se realizan siguiendo una planificación inicial.
- ▶ Relativo margen de maniobra ante imprevistos.



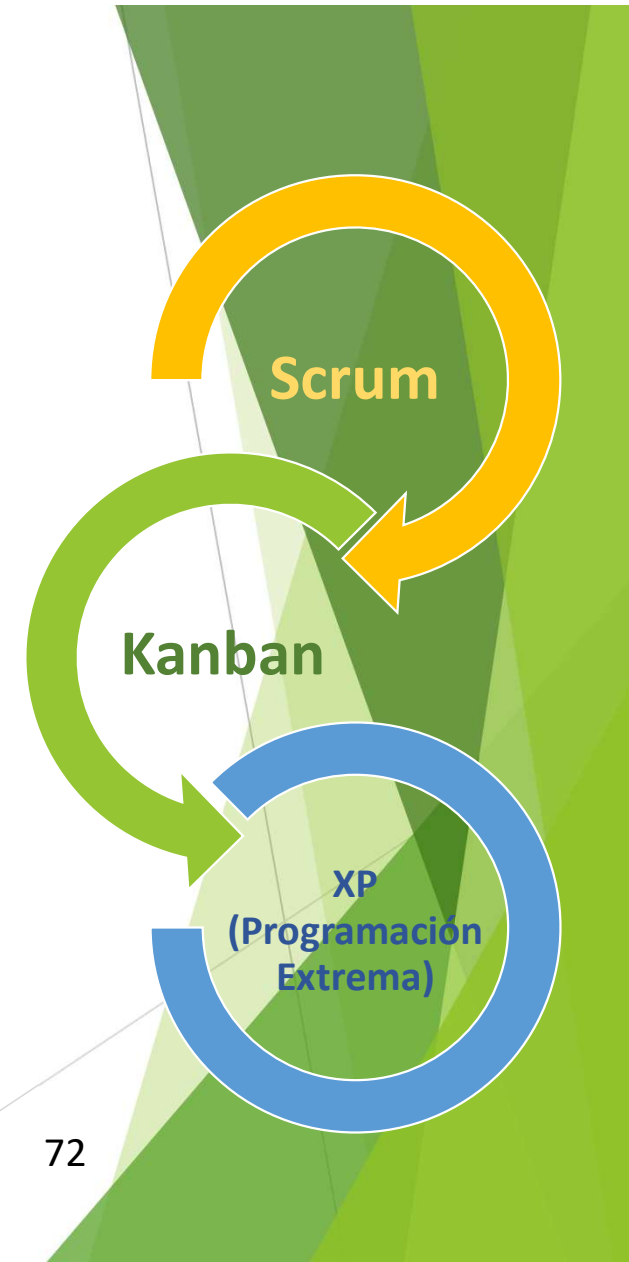
## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

**Agile** es nueva forma de trabajar, cuyo objetivo es cumplir con el desarrollo del proyecto de una manera rápida, flexible y adaptada a las circunstancias.

- ▶ Todos los equipos cooperan en todas las etapas.
- ▶ Los equipos se reúnen diariamente para explicar el estado de las tareas (terminadas, en proceso, pendientes), así como los problemas surgidos.
- ▶ El proyecto se desarrolla poco a poco, de forma iterativa.
- ▶ Se utiliza la retroalimentación como forma de controlar el desarrollo (lo que ocurra durante una iteración puede afectar a la siguiente)
- ▶ Margen total de mejora frente a imprevistos.

Para más información leer el Manifiesto Ágil





## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

#### VENTAJAS O BENEFICIOS

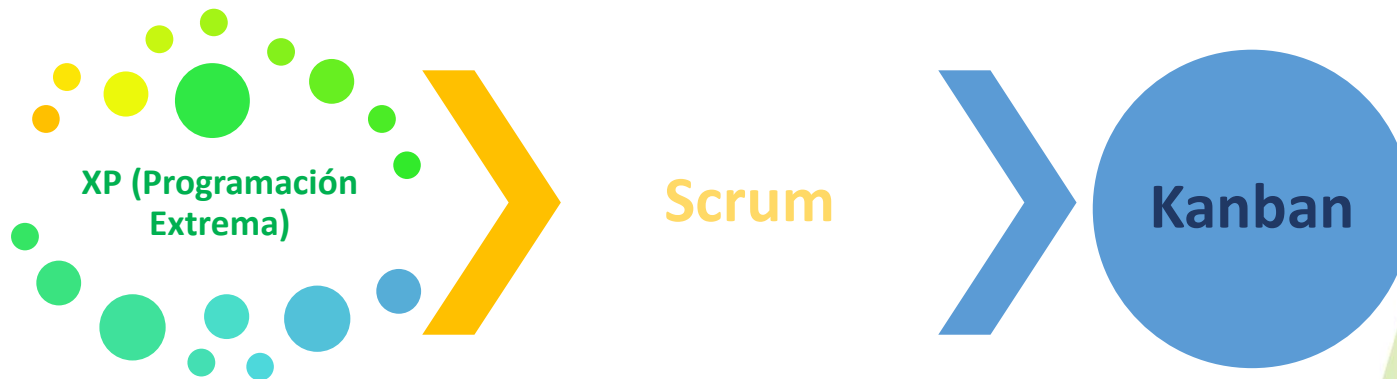
- Mejora la calidad alertando y reduciendo los errores o problemas.
- Mejora satisfacción del cliente.
- Mayor compromiso, motivación e implicación de los empleados.
- Mayor rapidez y eficiencia, y menores costes.
- Aumento de la productividad.
- Acelera el retorno de la inversión.

## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

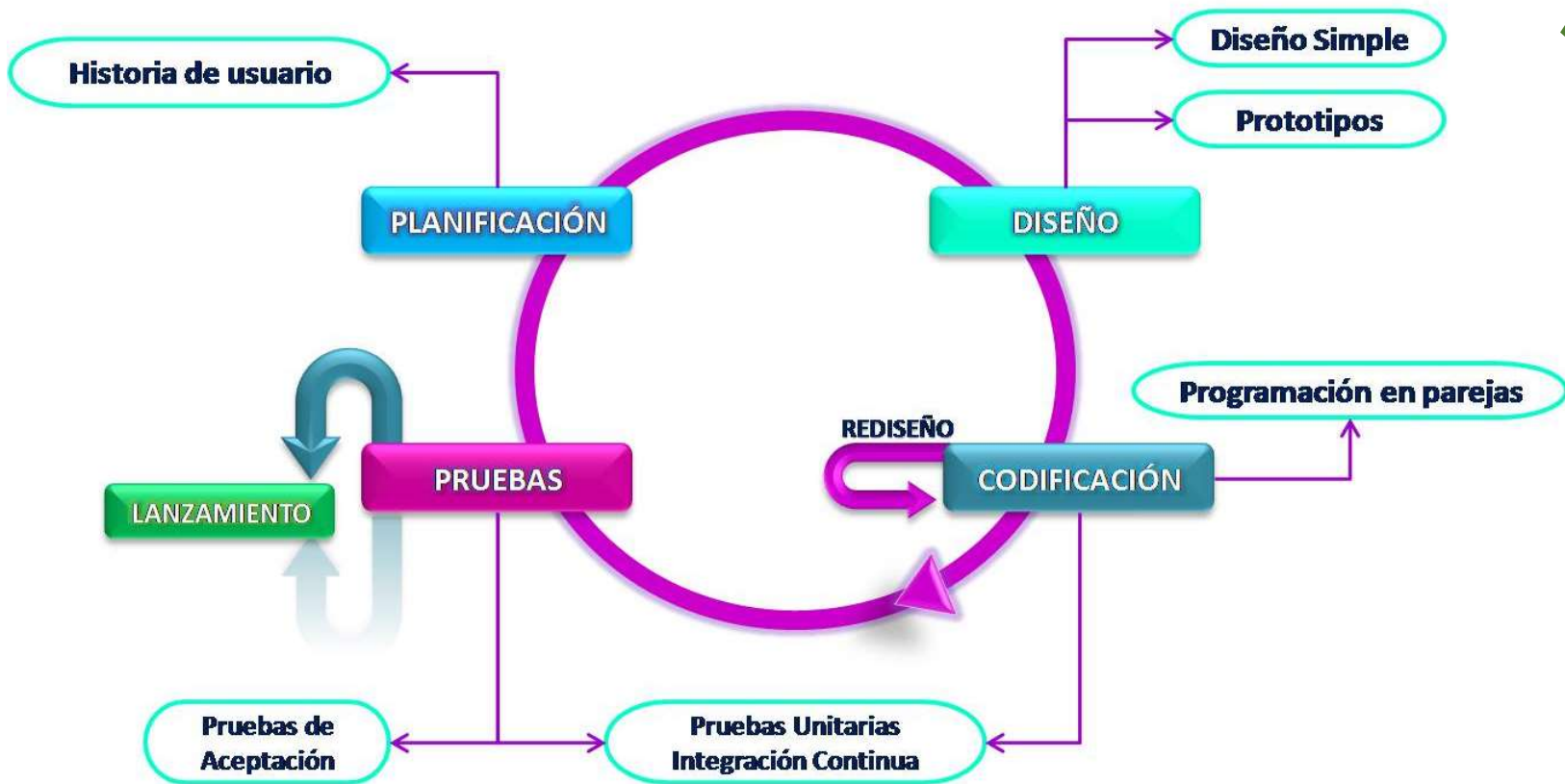
#### Metodologías Agile más habituales

A continuación, veremos algunas de las metodologías ágiles más utilizadas.



# 3. INGENIERIA DEL SW

## 3.5. Metodologías Ágiles



## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

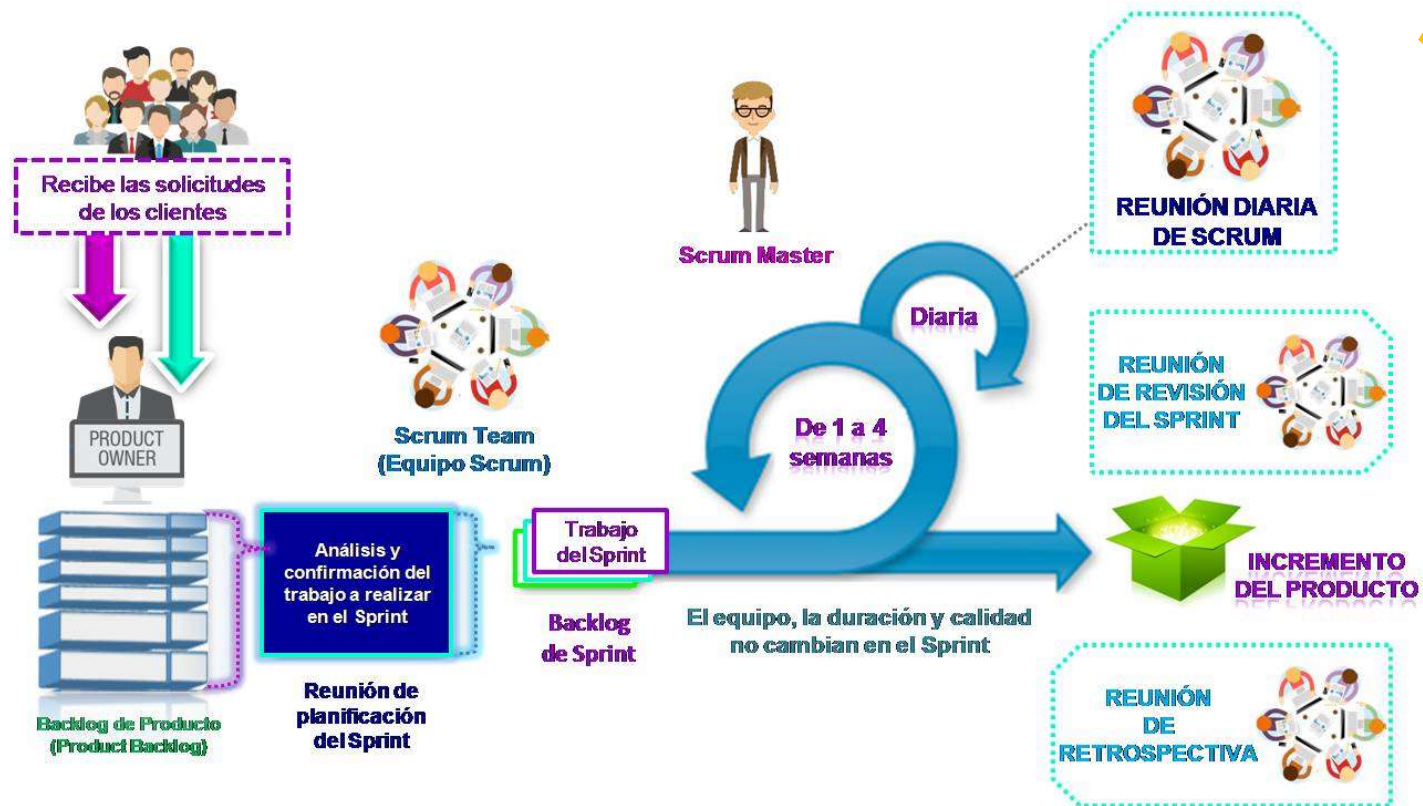
#### Características:

- ▶ Desarrollo iterativo e incremental.
- ▶ Programación en parejas.
- ▶ Pruebas unitarias continuas.
- ▶ Corrección periódica de errores.
- ▶ Integración del equipo de programación con el cliente.
- ▶ Simplicidad, propiedad del código compartida y refactorización del código.



# 3. INGENIERIA DEL SW

## 3.5. Metodologías Ágiles



## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

#### Características:

- ▶ Desarrollo iterativo e incremental.
- ▶ Trabajo colaborativo.
- ▶ Entregas regulares y parciales (**sprint**).
- ▶ Minimización de riesgos.
- ▶ Resultados inmediatos.

#### ▶ Aspectos fundamentales:

- ▶ Innovación
- ▶ Productividad
- ▶ Flexibilidad
- ▶ Competitividad



## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

#### Equipo:

#### Stakeholder

- Es el **cliente**
- Define los requerimientos (**Product Backlog**)
- Recibe el producto al final de cada iteración
- Proporciona el feedback correspondiente.

#### Product Owner

- Es el **intermediario** de la comunicación entre el cliente (**stakeholder**) y el equipo de desarrollo.
- Debe priorizar los requerimientos según sean las necesidades de la solicitud.

#### Scrum Master

- Actúa como facilitador ante todo el equipo de desarrollo
- Elimina todos aquellos impedimentos que identifique durante el proceso
- Se encarga de que se sigan los valores y principios ágiles, las reglas y los procesos de Scrum.
- Incentiva al grupo de trabajo.

#### Scrum Team (Equipo de desarrollo)

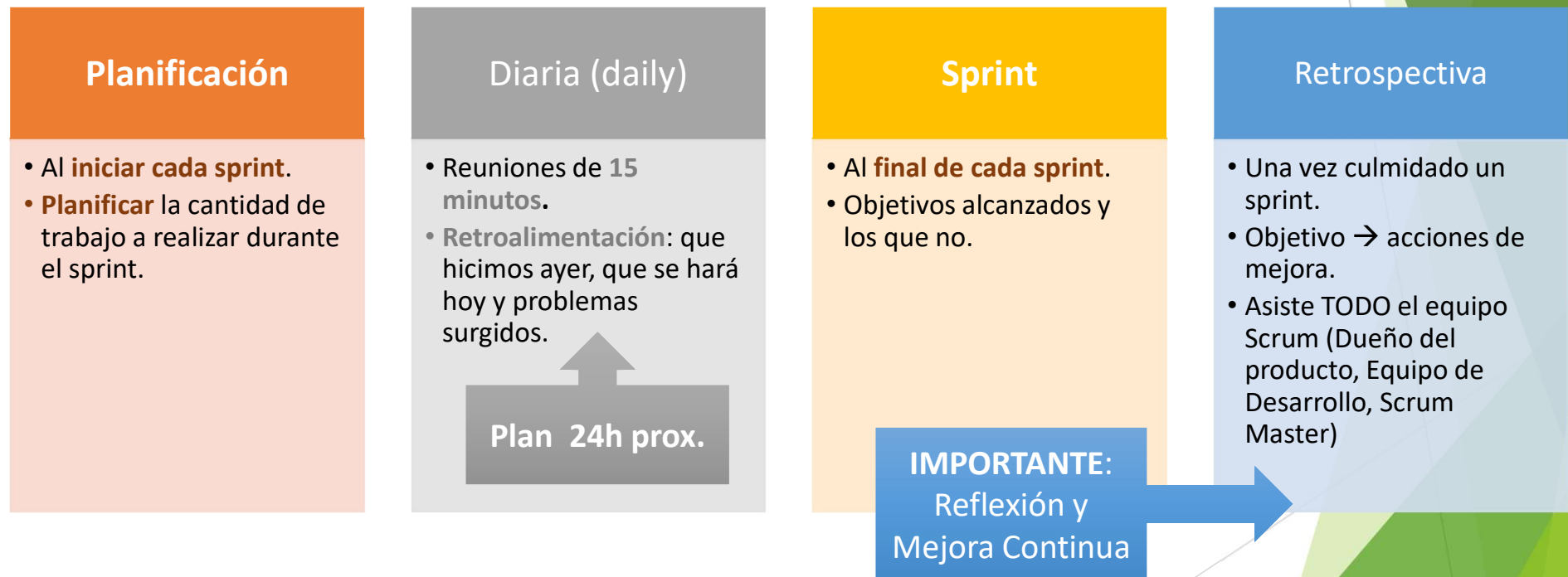
- Se encarga de desarrollar los casos de uso definidos en el Product Backlog
- Equipo auto gestionado (no existe un jefe de equipo).
- Todos los miembros se deben de encargar de realizar las estimaciones y en base a la velocidad obtenida en las iteraciones irán construyendo el **Sprint Backlog**.



# 3. INGENIERIA DEL SW

## 3.5. Metodologías Ágiles

### Reuniones:





# 3. INGENIERIA DEL SW

## 3.5. Metodologías Ágiles



## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

#### Características:

- ▶ Sencillez
- ▶ Simplifica la planificación y asignación de tareas.
- ▶ Tablero con el flujo de trabajo:
  - ▶ Pendiente → Solicitudes del cliente
  - ▶ En progreso → Capacidad del equipo
  - ▶ Terminado
- ▶ Las tarjetas van cambiando de columna según el progreso del trabajo.

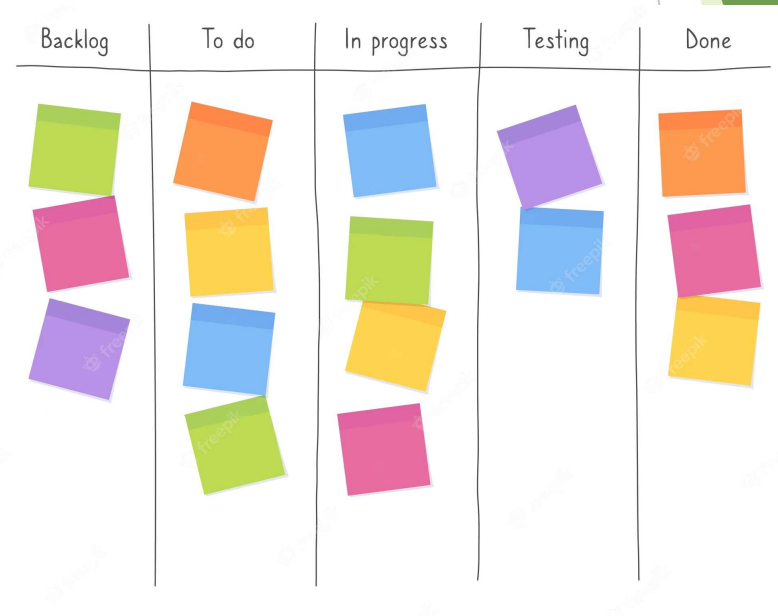


## 3. INGENIERIA DEL SW

### 3.5. Metodologías Ágiles

#### Ventajas:

- Planificación de tareas.
- Tiempos de ciclos reducidos.
- Rendimiento del equipo de trabajo.
- Métricas visuales.
- Menos cuellos de botella.
- Entrega continua.





**Fin**

Entornos de desarrollo  
M<sup>a</sup> Carmen Safont Richarte