

# ENTORNOS DE DESARROLLO

## TEMA 1:

## DESARROLLO DE SOFTWARE

Desarrollo de Aplicaciones Multiplataforma



**Curso 2021/2022**

**Profesor:**

Marcos Unzueta Puente

# CONTENIDO BOCYL (nº 115 15-junio-2011)

- Concepto de programa informático.
- Relación entre el software y el hardware de un equipo informático.
- Código fuente, código objeto y código ejecutable; máquinas virtuales.
- Tipos de lenguajes de programación. Clasificaciones.
- Características de los lenguajes más difundidos.
- Proceso de obtención de código ejecutable a partir del código fuente: compilación y linkado; herramientas implicadas.
- Aplicaciones informáticas. Definición. Clasificación.
- Fases del desarrollo de una aplicación: análisis, diseño, codificación, pruebas, documentación, explotación y mantenimiento, entre otras.
- Lenguaje de propósito general para el modelado: UML. Diagramas.

# Índice

CONTENIDO BOCYL (nº 115 15-junio-2011).....	2
Índice .....	3
Índice de ilustraciones .....	5
1. SOFTWARE Y PROGRAMA. TIPOS DE SOFTWARE.....	7
1.1. Sistemas Operativos .....	9
1.2. Drivers.....	13
1.3. Entornos de Desarrollo Integrados (IDE) .....	15
2. RELACIÓN HARDWARE-SOFTWARE .....	20
2.1. Punto de vista del Sistema Operativo.....	22
2.2. Punto de vista de las aplicaciones .....	23
3. DESARROLLO DEL SOFTWARE .....	26
3.1. Definición de software.....	26
3.2. Características del software.....	26
3.3. Características de un buen software .....	26
3.4. CICLOS DE VIDA DEL DESARROLLO SOFTWARE .....	29
3.4.1. Modelos Tradicionales.....	30
3.4.2. Modelos Evolutivos .....	31
3.4.3. Proceso Unificado.....	34
3.4.4. Metodologías ágiles.....	36
3.4.5. MODELO DE CONSTRUCCIÓN DE PROTOTIPOS.....	40
3.5. Herramientas de apoyo al desarrollo del software .....	43
4. LENGUAJES DE PROGRAMACIÓN .....	46
4.1. Lenguajes de programación según su nivel de abstracción .....	49
4.2. Lenguajes de programación según forma de ejecución .....	51
4.3. Lenguajes de programación según su paradigma de programación.....	52
4.4. Otros tipos de lenguajes de programación.....	54
5. FASES EN EL DESARROLLO Y EJECUCIÓN DEL SOFTWARE.....	57

5.1. Análisis .....	57
5.2. Diseño .....	62
5.3. Codificación e implementación .....	<b>¡Error! Marcador no definido.</b>
5.4. PRUEBAS .....	<b>¡Error! Marcador no definido.</b>
5.5. IMPLANTACIÓN → EXPLOTACIÓN / PRODUCCIÓN. ....	<b>¡Error! Marcador no definido.</b>
5.6. MANTENIMIENTO .....	<b>¡Error! Marcador no definido.</b>
5.7. DOCUMENTACIÓN .....	<b>¡Error! Marcador no definido.</b>
REFERENCIAS .....	64

## Índice de ilustraciones

Ilustración 1: Programas Informáticos .....	9
Ilustración 2: Sistema Operativo .....	10
Ilustración 3: Sistemas Operativos más usados 2020 .....	11
Ilustración 4: Meme sistemas operativos .....	12
Ilustración 5: Periféricos de entrada .....	14
Ilustración 6: Periféricos de salida .....	14
Ilustración 7: Periféricos de entrada/salida .....	14
Ilustración 8: Periféricos de almacenamiento .....	15
Ilustración 9: Compilador .....	16
Ilustración 10: Interfaz de Eclipse.....	17
Ilustración 11: Interfaz de IntelliJ.....	17
Ilustración 12: Interfaz JBiulder .....	18
Ilustración 13: Interfaz KDevelop .....	19
Ilustración 14: Fallout PS3 .....	20
Ilustración 15: Godzilla PS4 .....	21
Ilustración 16: Desaprovechamiento de recursos.....	21
Ilustración 17: Paralelismo .....	22
Ilustración 18: Programa en Python .....	24
Ilustración 19: Programa en Perl .....	24
Ilustración 20: Programa en Typescript.....	25
Ilustración 21: programa en C++ .....	25
Ilustración 22: Comandos .....	27
Ilustración 23: Cola Ticketea.....	27
Ilustración 24: Frase pomposa Steve Jobs.....	28
Ilustración 25: Modelo en cascada .....	30
Ilustración 26:Modelo incremental .....	32
Ilustración 27: Modelo en espiral .....	33
Ilustración 28: Ejemplo de modelo en espiral .....	34
Ilustración 29: Proceso Unificado .....	36
Ilustración 30: Regreso al futuro .....	37
Ilustración 31: Reunión metodologías ágiles.....	37
Ilustración 32: Dilbert metodologías ágiles .....	39
Ilustración 33: Dilbert programación extrema .....	40
Ilustración 34: Porcentaje metodologías usadas .....	40
Ilustración 35: Prototipado en papel .....	41
Ilustración 36: Android Studio .....	43
Ilustración 37:Logo Modern Requirements.....	44

Ilustración 38: Logo Visual Paradigm .....	44
Ilustración 39: Logo Visual Paradigm .....	44
Ilustración 40: Logo Python .....	44
Ilustración 41: Logo Postman .....	45
Ilustración 42: JavaDoc logo .....	45
Ilustración 43: Framer .....	45
Ilustración 44: Logos git github gitkraken.....	45
Ilustración 45: Compilación .....	47
Ilustración 46: Comic lenguajes de programación .....	48
Ilustración 47: Lenguaje máquina .....	49
Ilustración 48: Lenguaje ensamblador .....	49
Ilustración 49: Lenguaje de nivel medio.....	50
Ilustración 50: Lenguaje de alto nivel.....	50
Ilustración 51: Lenguaje visual .....	51
Ilustración 52: Compilación e interpretación .....	52
Ilustración 53: Programación estructurada.....	52
Ilustración 54: Programación orientada a objetos.....	53
Ilustración 55: Programación orientada a eventos .....	54
Ilustración 56: Lenguaje lado del cliente .....	54
Ilustración 57: CSS .....	54
Ilustración 58: Lado del servidor .....	55
Ilustración 59: Front-End vs Back-End.....	55
Ilustración 60: Lenguaje de comunicación con el SGBD.....	56

# 1. SOFTWARE Y PROGRAMA. TIPOS DE SOFTWARE

El funcionamiento de un ordenador depende principalmente de dos partes bien diferenciadas: hardware y software.

- **Hardware:** Se refiere al conjunto de materiales y componentes físicos de un dispositivo (aquello que es tangible, que se puede tocar). El origen etimológico de la palabra hardware viene del inglés y se compone de las palabras “hard” que significa “duro” y “ware” que significa cosas.
- **Software:** El software se considera la parte digital de un dispositivo. No tiene una existencia física, es intangible. El origen etimológico de la palabra hardware viene del inglés y se compone de las palabras “soft” que significa “blando” y “ware” que significa cosas.

Ninguno de estos elementos puede funcionar individualmente.

[¿Qué es software y qué es hardware?](#)

**EJERCICIO 1:** ¿Qué dispositivos, además del ordenador, que tengan hardware y software se te ocurren?

Existen muchas clasificaciones para el software, no obstante, la más común se basa en el objetivo que tiene dentro del sistema informático.

**Clasificación en función del objetivo:**

- **Software del sistema:** Permite al usuario ignorar los detalles internos del sistema (tened en cuenta que el ordenador se maneja internamente en unos y ceros y nosotros nunca “hablamos” con él en unos y ceros.). Este tipo de Software actúa como traductor entre el usuario y el sistema. Administra la parte física y los recursos del dispositivo. Dentro del software del sistema se encuentran:
  - **Sistemas Operativos** (Windows, Linux, Mac)
  - **Controladores de dispositivos** (Drivers)
  - **Herramientas de diagnóstico** (Antivirus)
  - **Servidores**
  - ...

- **Software de programación:** Conjunto de herramientas que permiten al usuario crear programas informáticos. Dentro del software de programación se encuentran:
  - Editores de texto
  - Compiladores
  - Intérpretes
  - Enlazadores
  - Depuradores
  - Entornos de Desarrollo Integrado
  - ...
- **Software de aplicación:** Programas diseñados para que el usuario desarrolle una tarea específica en cualquier campo susceptible de ser automatizado o asistido.  
Hacen que el ordenador sea una herramienta útil para el usuario.  
Dentro del software de aplicación se encuentran:
  - Aplicaciones ofimáticas
  - Software educativo
  - Software empresarial
  - Bases de datos
  - Telecomunicaciones
  - Videojuegos
  - Software matemático.
  - Software de diseño asistido.
  - Aplicaciones para control de sistemas y automatización industrial.
  - ...

**EJERCICIO 2:** Investiga y escribe ejemplos de cada tipo de Software de Aplicación (con imágenes) e indica alguno más a parte de los especificados.

**Clasificación según la licencia por la que se distribuye:**

- **Software libre:** Software libre es aquel que se suministra con autorización para que cualquiera pueda usarlo, copiarlo y/o distribuirlo, ya sea con o sin modificaciones, gratuitamente o mediante pago.  
Libre no es sinónimo de gratuito.
- **Software propietario:** O software privado. Limita las libertades del usuario respecto a su distribución, modificación e incluso a su uso.

**Programa (informático):** Conjunto de instrucciones escritas en un lenguaje de programación y que permiten realizar tareas en un dispositivo programable.



Vamos a poner un ejemplo de la vida real, para entenderlo mejor. Imagina una receta de cocina.

En una receta tienes principalmente unos ingredientes y unos pasos a seguir (órdenes).

Estos pasos a seguir se deben ejecutar en el mismo orden que viene en la receta, sino el objetivo final (el plato que queremos hacer) no tendrá el resultado esperado.



*Ilustración 1: Programas Informáticos*

Windows, Google Chrome y Call of Duty tienen en común que son programas informáticos.

## 1.1. Sistemas Operativos

Software base que ha de estar instalado en la máquina para que las aplicaciones puedan ejecutarse y funcionar. Realmente es el software que hace al ordenador “útil” y amigable al usuario.



*Ilustración 2: Sistema Operativo*

El SO administra los recursos del computador (Hardware y Software) del equipo.

Cuando se pulsa el botón de encender del ordenador, este realiza pruebas para comprobar que todo funciona bien, comprueba su hardware e inicia el sistema operativo.

Para ordenador existen tres sistemas operativos principales: Linux, macOS y Windows.

Sistema Operativo	Porcentaje
Windows	76,84 %
OSX	17,63 %
Desconocido	2,49 %
Linux	1,76%
Chrome OS	1,28 %
FreeBSD	0 %

*Ilustración 3: Sistemas Operativos más usados 2020*

- **Microsoft Windows:**
  - Desarrollado en los 80.
  - Sus últimas versiones son Windows 11 (05/10/2021), Windows 10 (2014) y Windows 8 (2012).
  - Windows 11:
    - Actualización más ambiciosa de la compañía.
    - En vez de vender licencias por 150€ las van a dar de manera gratuita e intentarán ganar dinero de los servicios (Office, Skype, etc.)
    - Gran cambio de apariencia
    - Mejora en la experiencia táctil
    - Soporte para aplicaciones Android
  - Viene preinstalado en la mayoría de los ordenadores.
  - Es el sistema operativo de ordenadores más usado.
  - Tiene la mayor compatibilidad de software (casi todo el software existente tiene una versión para Windows).
  - Muy bueno para gaming.

[Explicación novedades Windows 11](#)

[Novedades Windows 11 \(vídeo\)](#)

- **MacOS:**
  - Solo puede utilizarse oficialmente en ordenadores Macintosh (Desarrollados por Apple)
  - Desarrollado por Apple Inc.
  - Segundo sistema operativo más utilizado a nivel mundial.
  - Diseño intuitivo

- macOS es menos exigente con el hardware porque está mejor optimizado y, por tanto, consume muchos menos recursos.
  - Muy estable
  - Muy seguro
  - Se sincroniza con otros dispositivos Apple
  - Más caro.
- **GNU/Linux:**
    - Hace referencia a los sistemas basados en UNIX cuyo núcleo se conoce como Linux (creado por Linus Torvalds).
    - Es un software libre.
    - Muy bueno para programar (se usa mucho en servidores).
    - Su uso no es tan amigable como el de otros Sistemas Operativos.
    - Muy poco utilizado en el sector doméstico.
    - Es el más seguro
    - Código fuente abierto.
    - Es muy flexible.



Ilustración 4: Meme sistemas operativos

**EJERCICIO 3:** Investiga sobre los Sistemas Operativos mencionados e indica las ventajas y desventajas que ven los usuarios a cada sistema operativo. Indica las ventajas y desventajas que ves como usuario a cada sistema operativo (solo de aquellos sistemas operativos que hayas usado).



**EJERCICIO 4:** Investiga sobre Sistemas Operativos más allá de ordenadores y móviles, indica el que más te ha llamado la atención y por qué.

**EJERCICIO 5:** ¿Es posible instalar Windows en un MAC? ¿Y Linux?

**EJERCICIO 6:** ¿De dónde pueden venir las letras OS de macOS?

## 1.2. Drivers

**Drivers:** Programas (forman parte del software del sistema) que actúan de enlace entre el sistema operativo instalado y los periféricos que forman el ordenador o que se tienen conectados a él.

**Periféricos (de un ordenador):** Forman parte del hardware del ordenador y son los dispositivos que están conectados a este pero que no son parte del equipo principal y permiten la entrada/salida de información desde o hacia el propio ordenador (los hay de entrada, salida, entrada/salida, almacenamiento ...).

*Ilustración 5: Periféricos de entrada**Ilustración 6: Periféricos de salida*

### PERIFERICOS ENTRADA/SALIDA

*Ilustración 7: Periféricos de entrada/salida*



Ilustración 8: Periféricos de almacenamiento

Un fallo en el driver de un periférico se puede traducir en el fallo en el funcionamiento de este (por ejemplo, un fallo en el driver del altavoz puede hacer que el altavoz no reproduzca sonidos.)

#### [Problemas más comunes en Drivers](#)

### 1.3. Entornos de Desarrollo Integrados (IDE)

Como se ha podido ver en la clasificación del software en función de su objetivo, los entornos de desarrollo forman parte del software de programación.

Los softwares de programación más comunes son los siguientes:

**Editores de texto:** Permite, entre otras cosas, modificar y crear archivos. Estos archivos serán los que contengan las instrucciones en el lenguaje de programación correspondiente.

Por sí solo, el editor de texto no puede generar un fichero ejecutable.

**Compiladores:** Se define como el Software que traduce un programa escrito en lenguaje de programación de alto nivel (C++, C, COBOL ...) a lenguaje máquina (lenguaje interpretable directamente por el ordenador).

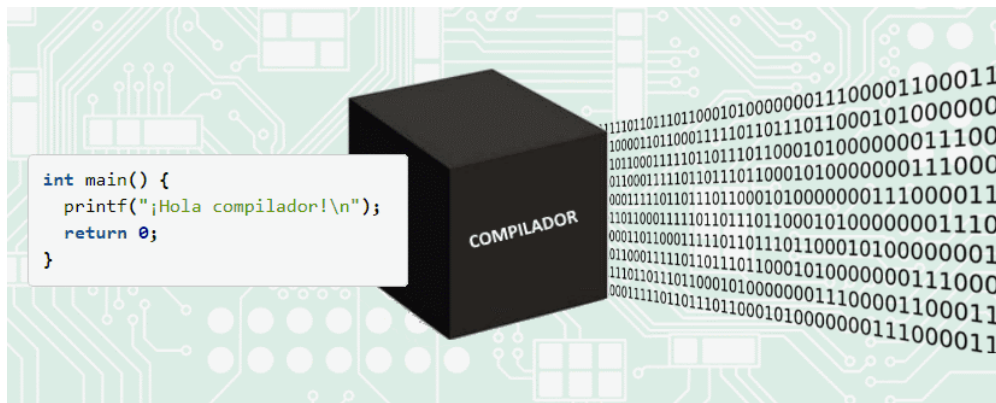


Ilustración 9: Compilador

**Intérpretes:** El intérprete, a diferencia del compilador, no convierte el lenguaje de programación a lenguaje máquina, este ejecuta directamente las instrucciones escritas en un lenguaje de programación dado. Algunos lenguajes de programación que usan intérpretes y no compiladores son BASIC, Java, Python... (lenguajes interpretados).

**Enlazadores:** Los enlazadores usan los resultados de las primeras etapas de compilación, otros recursos necesarios (como bibliotecas), quita los recursos que no necesita y enlaza el código generando un fichero ejecutable.

**Depuradores:** Se encargan de identificar y corregir errores de programación.

**Entorno de desarrollo integrado:** Agrupa todas las herramientas mencionadas previamente y proporciona una interfaz gráfica a partir de la que se puede programar, de esta manera se enmascaran y facilitan los procesos de enlazado, depuración, compilación, interpretación, edición de texto, etc.

Nos proporciona también comandos y atajos de teclado para agilizar la programación.

Algunos ejemplos son:

- Eclipse.



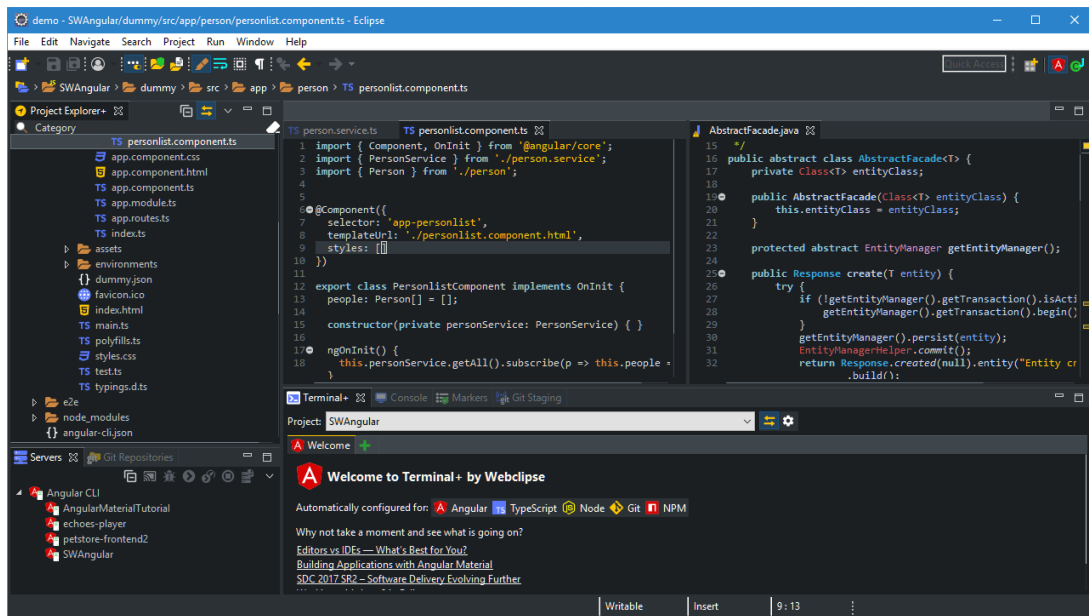


Ilustración 10: Interfaz de Eclipse

- NetBeans.
- IntelliJ IDE.

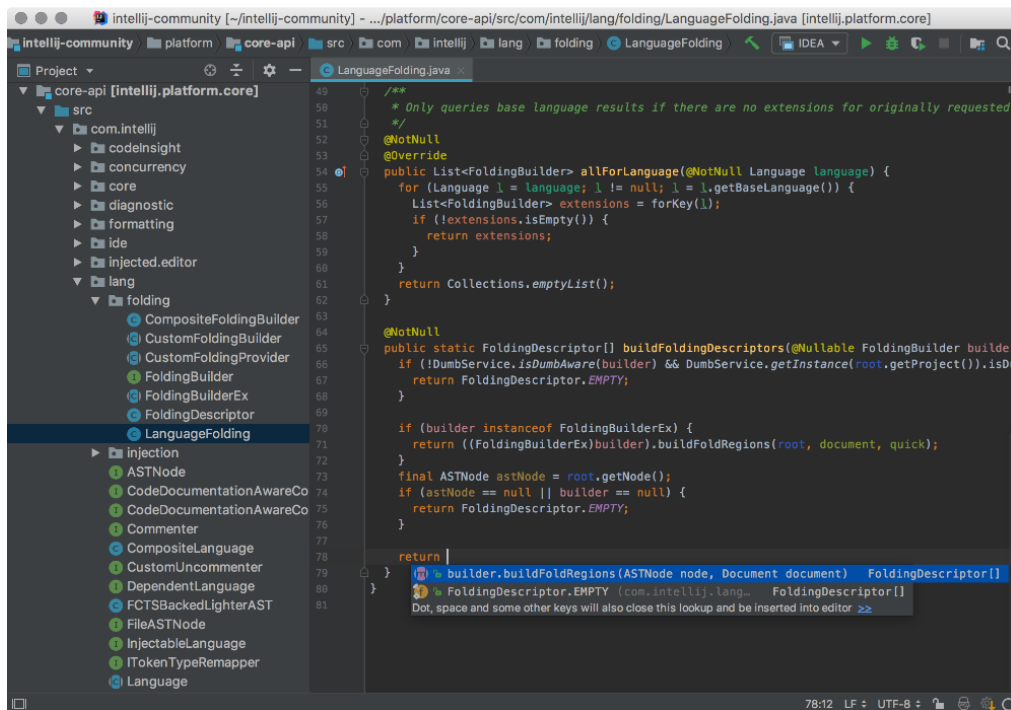


Ilustración 11: Interfaz de IntelliJ

- JBuilder de Borland.

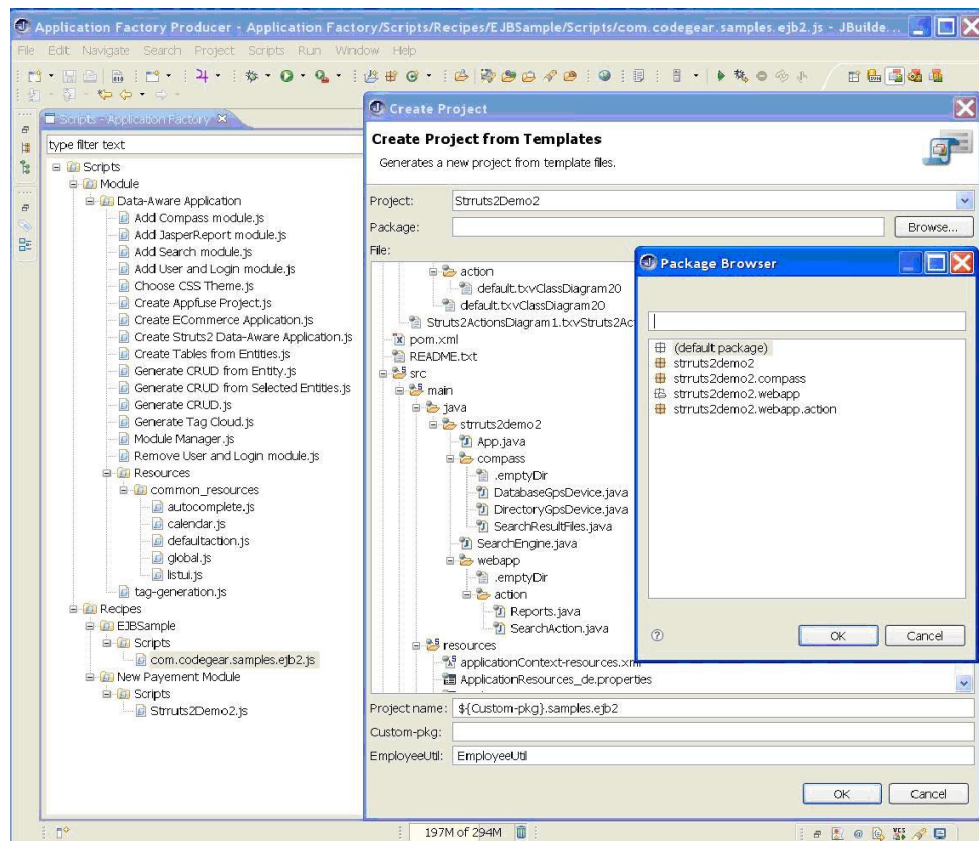


Ilustración 12: Interfaz JBuilder

- KDevelop

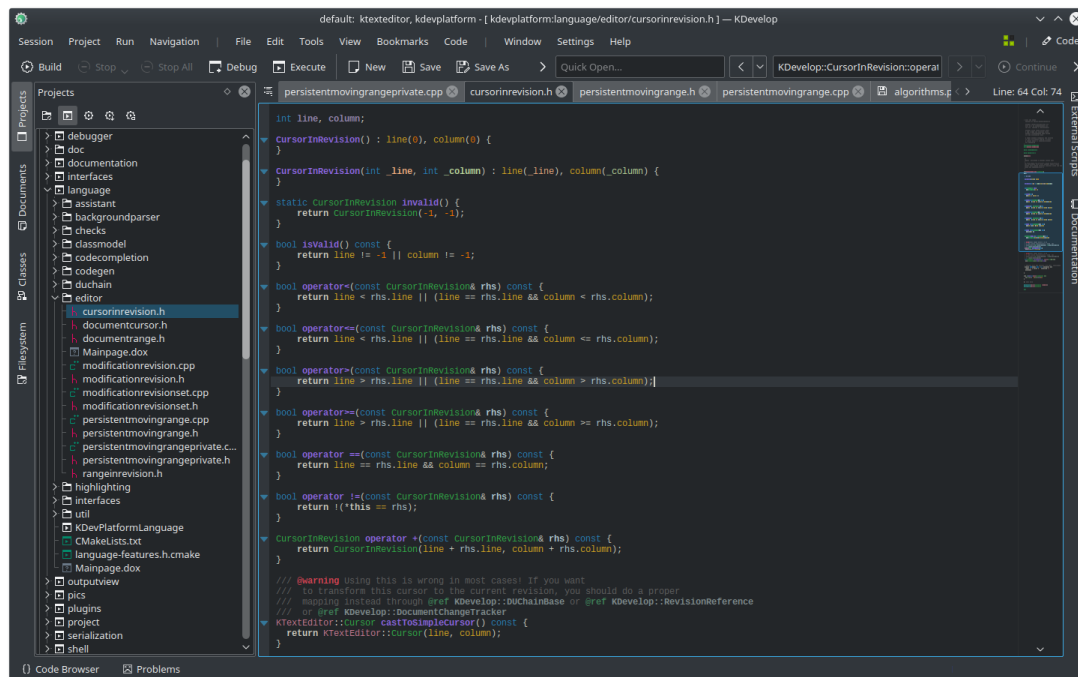


Ilustración 13: Interfaz KDevelop

- ...

**EJERCICIO 7:** ¿Qué ventajas has observado al programar en un entorno de desarrollo en vez de en el Bloc de Notas?

**EJERCICIO 8:** Dentro del entorno de desarrollo que utilizas ¿Dónde dirías que se encuentran los siguientes softwares de programación?:

- Editores de texto
- Intérpretes
- Enlazadores
- Depuradores

**EJERCICIO 9:** Analizando las imágenes del resto de entornos de desarrollo ¿Qué similitudes ves entre ellos? ¿y qué diferencias?

## 2.RELACIÓN HARDWARE-SOFTWARE

**Hardware:** Se refiere al conjunto de materiales y componentes físicos de un dispositivo (aquello que es tangible, que se puede tocar).

Todos los programas necesitan recursos hardware durante su ejecución.

Podemos tener el mejor hardware del mundo, pero si no somos buenos programadores y hacemos un buen software no nos valdrá de nada. De la misma manera un buen software puede ser desaprovechado cuando se ejecuta o se hace funcionar en un mal hardware, por ello debe existir un equilibrio entre ambas.



*Ilustración 14: Fallout PS3*



Ilustración 15: Godzilla PS4



Ilustración 16: Desaprovechamiento de recursos

Un ejemplo del buen aprovechamiento de los recursos hardware sería usar la programación en paralelo en lugar de la programación secuencial.

**EJERCICIO 10:** Importad en NetBeans los proyectos EjercicioSecuencial y EjercicioParalelo que se encuentran en los archivos del Teams. ¿Qué diferencia veis entre ambos? Si EjercicioSecuencial se ejecutara en el ordenador Marenostrum (<https://www.bsc.es/es/marenostrum/marenostrum>) y EjercicioParalelo lo ejecutarías vosotros en vuestro ordenador, ¿Cuál creéis que tardaría menos? ¿Por qué?

**EJERCICIO 11:** Enuncia otra situación que se te ocurra en el cual se podría utilizar la programación paralela. ¿En qué caso sería imposible usarla?

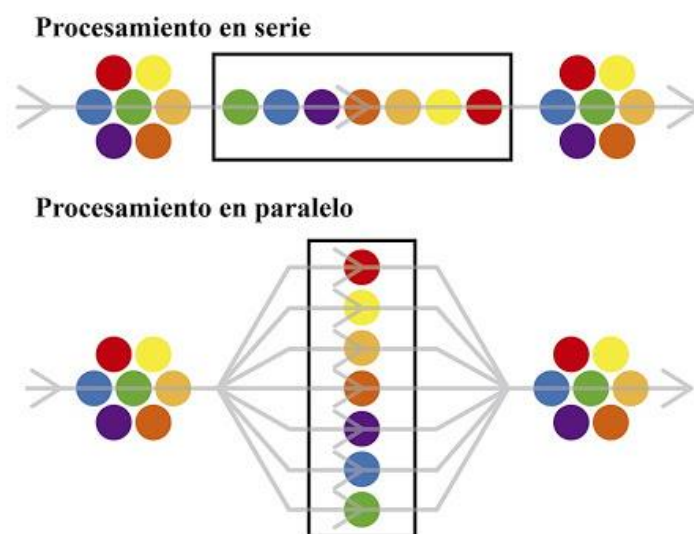


Ilustración 17: Paralelismo

**Proceso:** Programa en ejecución.

Para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación será necesario un proceso de traducción de código (compilar/enlazar - interpretar).

## 2.1. Punto de vista del Sistema Operativo

El **sistema operativo** es el encargado de coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.



Todas las aplicaciones necesitan recursos **hardware** durante su ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de Entrada/Salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).

## 2.2. Punto de vista de las aplicaciones

Una **aplicación** no es otra cosa que un conjunto de programas y éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar

Hay multitud de **lenguajes de programación** diferentes. Sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, el hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión) que, en informática, se traducen en secuencias de 0 y 1 (código binario).

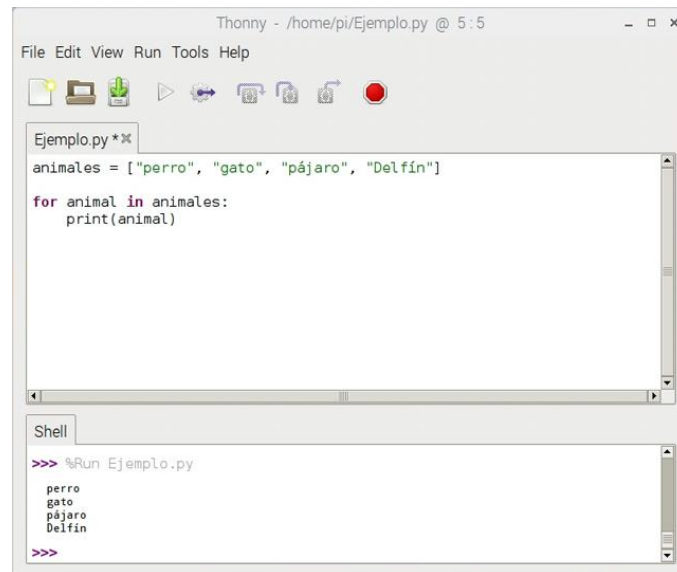
Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?

Hay que entender que un lenguaje de programación es un lenguaje, como podría ser el esperanto, si yo solo hablo español y mi compañero solo habla esperanto necesitaré a alguien entre medias que nos permita entendernos.

Será necesario un proceso de traducción de código para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

**EJERCICIO 12:** A pesar de la gran variedad existente, la mayoría de los lenguajes de programación guardan una gran de similitudes entre sí. Aunque no sepamos programar en ciertos lenguajes podemos llegar a entender lo que se está haciendo en estos. Intenta explicar lo que hacen los programas mostrados en las siguientes imágenes.

**Python:**

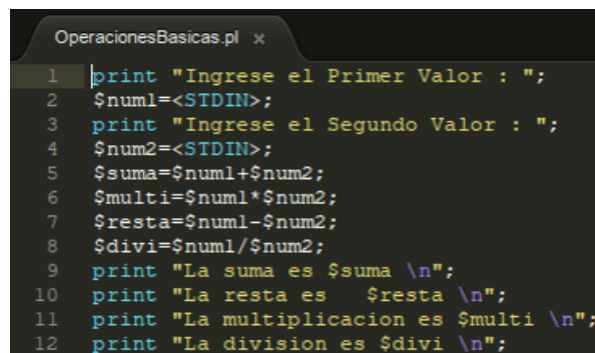


The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for opening files, saving, running, and other functions. The main editor window displays a Python script named 'Ejemplo.py' with the following code:

```
animales = ["perro", "gato", "pájaro", "Delfín"]  
  
for animal in animales:  
    print(animal)
```

Below the editor is a 'Shell' window showing the output of the script after running it:

```
>>> %Run Ejemplo.py  
perro  
gato  
pájaro  
Delfín  
>>>
```

*Ilustración 18: Programa en Python***Perl:**

The screenshot shows a Perl script named 'OperacionesBasicas.pl' with the following code:

```
1 print "Ingrese el Primer Valor : ";  
2 $num1=<STDIN>;  
3 print "Ingrese el Segundo Valor : ";  
4 $num2=<STDIN>;  
5 $suma=$num1+$num2;  
6 $multi=$num1*$num2;  
7 $resta=$num1-$num2;  
8 $divi=$num1/$num2;  
9 print "La suma es $suma \n";  
10 print "La resta es $resta \n";  
11 print "La multiplicacion es $multi \n";  
12 print "La division es $divi \n";
```

*Ilustración 19: Programa en Perl***Typescript:**



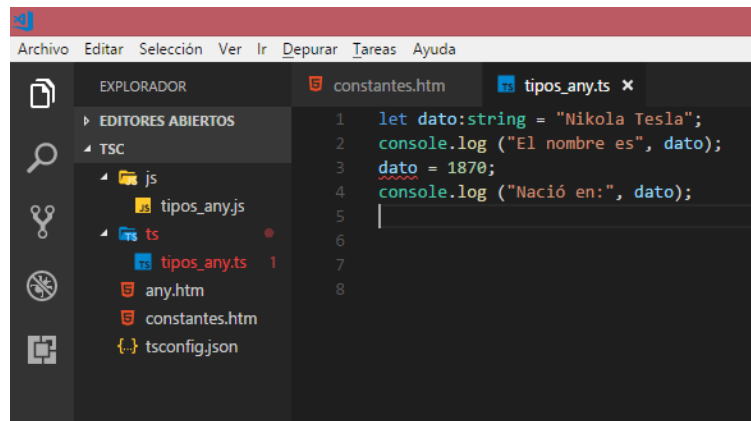


Ilustración 20: Programa en Typescript

**C++:**

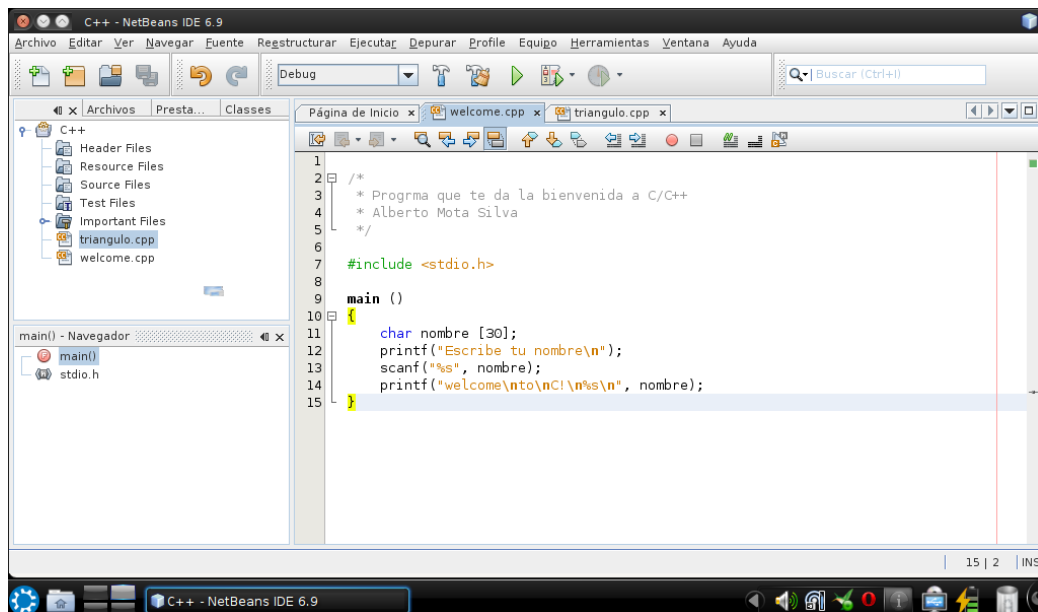


Ilustración 21: programa en C++

## 3. DESARROLLO DEL SOFTWARE

**Desarrollo del software:** proceso que ocurre desde que se concibe una idea de software hasta que este es retirado o remplazado.

### 3.1. Definición de software

**Instrucciones:** Conjunto de datos que, cuando se ejecutan, proporcionan la información necesaria para indicar al procesador que acción ha de realizar.

**Estructuras de datos** que permiten a los programas manipular adecuadamente la información.

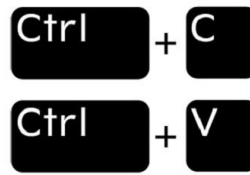
**Documentos** que describen la operación y uso de programas

### 3.2. Características del software

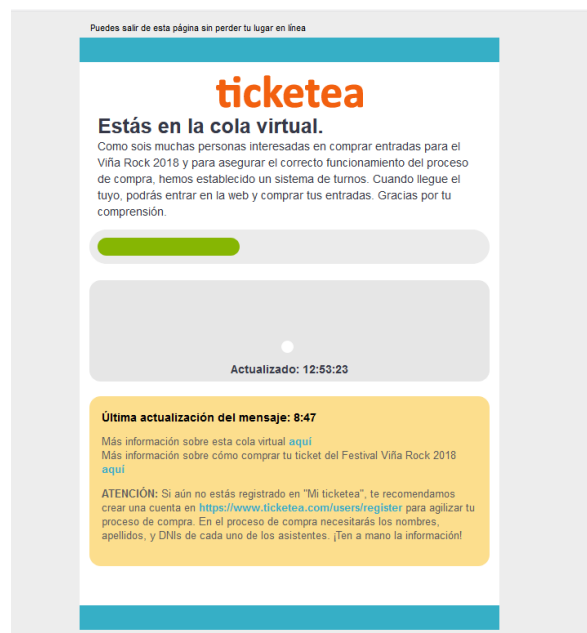
- No se fabrica, se desarrolla.
- No se estropea, falla.
- Sometido a cambios se deteriora.
- No tiene forma física.
- Normalmente se construye a medida.

### 3.3. Características de un buen software

- **Útil y aprovechable:** Hace la vida de los usuarios más fácil y cumple su objetivo.
- **Fiable:** Tiene pocos errores y no falla en tiempo de ejecución. [Videojuegos que no funcionaban el día de su lanzamiento](#)
- **Seguro:** Es resistente frente a ataques externos (ningún sistema es infalible, pero ha de ofrecer cierta seguridad). Se debe priorizar la seguridad de los datos del cliente. [Ciberataque EasyJet](#)
- **Eficiente:** Aprovecha correctamente los recursos disponibles (uso eficaz el almacenamiento, de la potencia, etc.).
- **Eficaz:** Permite al usuario hacer lo mismo en menos tiempo (usando comandos, atajos de teclado ...).

*Ilustración 22: Comandos*

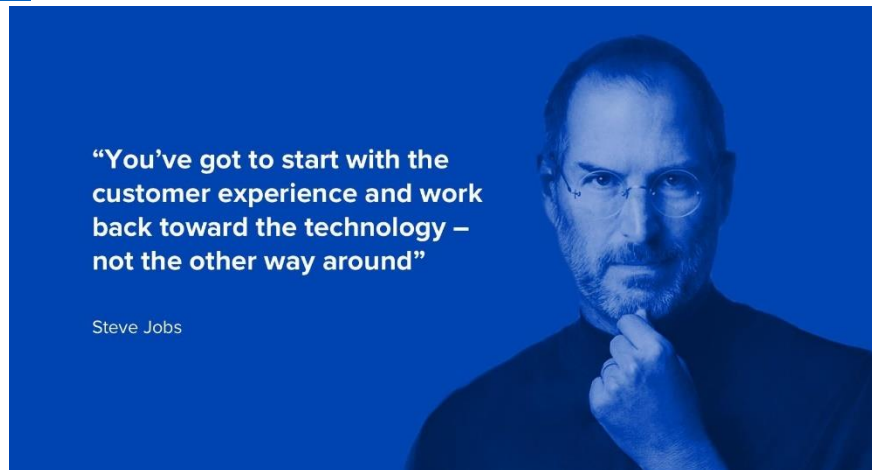
- **Flexible y escalable:** Es sencillo de modificar, de tal manera que permite adaptarse a los cambios que van surgiendo. El programa ha de poder soportar el crecimiento del mismo.
- **Accesible:** Tanto para la compra como para el mantenimiento. Si se corresponde con un servicio web es conveniente que se pueda acceder a él desde la mayor cantidad posible de ubicaciones y que se encuentre disponible todos los días a todas horas.

*Ilustración 23: Cola Ticketea*

- **Portable:** Que pueda ser usado en distintos dispositivos, sistemas operativos, etc.
- **Usable y fácil de aprender:** El software creado ha de ser “sencillo de usar”. Es conveniente buscar el equilibrio entre usabilidad y eficiencia, ya que suelen ser conceptos que pueden colisionar en alguna ocasión.
- **Adaptado al usuario** – Hay que tener en cuenta quien es el usuario final y como orientar el sistema para él. Esto quiere decir que, aunque las características

mencionadas anteriormente son positivas sobre un producto final, hay que saber cuáles priorizar y cuáles dejar en un segundo plano cuando enfrentamos un proyecto.

Por ejemplo, cuando hacemos una aplicación orientada hacia la población en general, hay que tener en cuenta que dentro de esta población nos encontraremos individuos con conocimientos en informática muy dispares. [Gif Abuelos](#)



*Ilustración 24: Frase pomposa Steve Jobs*

**EJERCICIO 13:** Apoyándote en las características del software y en las características de un buen software da dos ejemplos de buenos softwares y dos ejemplos de malos softwares.

**EJERCICIO 14:** ¿Qué tal software le parece la siguiente página web <http://arngren.net/>? ¿Cuál es la característica en la que más “cojea”?

**EJERCICIO 15:** Indica que características consideras de máxima importancia en los siguientes sistemas software:

- Aplicación solicitar cita médica.
- Un nuevo juego de ordenador que solo se puede jugar en local.
- Un entorno de desarrollo.
- Una aplicación del banco Santander.

## 3.4. CICLOS DE VIDA DEL DESARROLLO SOFTWARE

**Ciclo de vida del desarrollo software:** Conjunto de fases por las que pasa el software desde que nace la idea inicial hasta que es retirado o remplazado (muere). Este proceso ha de asegurarse de generar un producto que dé solución a las necesidades de los usuarios finales del mismo.

El ciclo de vida se divide en etapas o fases, y cada etapa realiza una serie de tareas.

Las etapas más comunes del ciclo de vida del desarrollo software son las siguientes:

- **Análisis:** Se investiga el problema que se quiere resolver. En esta etapa se especifican y analizan los requisitos funcionales y no funcionales del sistema que pretendemos desarrollar. Esta etapa se basa en **el qué**.
- **Diseño:** En este momento ya sabemos qué se va a hacer, ahora tenemos que centrarnos en **cómo** se va a hacer. Se indican las estructuras de datos, la interfaz del usuario, los procedimientos y una parte importante de la ingeniería del software.
- **Desarrollo o codificación:** Consiste en utilizar los modelos creados durante la etapa de diseño para desarrollar los componentes del sistema. Se ha de obtener código ejecutable.
- **Pruebas:** Consiste en comprobar que el software responda/realice correctamente las tareas indicadas en la especificación, se detectan errores y se depuran.
- **Mantenimiento:** El mantenimiento se define como el proceso de control, mejora y optimización del software. La fase de mantenimiento genera nuevas versiones del software.

Cada etapa necesita varios documentos que se generan en la etapa anterior y genera unos documentos de salida. Por ello, es muy importante la tarea de documentación que ha de realizarse en todas las etapas.

Existen muchas clasificaciones para los modelos de desarrollo software, y muchas de ellas se cruzan, no obstante, nosotros vamos a utilizar la siguiente.

### 3.4.1. Modelos Tradicionales

#### 3.4.1.1. Modelo en Cascada

También llamado “Ciclo de Vida Básico”.

Sugiere un enfoque sistemático y secuencial de la ingeniería del software, cada etapa debe esperar al fin de la etapa anterior para empezar, después de cada etapa se realiza una revisión para comprobar si se puede pasar a la etapa siguiente.

Este modelo permite hacer iteraciones, no obstante, si volvemos a una de las etapas anteriores hay que volver a recorrer el resto de las etapas (es muy costoso).

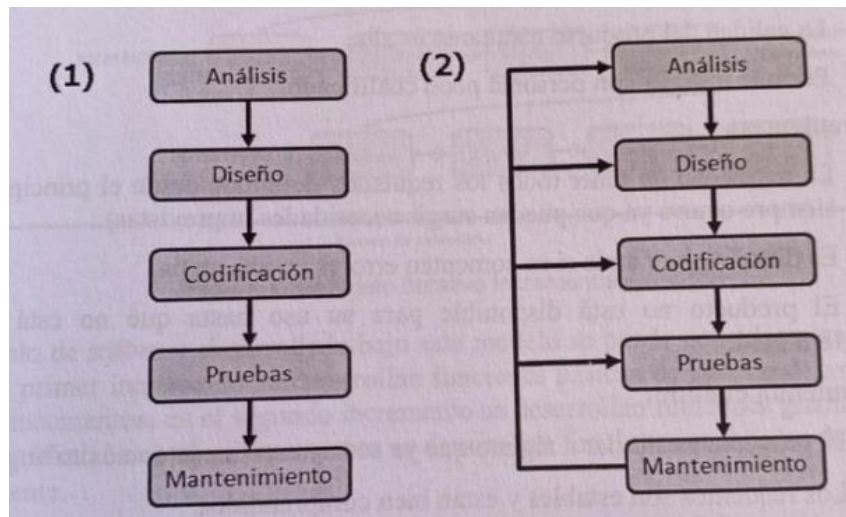


Ilustración 25. Modelo en cascada

- **Ventajas:**
  - Este modelo es fácil de comprender, planificar y seguir.
  - La calidad del producto resultante es alta.
  - Permite trabajar con personal poco cualificado.
- **Desventajas:**
  - Se necesita que todos los requisitos estén definidos desde el principio (si surgen necesidades con las que no se preveía en un principio tenemos un problema).
  - Es difícil y costoso volver atrás si se cometen errores en una etapa.
  - El producto no está disponible para su uso hasta que no está completado y terminado.

- **Se recomienda cuando:**
  - El proyecto es similar a uno que se ha realizado con éxito anteriormente.
  - Los requisitos son estables y están bien comprendidos.
  - Los clientes no necesitan versiones inmediatas.

### 3.4.2. Modelos Evolutivos

En ninguno de los paradigmas anteriores se tiene en cuenta la naturaleza evolutiva del software. Los modelos evolutivos permiten desarrollar versiones cada vez más completas del software. Se adaptan a los cambios de los requisitos del sistema a lo largo del tiempo.

Por otra parte, la competencia en el mercado de software es tan grande que las empresas no pueden esperar a tener un producto completamente finalizado para ponerlo a disposición del usuario final, en su lugar van introduciendo versiones cada vez más completas.

En el modelo en cascada solo se obtiene un producto completo, en el modelo evolutivo se van desarrollando versiones cada vez más completas hasta llegar al producto final.

#### *3.4.2.1. Modelo Incremental*

Consiste en repetir ordenadamente las fases del Modelo Lineal Secuencial, cada vez que se completan las fases se realiza una iteración, al acabar una iteración empieza la siguiente.

Cada iteración tiene como objetivo obtener una versión del producto final (en muchos casos las primeras versiones serán incompletas) que se denominan incrementos. Se va refinando el producto en cada iteración hasta obtener el producto final.

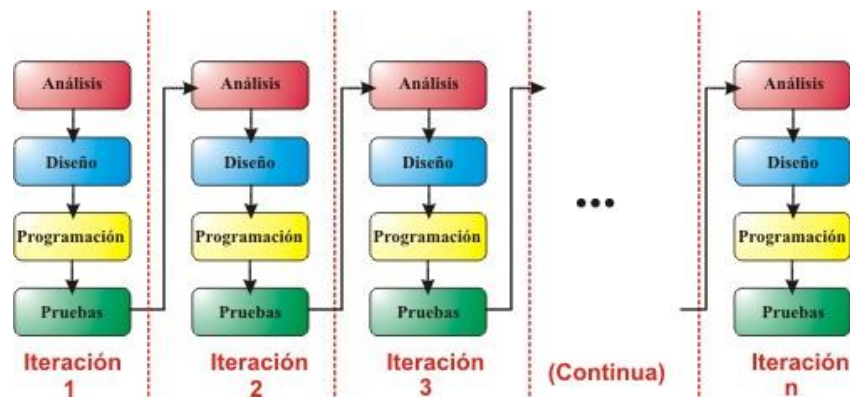


Ilustración 26: Modelo incremental

Es un modelo más flexible, se hace mucho más fácil probar y depurar una iteración que todo el ciclo de vida del desarrollo.

Cada iteración es rígida y no se superpone con otras.

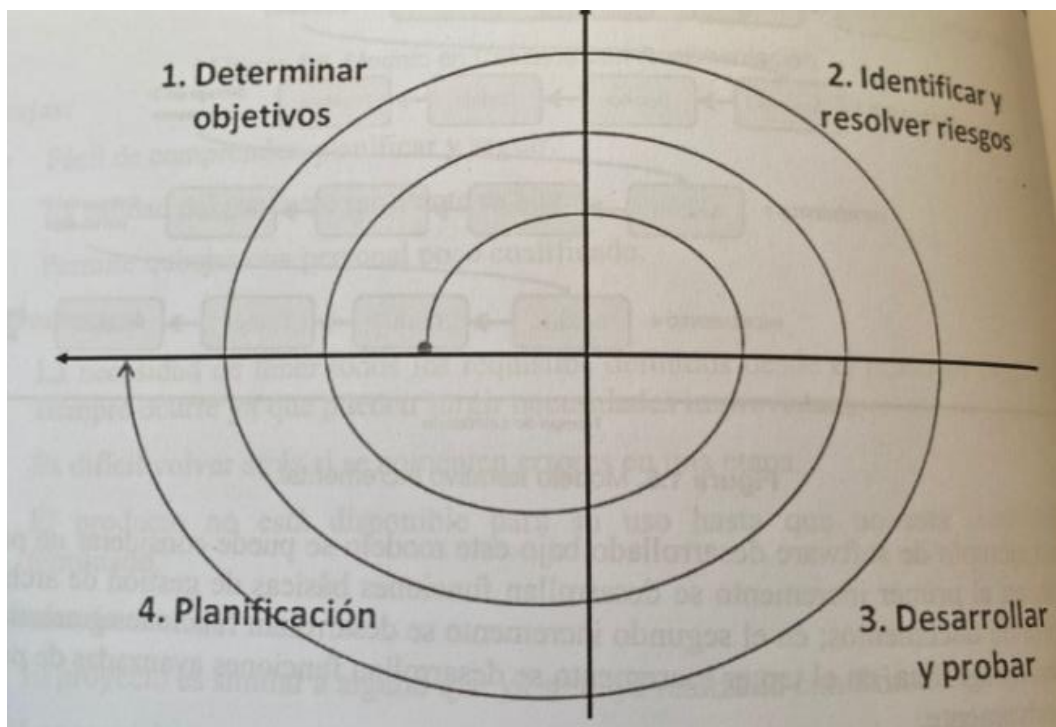
- **Ventajas:**
  - No se necesitan conocer los requisitos al comienzo.
  - Permite la entrega al cliente de partes operativas del software.
  - Las entregas facilitan la realimentación en los próximos entregables.
- **Desventajas:**
  - Es más complejo estimar el esfuerzo y coste final necesario.
  - Se corre el riesgo de que el proyecto se alargue mucho más de lo previsto.
  - No es recomendable para sistemas de tiempo real, alto nivel de seguridad y/o un alto índice de riesgo.
- **Se recomienda cuando:**
  - Los requisitos del sistema son muy volátiles y es posible que haya grandes cambios.
  - Se están probando nuevas tecnologías.

### 3.4.2.2. Modelo en espiral

El proceso de desarrollo de software se representa como una espiral, donde en cada ciclo se desarrolla una parte de este.

Cada ciclo está formado por cuatro fases y cuando termina produce una versión incremental del software con respecto al ciclo anterior.





*Ilustración 27: Modelo en espiral*

Durante los primeros ciclos la versión incremental podrían ser maquetas en papel o modelos de pantallas (propios de interfaz).

Las fases a seguir son las siguientes:

- **Determinar objetivos:** Se identifican los objetivos y se plantean las alternativas existentes para cumplirlos.
- **Análisis de riesgo:** Se evalúan las alternativas en función de los objetivos y las limitaciones. Se identifican los riesgos involucrados y la manera de resolverlos.
- **Desarrollar y probar:** Desarrolla la solución al problema en este ciclo y verifica que sea correcta.
- **Planificación:** Revisa y evalúa todo lo que se ha hecho, y con esto se decide si se continúa, entonces hay que planificar las fases del ciclo siguiente.

Se mostrará a continuación un modelo de 4 ciclos.

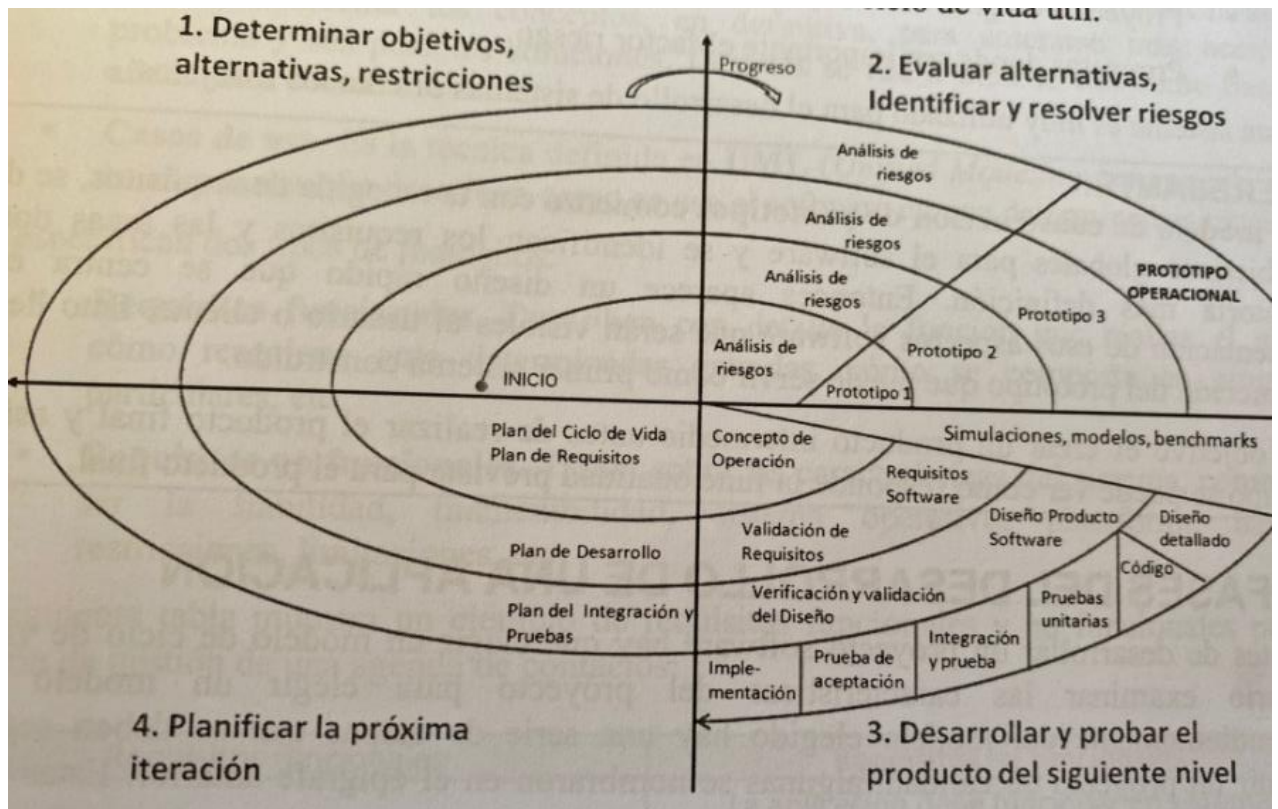


Ilustración 28: Ejemplo de modelo en espiral

- **Ventajas:**
  - No requiere una definición completa de los requisitos para poder ponerse en práctica.
  - Se realiza un análisis de riesgos en todas las etapas.
  - Se reducen los riesgos del proyecto.
  - Se incorporan objetivos de calidad.
- **Desventajas:**
  - Evaluar los riesgos es un proceso complicado.
  - El costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
  - El éxito del proyecto se basa, en gran medida, en la fase de análisis de riesgos.
- **Se recomienda cuando:**
  - El proyecto es de gran tamaño y se realizarán muchos cambios.
  - El factor de riesgo es muy importante en el proyecto.

### 3.4.3. Proceso Unificado

El Proceso Unificado es una metodología de desarrollo de software que está basado en componentes e interfaces bien definidas.

Cada ciclo de desarrollo concluye con una versión entregable del producto.

Cada ciclo consta de cuatro fases:

- **Inicio:** Se define el alcance del proyecto, los objetivos de este, se valoran los factores de riesgo y se desarrollan los casos de uso. Es la única fase que no necesariamente genera una versión ejecutable.
- **Elaboración:** Se completa el análisis de los casos de uso y se define la arquitectura del sistema.
- **Construcción:** Se construye el producto. Está compuesta por un ciclo de varias iteraciones, en el cual se van desarrollando sucesivamente los casos de uso. Este enfoque permite tener en momentos tempranos del desarrollo versiones del sistema que cuenten con los principales casos de uso.
- **Transición:** El producto se convierte en versión beta, se corrigen problemas y se incorporan mejoras sugeridas en la revisión. Culmina con el sistema en fase de producción.

**Hitos:** Los hitos son puntos de control en los cuales los participantes en el proyecto revisan el progreso del proyecto y se toman decisiones de cara a la siguiente iteración o fase.

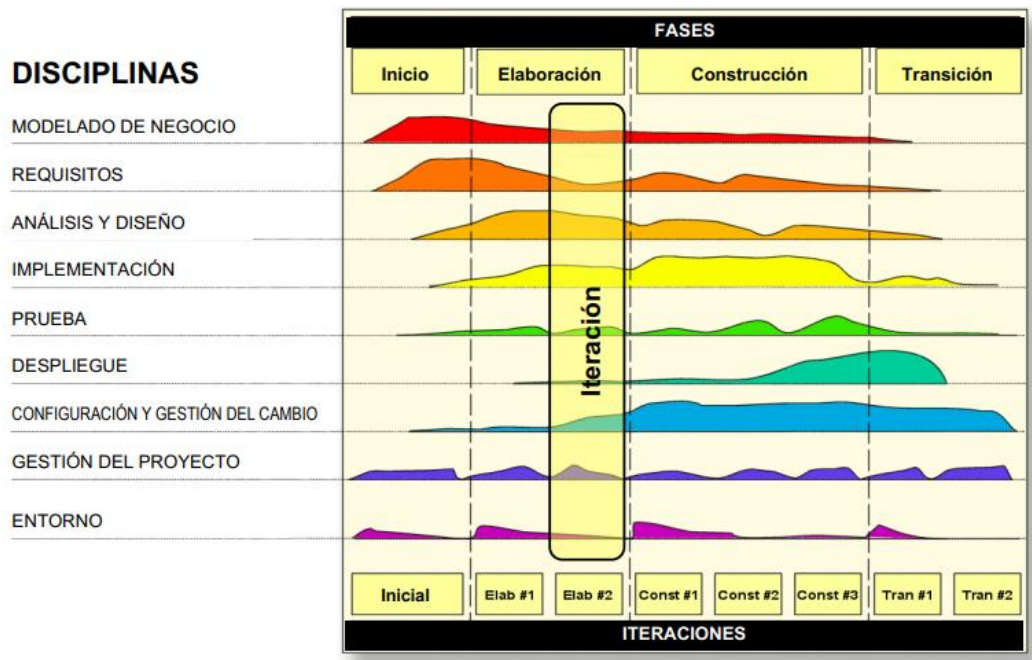


Ilustración 29: Proceso Unificado

### 3.4.4. Metodologías ágiles

Como hemos podido ver, la metodologías que se han explicado anteriormente se centran en el control del proceso, estableciendo rigurosamente las actividades, herramientas y notaciones al respecto, dado estas reglas estas metodologías se caracterizan por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

El mercado de la informática resulta tremendamente competitivo, por tanto, para los trabajadores, la exigencia y la presión cada vez es mayor.

El cliente solicita un producto de alta calidad, que se desarrolle en poco tiempo y dentro de un presupuesto ajustado, esto obviamente es prácticamente imposible, no obstante, para poder competir, las empresas se ofrecen a prestar este servicio.



*Ilustración 30: Regreso al futuro*

Dada esta situación, las metodologías ágiles surgen como una posible respuesta metodológica. En febrero de 2001, tras una reunión celebrada por 17 expertos de la industria del software (dentro de los que se encontraban algunos de los creadores o impulsores de las metodologías de software) en Utah, nace el término 'ágil' aplicado al desarrollo de software. Tenían como objetivo establecer los valores y principios que permitirían desarrollar software rápidamente y que a su vez permitiera aplicar de manera eficaz los cambios que puedan surgir a lo largo del proyecto.



*Ilustración 31: Reunión metodologías ágiles*

## 12 principios del Manifiesto Ágil:

- 1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

- 2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- 3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- 4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- 5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- 6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- 7. El software funcionando es la medida principal de progreso.
- 8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- 9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- 10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- 11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- 12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Este tipo de metodologías han llevado a un problema, una empresa dedicada al desarrollo de software que no opte por una metodología ágil difícilmente podrá sobrevivir al entorno actual, pero, sobre todo, a los nuevos tiempos que se avecinan.



Los procesos ágiles también defienden que es más importante que el software funcione que la documentación exhaustiva.

### Opinión:

*En muchos casos el concepto metodología ágil se ha utilizado para enmascarar la explotación laboral, es importante tener en cuenta los derechos de los trabajadores en el país en el que estamos para evitar ser víctimas de esto.*

[Estatuto de los trabajadores](#)

*El cliente no siempre tiene la razón, los desarrollos deben tener unas fases y llevan un tiempo, nosotros como trabajadores del sector debemos defendernos y defender nuestro trabajo, solicitando los costes correspondientes al trabajo y tiempo que se le dedica al producto, y no caer en empleos que nos exijan unas condiciones alienantes.*

*Las metodologías pueden concluir en un proyecto mejor o peor, pero nunca deben derivar en unas malas condiciones laborales.*

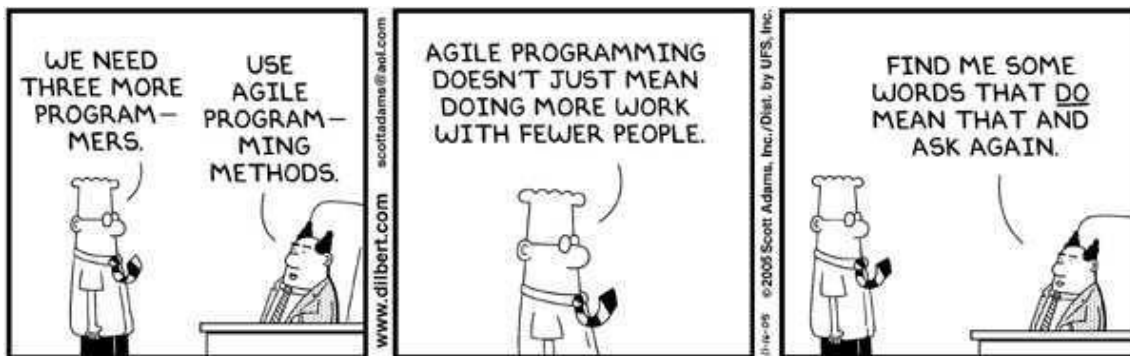


Ilustración 32: Dilbert metodologías ágiles

Otro punto de vista (más positivo): [Otra opinión](#)

#### 3.4.4.1. Programación extrema



Ilustración 33: Dilbert programación extrema

Para profundizar más... [Programación extrema](#)

#### 3.4.4.2. Scrum

Scrum es la metodología ágil más utilizada.

En 2017, supuso el 58% del total de uso de todas las metodologías, muy por delante de cualquier otra.

Esta metodología está basada principalmente en la creación y asignación de cometidos.

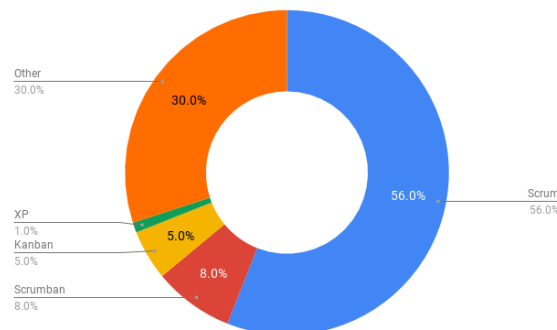


Ilustración 34: Porcentaje metodologías usadas

#### 3.4.5. MODELO DE CONSTRUCCIÓN DE PROTOTIPOS

Es más una técnica que un modelo propio y se utiliza cuando la especificación de requisitos es complicada



**Especificación de requisitos:** El objetivo principal de la Especificación de Requisitos del Sistema (ERS) es servir como medio de comunicación entre clientes, usuarios, ingenieros de requisitos, desarrolladores y demás participantes.

En la ERS deben recogerse tanto las necesidades de clientes y usuarios (necesidades del negocio, también conocidas como requisitos de usuario, requisitos de cliente, necesidades de usuario, etc.) como los requisitos que debe cumplir el sistema software a desarrollar para satisfacer dichas necesidades (requisitos del producto, también conocidos como requisitos de sistema o requisitos software).

**Prototipo:** Es un modelo experimental de un sistema o de un componente de un sistema. Nos permite hacernos una idea del aspecto que tendrá el sistema final.

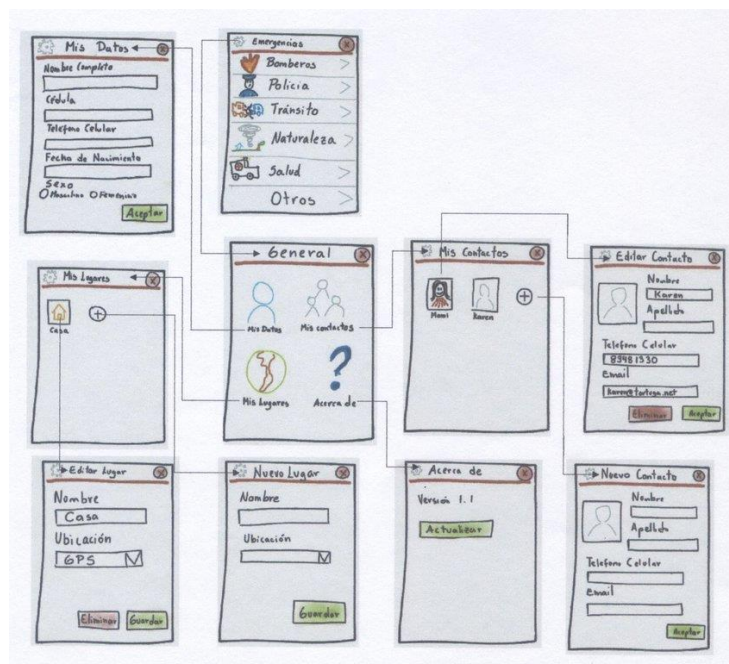


Ilustración 35: Prototipado en papel

Los prototipos son una herramienta para el desarrollo software, no corresponden en sí mismo un modelo como tal.

El cliente, a menudo, define un conjunto de objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida. En otros casos el responsable de desarrollo no está seguro de la eficacia de un algoritmo, etc.

En estos casos el modelo de construcción de prototipos ayuda a la especificación de requisitos.

Cuando termina el proceso, sabemos cómo debe funcionar el sistema, entonces se diseña, se codifica y se prueba.

Lo ideal es que el prototipo sirva como un mecanismo para identificar los requisitos del software.

La elaboración de un prototipo correspondería con la fase de análisis, el resto del desarrollo software dependerá del modelo que se aplique.

**EJERCICIO 16:** Esta tarea se puede realizar en grupos de dos o de manera individual.

**1º:** Desarrollad una idea de una página web o una aplicación de móvil. Esta página ha de contemplar una gestión de usuarios (registrar usuario, login, eliminar, olvido de contraseña, etc...) además de su funcionalidad.

**2º:** Realizad un prototipo a papel de esa idea.

**3º:** Realizad un prototipo informático de esa idea, usando cualquiera de las herramientas de prototipado que se encuentran en la web (Framer, InVision, Bubble ...).

**4º:** Se realizará una breve exposición utilizando el formato Petxa Kutxa ( [Formato Petxa Kutxa](#) ) de cada grupo explicando la página web desarrollada y la tecnología usada para el prototipado.

**EJERCICIO 17:** Crea una tabla comparativa con las ventajas y las desventajas de los siguientes modelos (las ventajas y desventajas pueden ser subjetivas si está bien explicado el motivo).

- En cascada
- Scrum
- Proceso unificado
- En espiral
- Incremental

Para ello debéis de comprender la información de los apuntes e investigar un poco por vuestra cuenta. Aquí os doy una página para que os de algo de inspiración: [Tabla comparativa modelos](#).

**EJERCICIO 18:** Crea una historia de desarrollo de un proyecto en una metodología clásica, una evolutiva, una centrada en la reutilización y una metodología ágil (Historias que no sean muy cortas (no menos de un tercio folio) ni muy largas (no más de medio

folio) y que se centren en las características importantes de la metodología. Lo que se valorará es la comprensión del alumno de la metodología y la capacidad de visualizarla en un entorno coherente.

### 3.5. Herramientas de apoyo al desarrollo del software

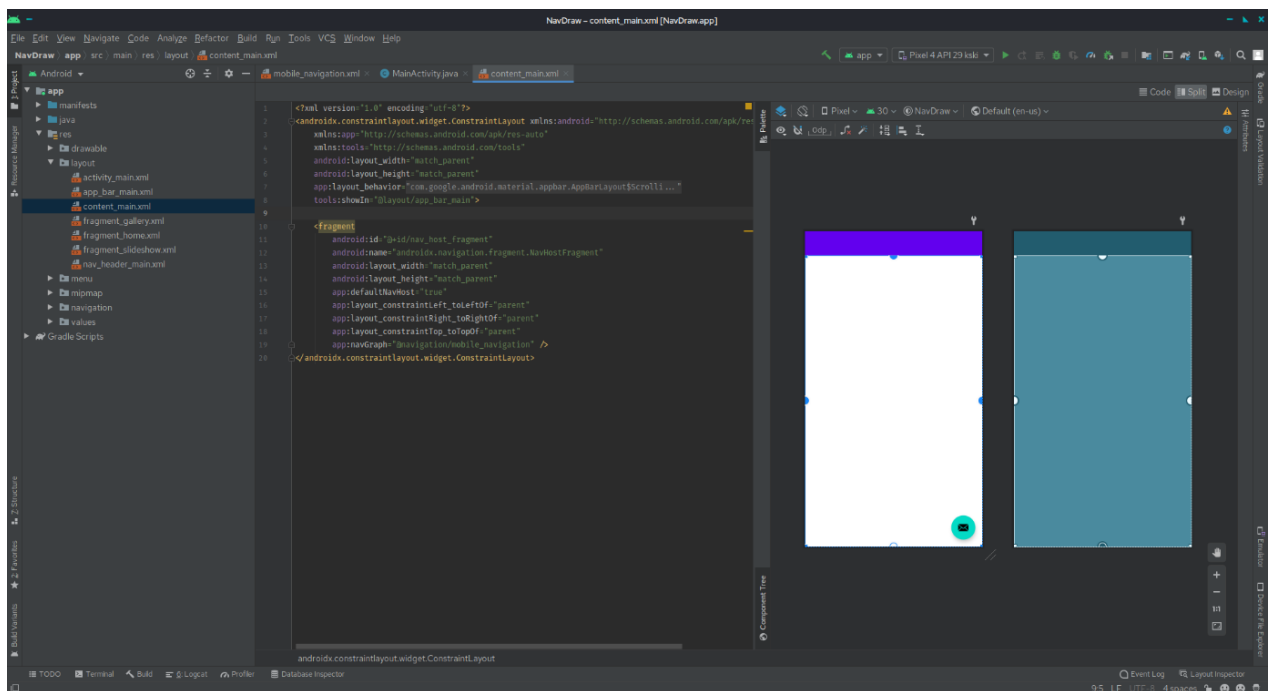


Ilustración 36: Android Studio

**Framework:** Un Framework es un marco de trabajo, o de aplicaciones. Una de las definiciones más aceptadas para framework concluye en que este es un esquema para el desarrollo y/o la implementación de una aplicación.

**Técnicas de 4ª generación:**

Basado en la utilización de un amplio espectro de herramientas para facilitar el trabajo al ingeniero del software en:

- **Especificación de requisitos**



*Ilustración 37: Logo Modern Requirements*

- **Análisis y Modelado**



*Ilustración 38: Logo Visual Paradigm*

- **Diseño**



*Ilustración 39: Logo Visual Paradigm*

- **Generación de código**



*Ilustración 40: Logo Python*

- **Pruebas**



Ilustración 41: Logo Postman

- **Documentación**



Ilustración 42: JavaDoc logo

- **Construcción de prototipos**

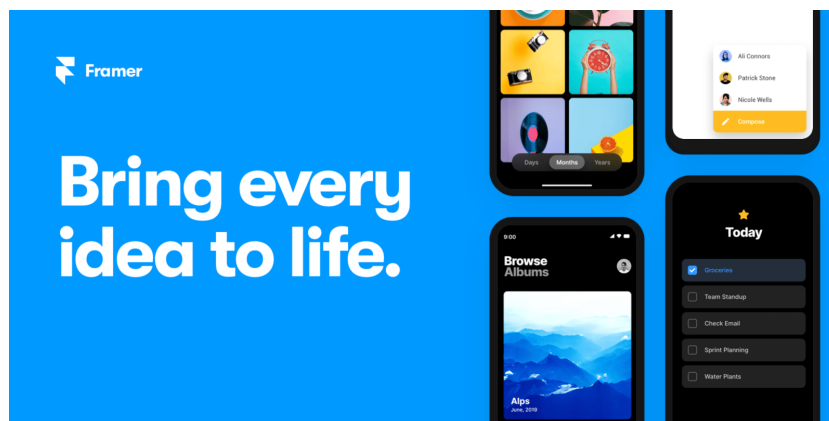


Ilustración 43: Framer

- **Control de versiones**

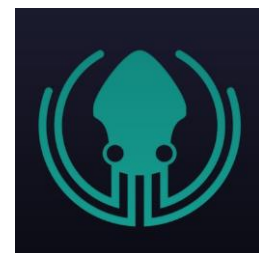


Ilustración 44: Logos git github gitkraken

API de ejemplo: <https://freegeoip.app/>

Dirección a la que llamar: <https://freegeoip.app/json/>

<https://pokeapi.co/docs/v2#pokemon-section>

<https://www.frankfurter.app/docs>

## 4. LENGUAJES DE PROGRAMACIÓN

**Un lenguaje de programación:** es un conjunto de caracteres, las reglas para la combinación de esos caracteres y las reglas que definen sus efectos cuando son ejecutadas por un ordenador. Permiten crear programas.

Un lenguaje de programación está formado por:

- **Alfabeto o vocabulario:** Conjunto de símbolos permitidos. Se combinan para producir elementos del lenguaje (i y f generan if).
- **Sintaxis:** Reglas que indican cómo se combinan los elementos del lenguaje para producir expresiones (if (variable == 10) Y (variable2 == 5)).
- **Semántica:** Reglas que determinan el significado de cualquier expresión de lenguaje y que indica como se convierten en CPU.

Dos programas podrían hacer lo mismo (semántica) pero los símbolos utilizados para escribir el programa serían diferentes (sintaxis).

Un compilador puede validar la sintaxis, pero no encontrará todos los errores semánticos.

El ordenador solo entiende el lenguaje máquina, por tanto, el código escrito en un lenguaje de programación necesita de una traducción (compilación o interprete) para su ejecución.

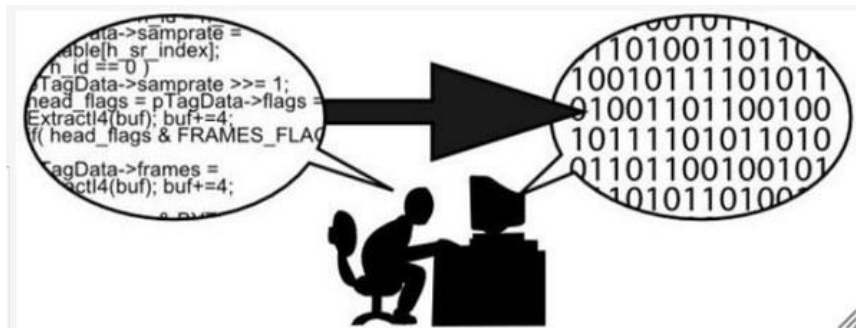


Ilustración 45: Compilación

En función de las características del programa que queremos desarrollar y del entorno en el que se va a ejecutar elegiremos un lenguaje de programación u otro.

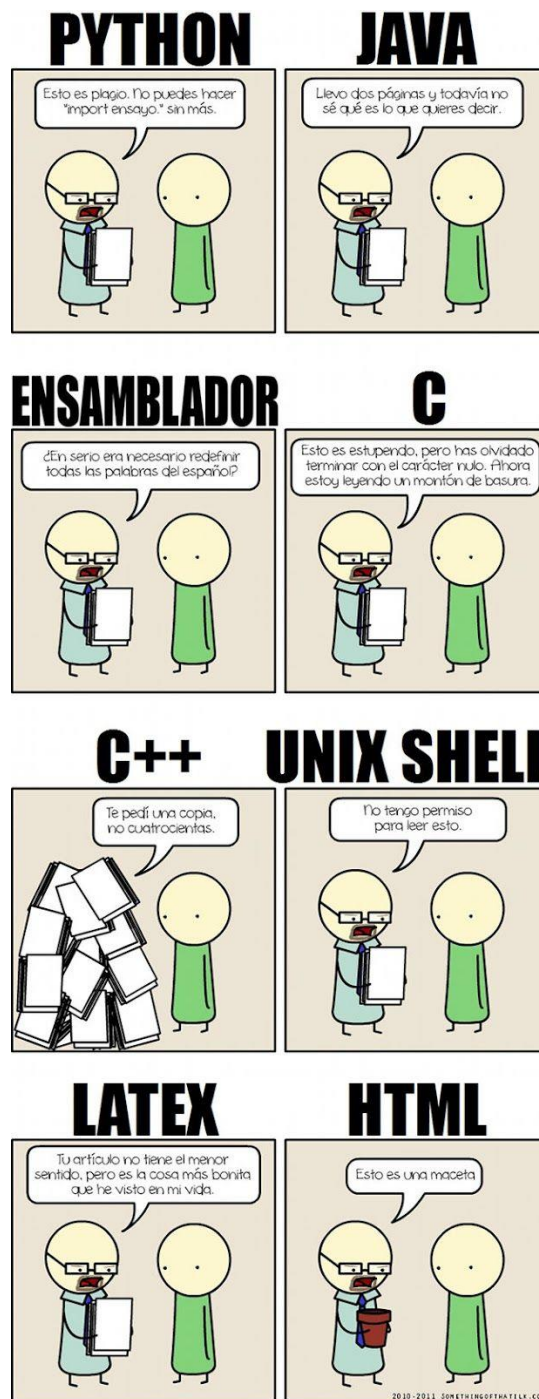


Ilustración 46: Comic lenguajes de programación



## 4.1. Lenguajes de programación según su nivel de abstracción

- **Lenguaje máquina:** Es el único lenguaje que entiende directamente el ordenador y es único para cada procesador. Las instrucciones están formadas por cadenas de ceros y unos.

El resto de los lenguajes necesitan ser traducidos a lenguaje máquina para poder ejecutarse.

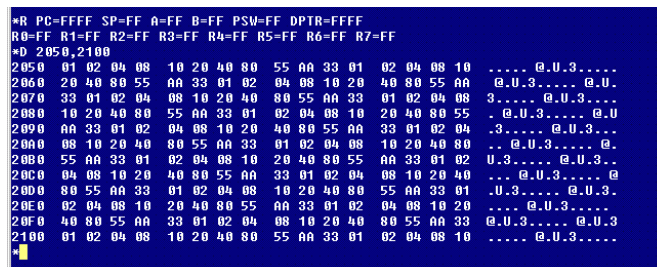


Ilustración 47: Lenguaje máquina

- **Lenguaje ensamblador:** Creado para facilitar la labor del programador, pero aún muy difícil de usar.

Utiliza nombres nemotécnicos y las instrucciones trabajan directamente con registros de memoria física de la máquina.

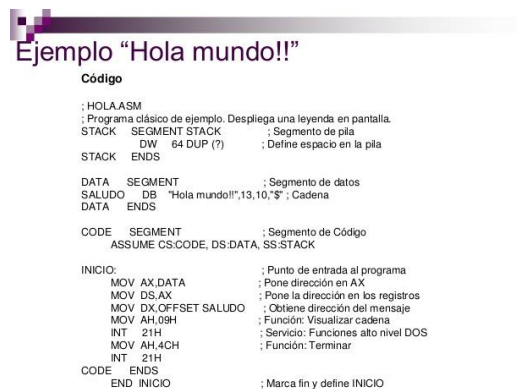


Ilustración 48: Lenguaje ensamblador

- **Lenguaje de nivel medio:** Combinan las características de los lenguajes de alto nivel y bajo nivel. C es un lenguaje de programación de nivel medio.

```

1  #include <stdio.h>
2
3  void leerint(int *a);
4  int sumar(int a,int b);
5  void dividir(int a, int b, int *coc,int *resto);
6
7  int main(){
8      int a,b, coc, resto;
9      leerint(&a);
10     leerint(&b);
11     printf("%d + %d = %d\n",a,b,sumar(a,b));
12     dividir(a,b,&coc,&resto);
13     printf("%d / %d = %d Resto: %d \n",a,b,coc,resto);
14     return 0;
15 }
16
17 void leerint(int *a){
18     printf("Introduce un numero: \n");
19     scanf("%d",a);
20 }
21
22 int sumar(int a,int b){
23     return (a + b);
24 }
25
26 void dividir(int a, int b, int *coc,int *resto){
27     *coc = (int) a / b;
28     *resto = a % b;
29 }
30

```

Ilustración 49: Lenguaje de nivel medio

- **Lenguaje de alto nivel:** más cercano la lenguaje de comunicación humano (inglés). Basado en código. Dentro de los lenguajes de alto nivel también se puede encontrar un nivel medio.

```

Principal.java x
1  package com.carlos;
2
3  /**
4   * Esta es la clase principal del programa.
5   * @author carlos
6   * @version 1.0
7   * @since 19/08/2019*/
8
9  public class Principal {
10
11     public enum Dias {Lunes, Martes, Miércoles};
12
13     public static void main(String[] args) {
14
15         int num1 = 9;
16         float num2 = 9.3f;
17         float suma = num1 + num2;
18
19         System.out.println(suma);
20     }
21 }

```

Ilustración 50: Lenguaje de alto nivel

- **Lenguaje visual:** apoyo de técnicas visuales y herramientas CASE para el desarrollo del software.

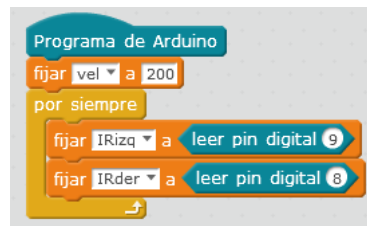


Ilustración 51: Lenguaje visual

**EJERCICIO 20:** Realiza un pequeño juego en scratch usando algunas estructuras de control (para esto puedes apoyarte de algún pequeño tutorial como por ejemplo este [Tutorial persecución](#)). ¿A qué se traduciría directamente en Java las estructuras que has usado en Scratch? ¿En qué tipo de lenguaje de los vistos anteriormente crees que se engloba Scratch?

**EJERCICIO 21:** Viendo los tipos de lenguajes de programación ¿Hacia qué fin crees que han evolucionado? ¿Qué limitaciones le ves a la programación visual respecto a la de alto nivel? ¿y qué ventajas?

## 4.2. Lenguajes de programación según forma de ejecución

Los lenguajes de alto nivel tienen que ser traducidos a código máquina para que lo pueda utilizar el ordenador.

- **Lenguajes compilados:** Un compilador se trata de un programa que puede leer un programa escrito en un determinado lenguaje (lenguaje fuente) y traducirlo a un programa equivalente en otro lenguaje (lenguaje destino).  
El compilador dará errores si el programa en el lenguaje fuente no está bien escrito.
- **Lenguajes interpretados:** Otra alternativa para traducir los programas escritos en un lenguaje de alto nivel son los intérpretes.  
En vez de producir un programa destino, el intérprete da la apariencia de ejecutar directamente las operaciones en el programa fuente con las entradas del usuario.  
Cada vez que se ejecuta una instrucción se debe interpretar y traducir a lenguaje máquina.  
El programa destino en lenguaje máquina que genera el compilador suele ser más rápido que un intérprete al momento de asignar las entradas a las salidas.

El intérprete elimina la necesidad de realizar una compilación después de cada modificación del programa.

Los lenguajes interpretados son más flexibles con los entornos de programación y depuración.

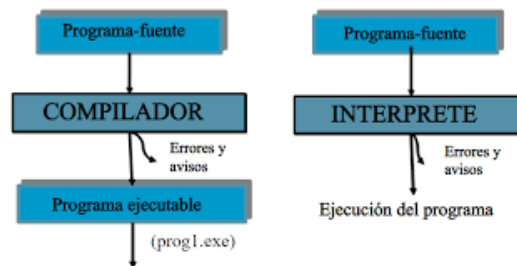


Ilustración 52: Compilación e interpretación

### 4.3. Lenguajes de programación según su paradigma de programación

- **Lenguaje de programación estructurada:** Algunos lenguajes de programación estructurada son C, Pascal, Fortran, BASIC

La programación estructurada es un paradigma de programación basado en utilizar funciones o subrutinas, y únicamente tres estructuras de control:

- **Secuencia:** Ejecución de una sentencia tras otra.
- **Selección o condicional:** Ejecución de una sentencia o conjunto de sentencias, según el valor de una variable booleana.
- **Iteración (ciclo o bucle):** Ejecución de una sentencia o conjunto de sentencias, mientras una variable booleana sea verdadera.

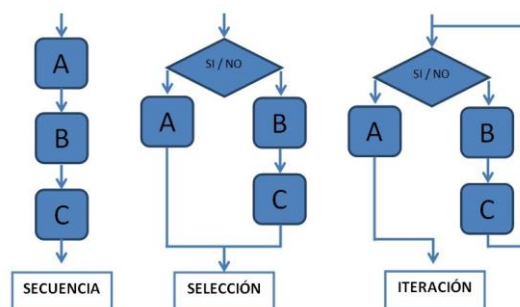


Ilustración 53: Programación estructurada

En la programación estructurada todo el código se concentra en un único bloque, por tanto, para problemas grandes el manejo es muy complejo.

La Programación estructurada evolucionó hacia la Programación modular, que divide el programa en trozos de código llamados módulos con una funcionalidad concreta, que podrán ser reutilizados.

- **Lenguaje de programación orientada a objetos:** Algunos lenguajes de programación orientados a objetos son Java, C++, C#, Ada, etc.

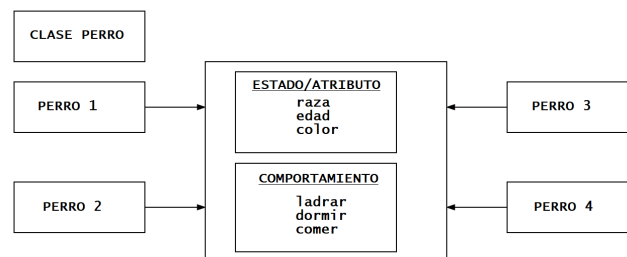
Los programas se componen por un conjunto de objetos no por un conjunto de instrucciones o módulos.

La programación Orientada a objetos se define como un paradigma de la programación, una manera de programar específica, donde se organiza el código en unidades denominadas **clases**, de las cuales se crean **objetos** que se relacionan entre sí para conseguir los objetivos de las aplicaciones.

Las clases son plantillas a partir de las cuales se crearán objetos.

Los objetos son instancias de la clase.

La programación orientada a objetos favorece la reutilización del software, el trabajo en equipo y el mantenimiento del software.



*Ilustración 54: Programación orientada a objetos*

**EJERCICIO 22:** ¿Qué objetos crees que puede haber en una aplicación como Insagram? Indica algunos atributos y métodos que crees que tendrá.

**EJERCICIO 23:** ¿Qué objetos crees que puede haber en una aplicación como la del banco Santander? Indica algunos atributos y métodos que crees que tendrá.

**EJERCICIO 24:** ¿Qué objetos crees que puede haber en Youtube? Indica algunos atributos y métodos que crees que tendrá.

- **Lenguaje de programación orientado a eventos:** Visual Basic, Javascript.

La Programación orientada a eventos es un paradigma de programación en el que la estructura y la ejecución de los programas van determinados por los

sucesos o acciones que ocurren en el sistema, definidos por el usuario o por el propio sistema.

```

<div id="content">
  <div class="FullContentDiv" id="ExternalIframe"></div>
  <div id="GridViewContainer">
    <div id="bcbc">
      <div class="pageitem" id="breadcrumbsID">
        <span class="blue Pointer" onclick="javascript:initpage(161540)" ">
          <b>Sign - 161540</b>
          <b>></b>
        </span>
        <span class="blue Pointer" onclick="javascript:initpage(164725)" ">...</span>
      </div>
    </div>
  </div>
</div>

```

Ilustración 55: Programación orientada a eventos

**EJERCICIO 25:** Enumera eventos que pueden ocurrir en una interfaz web a partir de los cuales se puede disparar una acción.

## 4.4. Otros tipos de lenguajes de programación

- **Lenguaje del lado del cliente:** Html, Xhtml, CSS, JavaScript (Ajax), jQuery

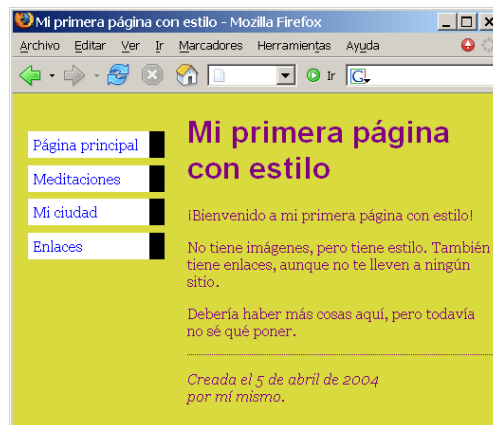


Ilustración 56: Lenguaje lado del cliente



Ilustración 57: CSS

- **Lenguaje del lado del servidor:** PHP, Java (JSP, Servlets), C++, Python

```
main.cpp
1  /*1. Escribe un programa que lea de la entrada estándar dos números y muestre
2  en la salida estándar su suma, resta, multiplicación y división. */
3
4  #include<iostream>
5
6  using namespace std;
7
8  int main(){
9      int n1,n2, suma = 0, resta = 0, multiplicacion=0, division=0;
10
11      cout<<"Digite un numero: "; cin>>n1;
12      cout<<"Digite otro numero: "; cin>>n2;
13
14      suma = n1 + n2;
15      resta = n1 - n2;
16      multiplicacion = n1 * n2;
17      division = n1 / n2;
18
19      cout<<"\nLa suma es : "<<suma<<endl;
20      cout<<"La resta es: "<<resta<<endl;
21      cout<<"La multiplicacion es "<<multiplicacion<<endl;
22      cout<<"La division es: "<<division<<endl;
23
24      return 0;
25  }
```

input

Digite un numero:

Ilustración 58: Lado del servidor

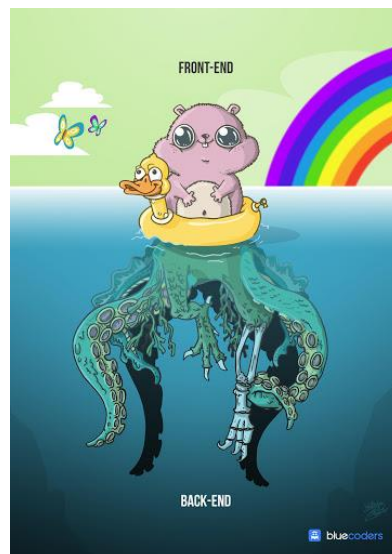


Ilustración 59: Front-End vs Back-End

**Lenguaje de comunicación con el SGBD:** SQL, SQL/PSM, PL/SQL, MySQL's procedural language.

Todos los SGBD ofrecen lenguajes apropiados para realizar la comunicación entre los SGBD y los usuarios.

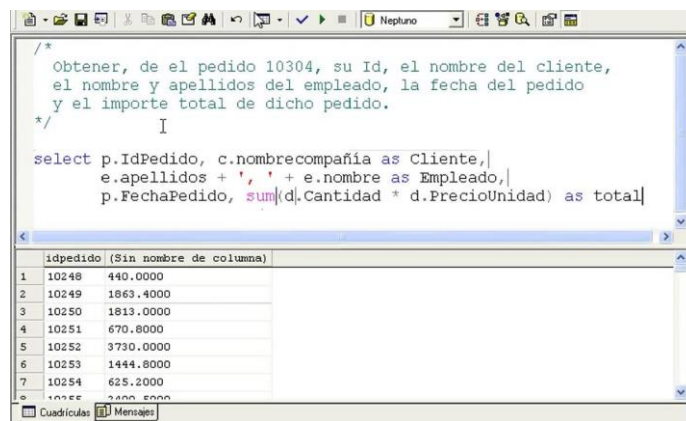


Ilustración 60: Lenguaje de comunicación con el SGBD

**EJERCICIO 26:** Que actividades puede llevar a cabo sobre una base de datos un lenguaje como SQL.

Lenguajes de programación más usados: [Lenguajes de programación más usados](#)

**EJERCICIO 27:** Elije un lenguaje de programación que no conozcas y realiza un manual en el que expliques los principios básicos del lenguaje de programación (5 páginas +-). Introducción, descripción del lenguaje, principios básicos (por ejemplo: estructura principal, tipos de datos, condicionales, bucles, etc.), como se realiza la compilación/interpretación, ejemplos de uso del sistema.



## 5. FASES EN EL DESARROLLO Y EJECUCIÓN DEL SOFTWARE

Independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad. Estas etapas son:

- **Análisis:** Se especifican los requisitos del sistema.
- **Diseño:** Se divide el sistema en partes y se determina la función de cada una.
- **Codificación:** Se elige un Lenguajes de Programación y se codifican los programas.
- **Pruebas:** Se prueban los programas para detectar errores y se depuran.
- **Implantación / explotación / producción:** Instalamos, configuramos y probamos la aplicación en los equipos del cliente.
- **Mantenimiento:** Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.
- **Documentación:** de todas las etapas, se documenta y guarda toda la información.

### 5.1. Análisis

Es la primera etapa que debemos llevar a cabo en todo proyecto de desarrollo software y es la que se basa en entender el problema al que se debe dar solución.

La fase de análisis da respuesta a la pregunta **¿Qué hay que hacer?**

En esta etapa se especifican y analizan los requisitos funcionales y no funcionales del sistema que pretendemos desarrollar.

Esta fase no es sencilla, es muy común que el cliente no tenga claros los requisitos, cambien los requisitos establecidos inicialmente, pueden generarse malentendidos con el cliente por la falta de conocimientos informáticos, etc.

Algunas técnicas utilizadas en esta fase son:

- **Entrevistas:** Es la técnica más tradicional. Consiste en una conversación con el cliente.



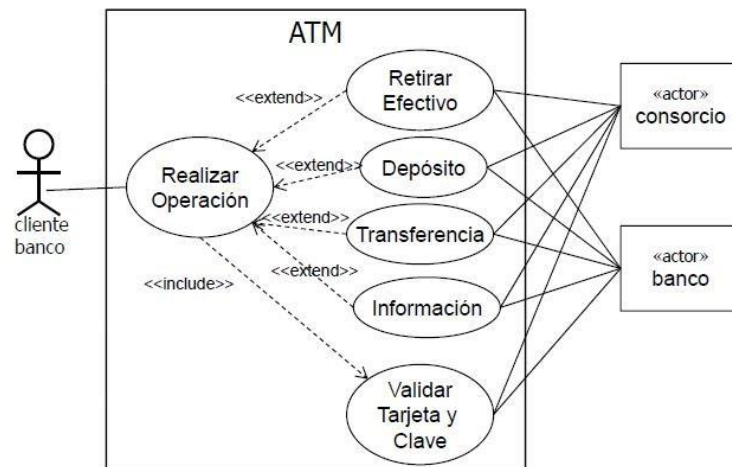
- **Brainstorming:** Consiste en una reunión en la cual se propondrán muchas ideas desde diferentes puntos de vista. Es adecuado utilizarla al principio del proyecto para proponer una gran variedad de soluciones a un problema.



- **Prototipos:** Consiste en el desarrollo de una versión inicial del sistema. Se utiliza para poder dar una visión más cercana a la solución del problema.



- **Diagramas de casos de uso:** Requiere conocimientos de UML, se basa en escenarios que describen un software en una determinada situación.



Todo este proceso se enfoca en la búsqueda de dos tipos de 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.

Ejemplo (aplicación de compra/venta de una empresa):

- El sistema ha de guardar la información de los productos existentes en el almacén, indicando fecha de compra, categoría, marca, unidades existentes, fecha de caducidad, precio ...
- El sistema deberá generar un ticket de venta indicando, fecha, vendedor, precio total de la venta ...
- El sistema tendrá varios niveles de acceso basándose en el perfil de cada usuario: Dirección, Administrador y vendedor.
- El sistema ha de solicitar una contraseña para que los usuarios puedan interactuar con él.
- El sistema ha de generar reportes mensuales de ventas.
- Los reportes generados por el sistema únicamente podrán ser ejecutados y obtenidos por los niveles de administración y dirección.
- La pantalla de inventario muestra los productos por código, categoría y producto.
- Los campos donde se especifican los precios de venta solo han de aceptar valores numéricos con hasta dos decimales.
- El sistema arrojará un mensaje de alerta cuando el producto esté cercano a su fecha de caducidad.

- El sistema enviará una alerta cuando haya modificaciones de los usuarios.
- **No funcionales:** Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.
  - El sistema deberá funcionar correctamente en los siguientes sistemas operativos: Windows 8, Windows 10 y Linux.
  - El sistema debe ser capaz de operar con, por lo menos, diez sesiones concurrentes a la vez.
  - Se deben generar respaldos semanales de la base de datos y estos deberán almacenarse en una ubicación diferente.
  - El usuario ha de poder aprender a utilizar el sistema en un tiempo no mayor a 8 horas.
  - El sistema contará con manuales de usuarios debidamente documentados.
  - Para almacenar los datos se usará un sistema gestor de base de datos.
  - Se usará un lenguaje orientado a objetos para el desarrollo de la aplicación.
  - La interfaz del usuario ha de ser intuitiva y fácil de manejar.

**Ejercicio 28:** Indica cuáles de los siguientes requisitos son funcionales y cuales no funcionales y explica por qué.

- El sistema enviará un correo electrónico al usuario cuando este se registre.
- El sistema ha de cumplir la ley de protección de datos.
- No se permitirá el registro de usuarios con datos obligatorios incompletos.
- El sistema debe tardar menos de 2 segundos en realizar cada petición.
- El sistema deberá funcionar correctamente en cualquier versión de Windows.
- El sistema deberá permitir agregar nuevos clientes.
- La aplicación exigirá 1GB mínimo de disco.

Lo fundamental es la buena comunicación entre el analista (quien analiza los requisitos) y el cliente (quien “encarga” el desarrollo software) 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). Según la última versión del estándar 830 [IEEE, 1998] la estructura del documento ERS es la siguiente:

<https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>

## 1. Introducción

- 1.1. Propósito . . . . .
- 1.2. Ámbito del Sistema . . . . .
- 1.3. Definiciones, Acrónimos y Abreviaturas . . . . .
- 1.4. Referencias . . . . .
- 1.5. Visión General del Documento . . . . .

## 2. Descripción General

- 2.1. Perspectiva del Producto . . . . .
- 2.2. Funciones del Producto . . . . .
- 2.3. Características de los Usuarios . . . . .
- 2.4. Restricciones . . . . .
- 2.5. Suposiciones y Dependencias . . . . .
- 2.6. Requisitos Futuros . . . . .

## 3. Requisitos Específicos

- 3.1. Interfaces Externas . . . . .
- 3.2. Funciones . . . . .
- 3.3. Requisitos de Rendimiento . . . . .
- 3.4. Restricciones de Diseño . . . . .
- 3.5. Atributos del Sistema . . . . .
- 3.6. Otros Requisitos . . . . .

## 4. Apéndices

**Ejercicio 29:** Imagina como sería una etapa de análisis de una empresa con las siguientes características.

- Un cliente que tiene una tienda de lanas en la ciudad de Bilbao quiere hacer una página web que además de mostrar sus productos y presentar su tienda permita hacer ventas online.

- El cliente no tiene ningún conocimiento de informática.
- Solicita una reunión con una empresa de desarrollo software llamada Informatiza 5.0.
- La empresa tiene 5 empleados.
- A partir de aquí imagina como sería la etapa el análisis del proyecto...

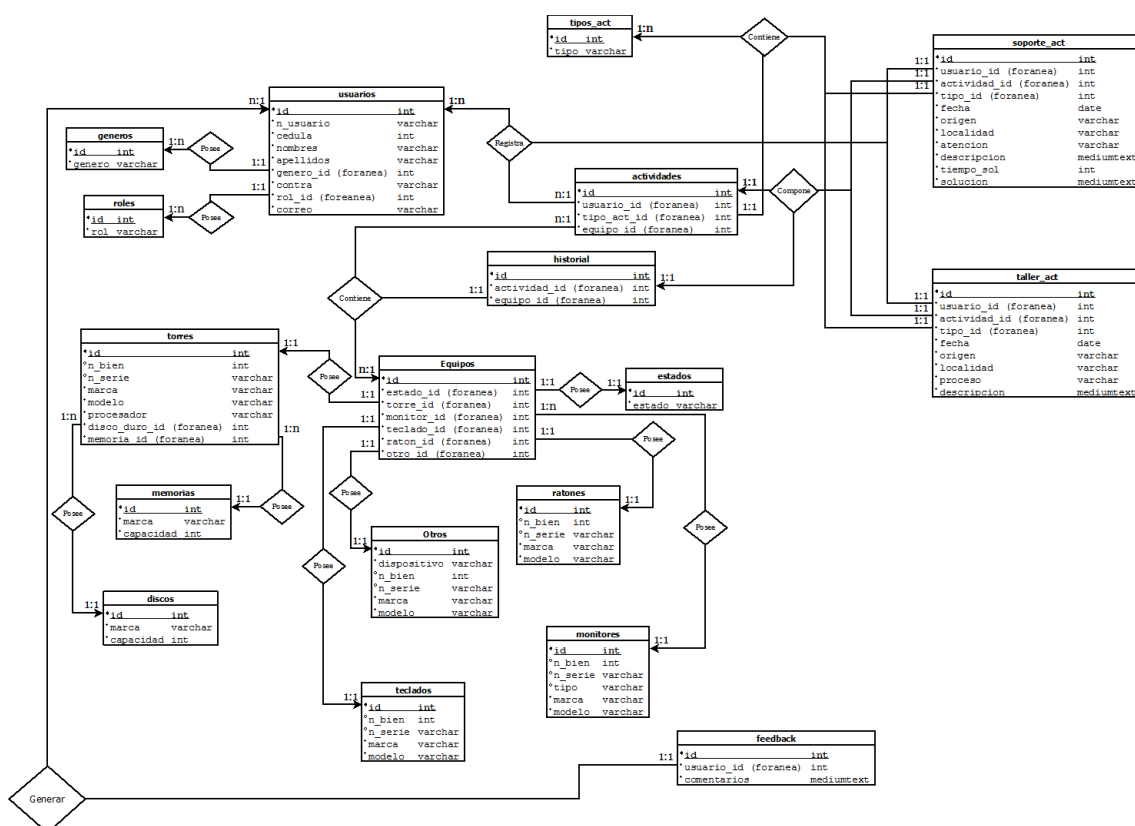
## 5.2. Diseño

Ahora ya sabemos lo que hacer, el siguiente paso es ¿Cómo hacerlo?

Para dar este paso se procede a traducir los requisitos en una representación software.

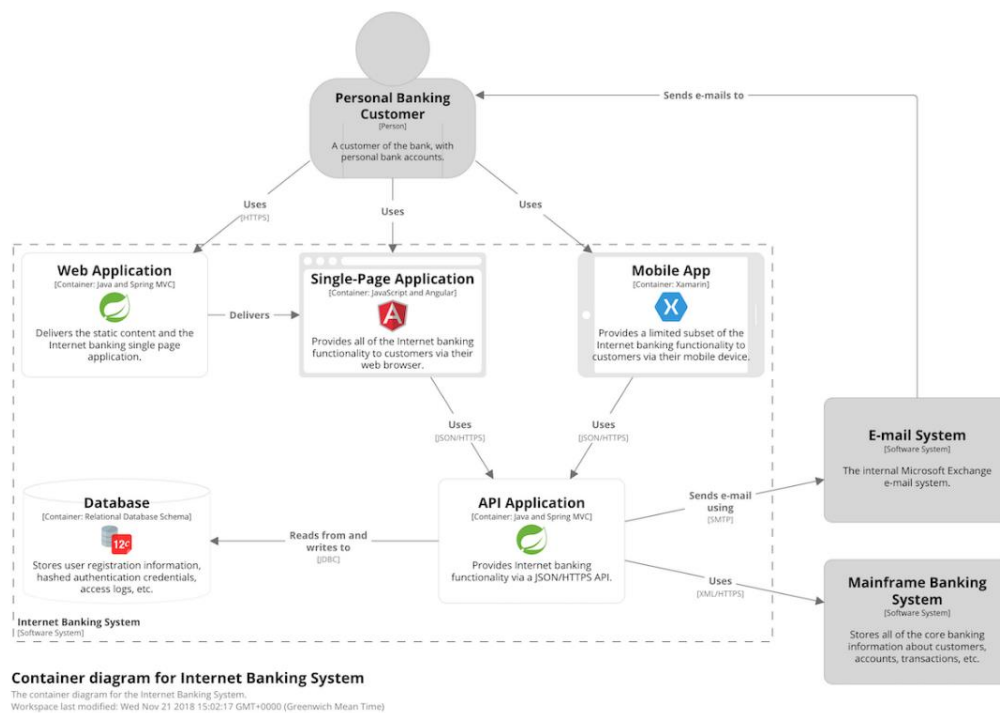
Algunos de los elementos que se consideran en la etapa de diseño son los siguientes:

- **Diseño de datos**



**Ejercicio 30:** Indica toda la información que puedas extrapolar de este diagrama.

- Diseño arquitectónico



**Ejercicio 31:** Indica toda la información que puedas extrapolar de este diagrama.

## REFERENCIAS

<https://lenguajesdeprogramacion.net/diccionario/que-es-un-programa-informatico/>

<https://desarrollarinclusion.cilsa.org/tecnologia-inclusiva/que-es-un-sistema-operativo/>

<https://microsofters.com/178183/windows-11-novedades-requisitos-como-instalarlo/>

<https://itsoftware.com.co/content/sistemas-operativos-mas-usados/>

<https://tecnotrono.com/software/sistemas-operativos/257/#Caracteristicas de Windows>

<https://latam.kaspersky.com/resource-center/definitions/shareware>

<https://siamchamnankit.co.th/history-some-pictures-and-pdfs-of-the-agile-manifesto-meeting-on-2001-a33c40bcc2b>

<https://gravitar.biz/bi/metodologias-agiles-intro/>

<https://carloszr.com/wp-content/uploads/comentario-javadoc.png>

<https://i1.wp.com/blog.solucioneslmv.com/wp-content/uploads/2018/01/entrevista-de-trabajo-1498644135726.jpg?fit=712%2C350>

[https://sites.google.com/site/todouml/\\_/rsrc/1358882769745/ejercicios/ejercicios-soluciones/20-casos-de-uso-de-un-cajero/atm1.JPG](https://sites.google.com/site/todouml/_/rsrc/1358882769745/ejercicios/ejercicios-soluciones/20-casos-de-uso-de-un-cajero/atm1.JPG)

<https://cdn5.icemd.com/app/uploads/2017/09/prototipo-app-790x526.jpg>

<https://i.stack.imgur.com/uHQ6c.png>

[https://static.wixstatic.com/media/a59c1e\\_10d73daf1dc44c549035d96ba5c0e605~mv2.png/v1/fill/w\\_1000,h\\_630,al\\_c,usm\\_0.66\\_1.00\\_0.01/a59c1e\\_10d73daf1dc44c549035d96ba5c0e605~mv2.png](https://static.wixstatic.com/media/a59c1e_10d73daf1dc44c549035d96ba5c0e605~mv2.png/v1/fill/w_1000,h_630,al_c,usm_0.66_1.00_0.01/a59c1e_10d73daf1dc44c549035d96ba5c0e605~mv2.png)

<https://ocw.unican.es/pluginfile.php/1408/course/section/1803/tema8-mantenimientoSistemasSoftware.pdf>

<https://m.facebook.com/KaizenCrowd/photos/a.772461076827820/920470682026858/?type=3&source=54>

<https://static.elcorreo.com/www/multimedia/202007/01/media/cortadas/compra-online2-kXjC-U110671851650wBB-1248x770@RC.jpg>