

# (index.html)

Tema1

Escuchar ([https://app-eu.readspeaker.com/cgi-bin/rsent?customerid=13417&lang=es\\_es&readid=main&url=https://aulavirtual-fpdrioja.larioja.org/pluginfile.php](https://app-eu.readspeaker.com/cgi-bin/rsent?customerid=13417&lang=es_es&readid=main&url=https://aulavirtual-fpdrioja.larioja.org/pluginfile.php))

## Desarrollo de software.



### Caso práctico

En BK Programación todos han vuelto ya de sus vacaciones.

Les espera un septiembre agitado, pues acaban de recibir una petición por parte de una cadena hotelera para desarrollar un proyecto software.

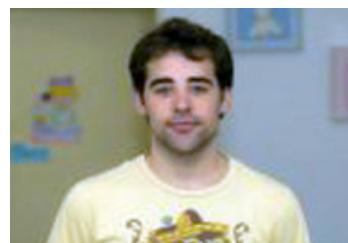
Ada, la supervisora de proyectos de BK Programación, se reúne con Juan y María (trabajadores de la empresa) para empezar a planificar el proyecto.

Ana, cuya especialidad es el diseño gráfico de páginas web, acaba de terminar el Ciclo de Grado Medio en Sistemas Microinformáticos y Redes y realizó la FCT en BK Programación. Trabaja en la empresa ayudando en los diseños, y aunque está contenta con su trabajo, le gustaría participar activamente en todas las fases en el proyecto. El problema es que carece de los conocimientos necesarios.

Antonio se ha enterado de la posibilidad de estudiar el nuevo Ciclo de Grado Superior de Diseño de Aplicaciones Multiplataforma a distancia, y está dispuesta a hacerlo. (No tendría que dejar el trabajo).

Le comenta sus planes a su amigo Antonio (que tiene conocimientos básicos de informática), y éste se une a ella.

Después de todo... ¿qué pueden perder?



**Materiales formativos de FP Online propiedad del Ministerio de Educación, Cultura y Deporte.**

Aviso Legal

# 1.- Software y programa. Tipos de software.



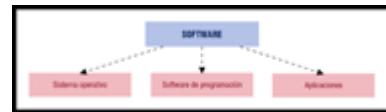
## Caso práctico



Todos en la empresa están entusiasmados con el proyecto que tienen entre manos. Saben que lo más importante es planificarlo todo de antemano y elegir el tipo de software más adecuado. Ana les escucha hablar y no llega a entender por qué hablan de "tipos de software". ¿Acaso el software no era la parte lógica del ordenador, sin más? **¿Cuáles son los tipos de software?**

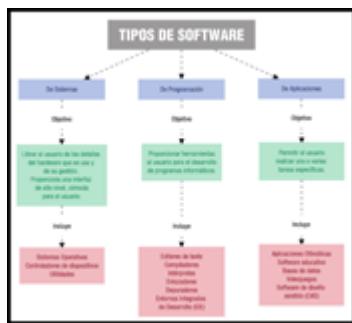
**Ada (Apache 2.2)**

Es de sobra conocido que el ordenador se compone de dos partes bien diferenciadas: hardware y software.



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

Según su función se distinguen **tres tipos de software**: sistema operativo, software de programación y aplicaciones.



El **sistema operativo** es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. Son ejemplos de sistemas operativos: Windows, Linux, Mac OS X ...

El **software de programación** es el conjunto de herramientas que nos permiten desarrollar programas informáticos, y las **aplicaciones informáticas** son un conjunto de programas que tienen una finalidad más o menos concreta. Son ejemplos de aplicaciones: un procesador de textos, una hoja de cálculo, el software para reproducir música, un videojuego, etc. A su vez, un programa es un conjunto de instrucciones escritas en un lenguaje de programación.

En este tema, nuestro interés se centra en las aplicaciones informáticas: cómo se desarrollan y cuáles son las fases por las que necesariamente han de pasar. A lo largo de esta primera unidad vas a aprender los conceptos fundamentales de software y las fases del llamado ciclo de vida de una aplicación informática. También aprenderás a distinguir los diferentes lenguajes de programación y los procesos que ocurren hasta que el programa funciona y realiza la acción deseada.



## Para saber más

En el siguiente enlace encontrarás más información de los tipos de software existente, así como ejemplos de cada uno que te ayudarán a profundizar sobre el tema.

[Ampliación sobre los tipos de software.](#)

---



## Reflexiona

Hay varios sistemas operativos en el mercado: Linux, Windows, Mac OS X etc. El más conocido es Windows. A pesar de eso, ¿por qué utilizamos cada vez más Linux?

---

## 2.- Relación hardware-software.



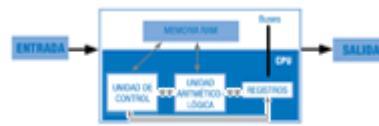
### Caso práctico

Después de saber ya diferenciar los distintos tipos de software, Ana se le plantea otra cuestión: El software, sea del tipo que sea, se ejecuta sobre los dispositivos físicos del ordenador. **¿Qué relación hay entre ellos?**



Como sabemos, al conjunto de dispositivos físicos que conforman un ordenador se le denomina hardware. Existe una relación indisoluble entre éste y el software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione. El software se ejecutará sobre los dispositivos físicos.

La primera arquitectura hardware con programa almacenado se estableció en 1946 por John Von Neumann:



Descripción de la imagen

Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:

#### a. Desde el 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).

#### b. Desde el punto de vista de las aplicaciones

Ya hemos dicho que una aplicación no es otra cosa que un conjunto de programas, y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Hay multitud de lenguajes de programación diferentes (como ya veremos en su momento). 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?.

Como veremos a lo largo de esta unidad, tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.



## Autoevaluación

**Para fabricar un programa informático que se ejecuta en una computadora:**

- Hay que escribir las instrucciones en código binario para que las entienda el hardware.
- Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
- Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
- Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

Incorrecta, ya que el ser humano no tiene capacidad para escribir programas usando 1 y 0.

No es correcta porque el hardware no entiende ese lenguaje.

Muy bien. Esa es la idea...

No es cierta ninguna de las dos afirmaciones.

## Solución

- 1.** Incorrecto (#answer-7\_63)
- 2.** Incorrecto (#answer-7\_80)
- 3.** Opción correcta (#answer-7\_83)
- 4.** Incorrecto (#answer-7\_86)

### 3.- Fases en el desarrollo y ejecución del software.



#### Caso práctico

En la reunión de BK acerca del nuevo proyecto Ada, la supervisora, dejó bien claro que lo primero y más importante es tener claro qué queremos que haga el software y con qué herramientas contamos: lo demás vendría después, ya que si esto no está bien planteado, ese error se propagará a todas las fases del proyecto.

—¿Por dónde empezamos? —pregunta Juan.

—**ANÁLISIS de REQUISITOS** —contesta Ada.



---

A modo resumen, las etapas que podemos encontrar dentro del desarrollo software son:

#### **1. ANÁLISIS DE REQUISITOS.**

Se especifican los requisitos funcionales y no funcionales del sistema.

#### **2. DISEÑO.**

Se divide el sistema en partes y se determina la función de cada una.

#### **3. CODIFICACIÓN.**

Se elige un Lenguajes de Programación y se codifican los programas.

#### **4. PRUEBAS.**

Se prueban los programas para detectar errores y se depuran.

#### **5. DOCUMENTACIÓN.**

De todas las etapas, se documenta y guarda toda la información.

#### **6. EXPLORACIÓN.**

Instalamos, configuramos y probamos la aplicación en los equipos del cliente.

#### **7. MANTENIMIENTO.**

Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.



#### Autoevaluación

**¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?**

- Sí.
- No.

Incorrecto. Recuerda que hay que dejar siempre una "puerta abierta" para volver atrás e introducir modificaciones.

Muy bien, vas captando la idea.

## Solución

- 1. Incorrecto (#answer-34\_63)**
  - 2. Opción correcta (#answer-34\_109)**
-

### 3.1.- Análisis.

Esta es la primera fase del proyecto. Una vez finalizada, pasamos a la siguiente (diseño).

Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.

**Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista.**

¿Qué se hace en esta fase?

Se especifican y analizan los requisitos funcionales y no funcionales del sistema.

**Requisitos:**

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

Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.

La culminación de esta fase es el documento ERS (Especificación de Requisitos Software). Donde el documento más importante es la relación de los requisitos funcionales y no funcionales del sistema, junto con la priorización de los mismos; indicando cuales son obligatorios, cuales opcionales, etc.



#### Citas para pensar

Todo aquello que no se detecte, o resulte mal entendido en la etapa inicial provocará un fuerte impacto negativo en los requisitos, propagando esta corriente degradante a lo largo de todo el proceso de desarrollo e **incrementando su perjuicio cuanto más tardía sea su detección**

*(Bell y Thayer 1976)(Davis 1993).*

### 3.1.1. Captura de requisitos

En este punto se muestra un ejercicio resuelto en el que a partir de información extraída del usuario hemos obtenido los requisitos funcionales y no funcionales. Para ello tenemos que recordar que:

- Los requisitos funcionales definen los servicios que el sistema debe proporcionar.
- Los requisitos no funcionales son restricciones que afectan a los requisitos funcionales del sistema, como restricciones de tiempo, estándares que tiene que seguir, legislación a cumplir, etc.

Además, es importante que entiendas que los requisitos funcionales deben estar redactados de forma que sean comprensibles para usuarios sin conocimientos técnicos de informática; y teniendo en cuenta que no deben de existir ambigüedades.



#### Ejercicio Resuelto

##### Enunciado

A partir del siguiente enunciado:

1. Crea una tabla con requisitos funcionales (Id. requisito | Descripción del requisito )
2. Crea una tabla con requisitos NO funcionales (Id. requisito | Descripción del requisito )

Una empresa, "Vuela S.L", es una empresa que se encarga de realizar reservas de vuelos para diferentes compañías aéreas. Dispone de 50 oficinas por diferentes provincias de España, con aproximadamente 2 millones de reservas de vuelos anuales. Actualmente, existe un sistema centralizado en Madrid. Sin embargo, todas las transacciones que se cierran en las diferentes agencias se realizan a mano siguiendo un formulario concreto y se comunican a la central vía telefónica. Los agentes que trabajan en las diferentes agencias, usan también horarios y folletos impresos con los diferentes itinerarios que ofrecen para contestar las cuestiones que les planteen los clientes. También existe la posibilidad de realizar las reservas por el cliente de manera telefónica, la cual se realiza directamente en el sistema central a través de la centralita allí existente.

En la nueva aplicación, cuando un cliente quiera reservar un vuelo, se les debe ofrecer información sobre las diferentes opciones disponibles para el trayecto requerido desde la propia aplicación. Además se podrá imprimir la relación de vuelos posibles para que el cliente tenga la información disponible. En caso de que el cliente se decida a reservar un vuelo, tiene la opción de hacer una reserva provisional, la cual es válida por 3 días, y el cliente recibirá un recibo con los detalles de la pre-reserva. Si el cliente decide confirmar la reserva, éste tiene que pagar un 20% del precio total a modo de depósito. Una semana antes de la fecha del viaje el cliente debe

abonar el precio total del billete. En tal caso, ocurren dos cosas: se remite de forma automática el billete al correo electrónico proporcionado por la persona y en caso de que el cliente lo solicitara en la reserva, se imprimen los billetes y se le envían por mensajería. Tanto las reservas provisionales como las confirmadas, tienen que poder realizarlas los agentes a través del nuevo sistema. En el nuevo sistema, la aplicación debe seguir permitiendo la reserva telefónica.

La mayoría de gerentes de la compañía expresan su preocupación pensando que el nuevo sistema puede ocasionar muchos problemas a menos que se utilice en hardware ya existente. Además, se cree que tiene que existir un sistema de respaldo, ya que la reserva telefónica ofrece, y debe seguir ofreciendo, servicio 24h al día, 365 días al año.

Indican que el sistema tiene que ser simple de usar para los usuarios de las diferentes agencias de viaje y de la central, proporcionando una guía de ayuda al usuario a lo largo de todo el proceso de reserva.

Mostrar retroalimentación





## 3.2.- Diseño.



### Caso práctico

Juan está agobiado por el proyecto. Ya han mantenido comunicaciones con el cliente y saben perfectamente qué debe hacer la aplicación. También tiene una lista de las características hardware de los equipos de su cliente y todos los requisitos. Tiene tanta información que no sabe por dónde empezar.

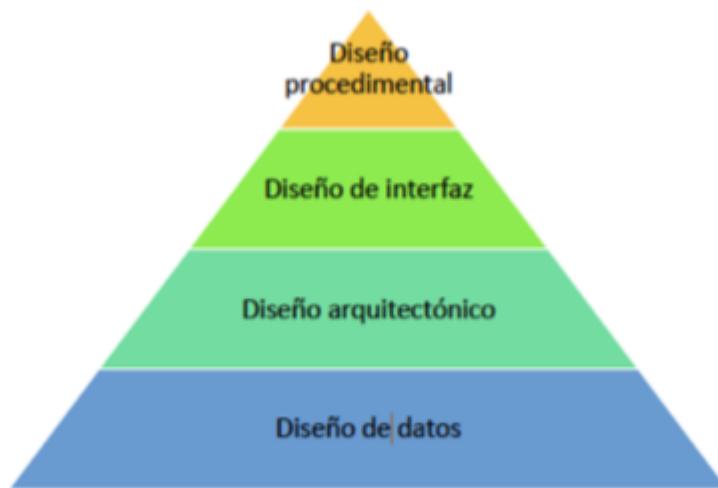
Decide hablar con Ada. Su supervisora, amable como siempre, le sugiere que empiece a dividir el problema en las partes implicadas.

—Vale, Ada, pero, **¿cómo lo divido?**



---

Una vez identificados los requisitos es necesario componer la forma en que se solucionará el problema. En esta fase traduciremos los requisitos funcionales y no funcionales en una representación del software a desarrollar. Podemos encontrar 4 tipos de diseño dentro de esta fase:



### Diseño de datos

Se encarga de transformar la información relativa al dominio del problema (obtenida durante el análisis) en estructuras de datos que se utilizan en la implementación. Es una tarea de abstracción, donde se definirán aquellas estructuras necesarias para abordar el desarrollo del programa.

Para realizar este diseño se pueden utilizar diagramas como entidad/relación (relacionadas con bases de datos) o diagramas de clases UML (de propósito más genérico), entre otros.



Diagrama entidad relación



Diagrama de clases UML

## Diseño de la interfaz

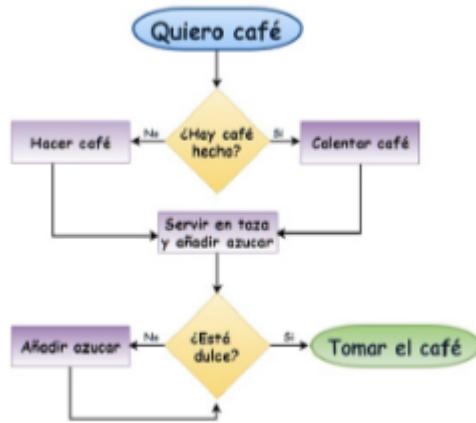
Describe cómo se comunica el software consigo mismo, con los sistemas que operan con él, y con las personas que lo utilizan. Para representar como pueden comunicarse las personas con el software se pueden utilizar prototipos. La siguiente imagen muestra un ejemplo de un prototipo realizado en papel, no es la única forma de realizarlos.



## Diseño procedimental

Transforma los elementos estructurales anteriormente diseñados en una descripción procedimental. Es decir, describe a gran nivel el detalle de los algoritmos que implementarán los anteriores diseños. El resultado de este diseño es un nivel de diseño suficiente para que sirva de guía a la hora de generar el código fuente. Existen numerosas técnicas para realizar el diseño procedimental.

**Diagramas de flujo:** se utilizan una serie de símbolos interconectados que tienen un significado concreto.



**Pseudocódigo:** utiliza texto descriptivo para realizar el diseño del algoritmo.

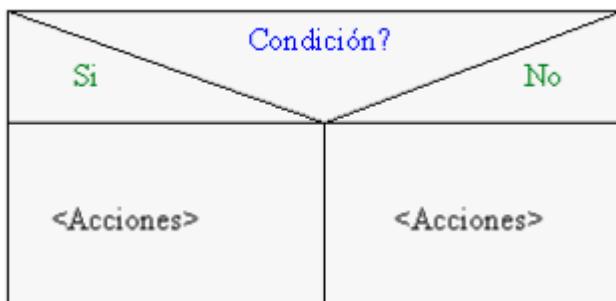
---

```

algoritmo Sumar
variables
  entero a, b, c

inicio
  escribir( "Introduzca el primer número (entero): " )
  leer( a )
  escribir( "Introduzca el segundo número (entero): " )
  leer( b )
  c ← a + b
  escribir( "La suma es: ", c )
fin
  
```

**Otros:** diagrama de cajas, tablas de decisión, ...



Condiciones	Reglas							
Condición 1	S	S	S	S	N	N	N	N
Condición 2	S	S	N	N	S	S	N	N
Condición 3	S	N	S	N	S	N	S	N
Acción 1	X	X						
Acción 2			X		X			X
Acción 3			X				X	
Acción 4				X				

### 3.3.- Codificación



#### Caso práctico

En BK, ya tienen el proyecto dividido en partes.

Ahora llega una parte clave: codificar los pasos y acciones a seguir para que el ordenador los execute. En otras palabras, **programar la aplicación**. Saben que no será fácil, pero afortunadamente cuentan con herramientas CASE que les van a ser de gran ayuda. A Ana el gustaría participar, pero cuando se habla de "código fuente", "ejecutable", etc. sabe que no tiene ni idea y que no tendrá más remedio que estudiarlo si quiere colaborar en esta fase del proyecto.



---

Durante la fase de codificación se realiza el proceso de programación. Consiste en elegir un determinado lenguaje de programación y generar el código fuente que codifique toda la información anterior.

**Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.**

Las características deseables de todo código son:

1. Modularidad: que esté dividido en trozos más pequeños.
2. Corrección: que haga lo que se le pide realmente.
3. Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
4. Eficiencia: que haga un buen uso de los recursos.
5. Portabilidad: que se pueda implementar en cualquier equipo.

---

Durante esta fase, el código pasa por diferentes estados:

**Código Fuente:** es el escrito por los programadores en algún editor de texto. Se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesarias.

```
read b
if [ $a -eq $b ]
then
{
    echo $a es igual a $b
}
else
{
    if [ $a -lt $b ]
    then
    {
        echo $a es menor que $b
    }
    else
    {
        echo $a es mayor que $b
    }
}
```

**Código Objeto:** es el código binario resultado de compilar el código fuente. La **compilación** es la traducción en una sola vez del código fuente, y se realiza utilizando un compilador. La **interpretación** es la traducción y ejecución simultánea del código fuente línea a línea.

El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora. Es un código intermedio entre el código fuente y el ejecutable (o máquina) y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso. Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.

**Código Ejecutable (o máquina):** Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. Ya sí es directamente inteligible por la computadora.

---

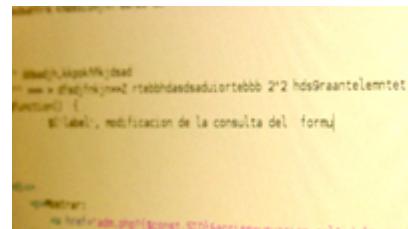
### 3.3.1.- Lenguajes de programación.



#### Caso práctico

Una de los aspectos del proyecto que más preocupa a Ada es la elección del lenguaje de programación a utilizar.

Necesita tener muy claros los requerimientos del cliente para enfocar correctamente la elección, pues según sean éstos unos lenguajes serán más efectivos que otros.



---

Los lenguajes de programación han evolucionado, y siguen haciéndolo, siempre hacia la mayor usabilidad de los mismos (que el mayor número posible de usuarios lo utilicen y exploten). La elección del lenguaje de programación para codificar un programa dependerá de las características del problema a resolver.

Podemos definir un lenguaje de programación como un conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas. Es un lenguaje artificial, una construcción mental del ser humano para expresar programas.

Además, cada lenguaje de programación, al igual que otro tipo de lenguajes, se basa en una gramática.

**Gramática del lenguaje:** Reglas aplicables al conjunto de símbolos y palabras especiales del lenguaje de programación para la construcción de sentencias correctas.

Además, cada gramática dispone de:

- 1. Léxico:** Es el conjunto finito de símbolos y palabras especiales, es el vocabulario del lenguaje.
- 2. Sintaxis:** Son las posibles combinaciones de los símbolos y palabras especiales. Está relacionada con la forma de los programas.
- 3. Semántica:** Es el significado de cada construcción del lenguaje, la acción que se llevará a cabo.



#### Para saber más

En el siguiente enlace, verás la evolución entre los distintos tipos de Lenguajes de Programación en la historia.

## Evolución de los Lenguajes de Programación.

---

### 3.3.1.1. Clasificación según la forma de ejecución

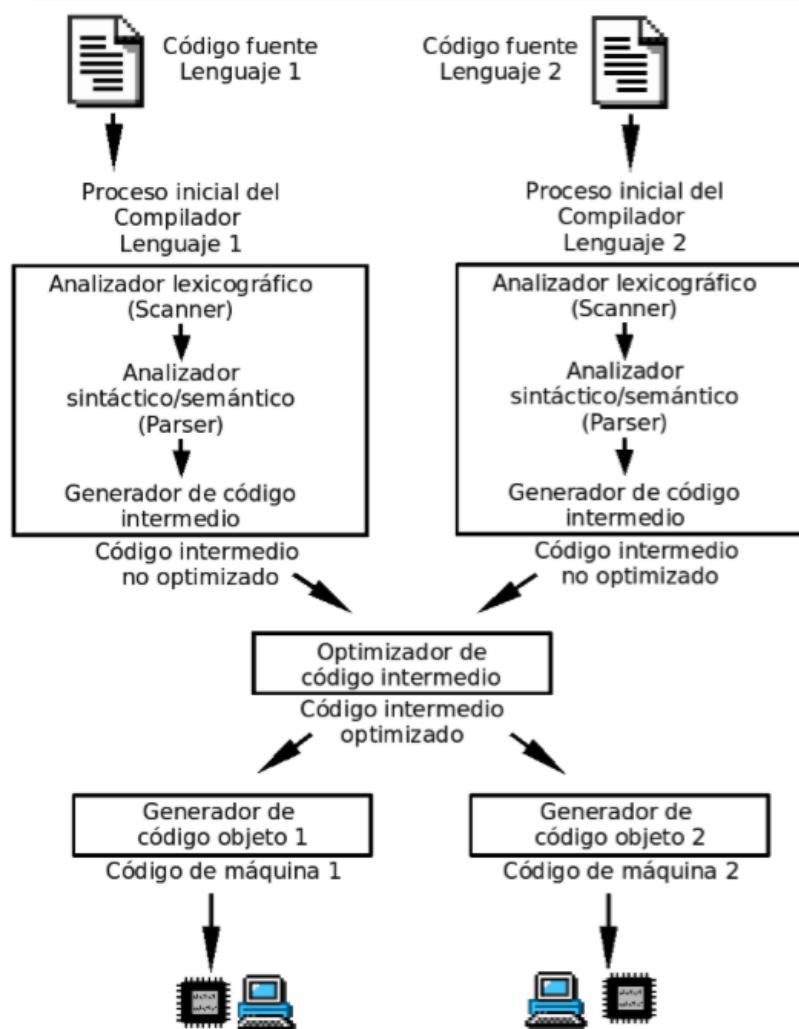
Los lenguajes de programación se pueden clasificar atendiendo a la forma de ejecutarse. Diferenciando entre lenguajes compilados y lenguajes interpretados.

#### Lenguajes compilados

En estos tipos de lenguajes existe un traductor de código fuente a código máquina denominado compilador. El compilador realizará la traducción y además informará de los posibles errores. Una vez subsanados, se generará el programa traducido a código máquina, conocido como código objeto. Este programa aún no podrá ser ejecutado hasta que no se le añadan los módulos de enlace o bibliotecas, durante el proceso de enlazado. Una vez finalizado el enlazado, se obtiene el código ejecutable.

Algunos ejemplos de este tipo de lenguajes son: Pascal, Fortran, Algol, C, C++, etc.

En la siguiente imagen se puede ver cómo sería el proceso de compilado.



#### Lenguajes interpretados

Se caracterizan por estar diseñados para que su ejecución se realice a través de un intérprete. Cada instrucción escrita en un lenguaje interpretado se analiza,

traduce y ejecuta tras haber sido verificada. Una vez realizado el proceso por el intérprete, la instrucción se ejecuta, pero no se guarda en memoria.

**Intérprete:** Es un programa traductor de un lenguaje de alto nivel en el que el proceso de traducción y de ejecución se llevan a cabo simultáneamente, es decir, la instrucción se pasa a lenguaje máquina y se ejecuta directamente. No se genera programa objeto, ni programa ejecutable.

Ejemplos de lenguajes interpretados son: Perl, PHP, Python, JavaScript, etc.

## Lenguajes virtuales

A medio camino entre los lenguajes compilados y los interpretados, existen los lenguajes que podemos denominar virtuales. Es el caso del Lenguaje Java. Java puede verse como compilado e interpretado a la vez, ya que su código fuente se compila para obtener el código binario en forma de **bytecode**, que son estructuras parecidas a las instrucciones máquina, con la importante propiedad de no ser dependientes de ningún tipo de máquina (se detallarán más adelante). Los ficheros que contienen los bytecodes de Java tienen extensión .class. La Máquina Virtual Java se encargará de interpretar este código y, para su ejecución, lo traducirá a código máquina del procesador en particular sobre el que se esté trabajando.



### Para saber más

En la página web siguiente encontrarás un resumen de las características de los Lenguajes de Programación más utilizados en la actualidad.

[Características de los Principales Lenguajes de Programación.](#)

---

### 3.3.1.2. Clasificación según el paradigma de programación

#### Programación estructurada

Aunque los requerimientos actuales de software son bastante más complejos de lo que la técnica de programación estructurada es capaz, es necesario por lo menos conocer las bases de los Lenguajes de Programación estructurados, ya que a partir de ellos se evolucionó hasta otros lenguajes y técnicas más completas (orientada a eventos u objetos) que son las que se usan actualmente.

La programación estructurada se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

- Sentencias secuenciales.
- Sentencias selectivas (condicionales).
- Sentencias repetitivas (iteraciones o bucles).



Los lenguajes de programación que se basan en la programación estructurada reciben el nombre de lenguajes de programación estructurados.

La programación estructurada fue de gran éxito por su sencillez a la hora de construir y leer programas. Fue sustituida por la programación modular, que permitía dividir los programas grandes en trozos más pequeños (siguiendo la conocida técnica "divide y vencerás"). Ejemplos de lenguajes estructurados: Pascal, C, Fortran.

#### VENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA

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

#### INCONVENIENTES

- Todo el programa se concentra en un único bloque (si se hace demasiado grande es difícil manejarlo).
- No permite reutilización eficaz de código, ya que todo va "en uno". Es por esto que a la programación estructurada le sustituyó la programación modular, donde los programas se codifican por módulos y bloques, permitiendo mayor reutilización.

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 reutilizables.

---

## Programación orientada a objetos

Después de comprender que la programación estructurada no es útil cuando los programas se hacen muy largos, es necesaria otra técnica de programación que solucione este inconveniente. Nace así la Programación Orientada a Objetos (en adelante, P.O.O.). Los lenguajes de programación orientados a objetos tratan a los programas no como un conjunto ordenado de instrucciones (tal como sucedía en la programación estructurada) sino como un conjunto de objetos que colaboran entre ellos para realizar acciones. Programación orientada a objetos

En la P.O.O. los programas se componen de objetos independientes entre sí que colaboran para realizar acciones. Los objetos son reutilizables para proyectos futuros.

---

Su primera desventaja es clara: no es una programación tan intuitiva como la estructurada. A pesar de eso, alrededor del 55% del software que producen las empresas se hace usando esta técnica. Los principales lenguajes orientados a objetos: Ada, C++, C#, Delphi, Java, etc.

Ventajas:

- El código es reutilizable.
- Si hay algún error, es más fácil de localizar y depurar en un objeto que en un programa entero.

Características:

- Los objetos del programa tendrán una serie de atributos que los diferencian unos de otros.
- Se define clase como una colección de objetos con características similares.
- Mediante los llamados métodos, los objetos se comunican con otros produciéndose un cambio de estado de los mismos.
- Los objetos son, pues, como unidades individuales e indivisibles que forman la base de este tipo de programación.

## Para saber más



En el siguiente enlace hay una guia muy interesante de introducción a la programación orientada a objetos, en concreto, del lenguaje Java.

[https://www.w3schools.com/java/java\\_oop.asp](https://www.w3schools.com/java/java_oop.asp)

---

### 3.3.1.3. Clasificación de los lenguajes según su nivel de abstracción

Los lenguajes de programación han evolucionado a lo largo del tiempo para aumentar su rendimiento y facilitar el trabajo a los programadores. Cada vez, existen lenguajes de programación que son más "legibles" y por lo tanto más fáciles de usar y entender. A continuación se muestran tres niveles de abstracción sobre los que se pueden clasificar los lenguajes de programación.

#### Bajo nivel de abstracción

- También se suele llamar lenguaje máquina.
- Sus instrucciones son complejas e ininteligibles. Se componen de combinaciones de unos y ceros.
- No necesita ser traducido, por lo tanto es el único lenguaje que entiende directamente el ordenador.
- Fue el primer lenguaje utilizado. En su momento, los expertos debían tener un dominio profundo del hardware para poder entender este lenguaje de programación.
- Difiere para cada procesador. Las instrucciones no son portables de un equipo a otro.
- Salvo excepciones, no se suele utilizar este lenguaje para programar hoy en día.

```
0110101101010110100010111010101111010100110101010  
010101010101010110101101010110100010111010101011110  
10101010001011010101010101101011010101101000101  
0101111010100110101010001011010101010101011010110  
0100010111010101011110101001101010100010110101010  
01101011010110100010111010101111010100110101010  
01010101010101101010101110101011010101101010111  
110101010001011010101010101101010110100010  
101011110101001101010100010110101010000101010101101  
101101000101110101011111010100110101010001011010  
1010101101011010101110100010001011101010111101010  
101000101101010101010101101010110101011010001011101  
1110101001101010100010110101011101010110101101010  
010111010101111010100110101010001011010101010101010
```

#### Lenguaje de nivel medio

- Dada la dificultad y poca portabilidad del lenguaje máquina, el *ensamblador* lo sustituyó para facilitar la labor de programación.
- Sigue estando cercano al hardware, pero, en lugar de unos y ceros, se programó usando mnemotécnicos, que son instrucciones inteligibles por el programador que permiten comprender de una forma más sencilla qué hace el programa.

- Este lenguaje necesita compilarse y traducirse al lenguaje máquina para poder ejecutarse.
- Se trabaja con los registros del procesador y direcciones físicas. En lenguajes de nivel más alto, ya se utilizan variables y estructuras más sofisticadas.
- Es difícil de comprender y programar.
- Dada la dificultad y poca portabilidad del lenguaje máquina, el ensamblador lo sustituyó para facilitar la labor de programación.

```

MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER PAGE    2

C000      ORG     ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START   LDS     #STACK

*****  

* FUNCTION: INITA - Initialize ACIA  

* INPUT: none  

* OUTPUT: none  

* CALLS: none  

* DESTROYS: acc A

0013      RESETA EQU     %00010011
0011      CTLREG EQU     %00010001

C003 86 13  INITA   LDA A  #RESETA  RESET ACIA
C005 B7 80 04  STA A  ACIA
C008 86 11  LDA A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04  STA A  ACIA

C00D 7E C0 F1  JMP     SIGNON  GO TO START OF MONITOR

*****  

* FUNCTION: INCH - Input character  

* INPUT: none  

* OUTPUT: char in acc A  

* DESTROYS: acc A  

* CALLS: none  

* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH    LDA A  ACIA    GET STATUS
C013 47        ASR A   RDRF    SHIFT RDRF FLAG INTO CARRY
C014 24 FA    BCC    INCH    RECEIVE NOT READY
C016 B6 80 05  LDA A  ACIA+1  GET CHAR
C019 84 7F    AND A  #$7F    MASK PARITY
C01B 7E C0 79  JMP     OUTCH   ECHO & RTS

*****  

* FUNCTION: INHEX - INPUT HEX DIGIT  

* INPUT: none  

* OUTPUT: Digit in acc A  

* CALLS: INCH  

* DESTROYS: acc A  

* Returns to monitor if not HEX input

C01E 8D F0  INHEX   BSR     INCH    GET A CHAR
C020 81 30  CMP A  #'0    ZERO
C022 2B 11  BMI     HEXERR  NOT HEX
C024 81 39  CMP A  #'9    NINE
C026 2F 0A  BLE     HEXRTS  GOOD HEX
C028 81 41  CMP A  #'A    NOT HEX
C02A 2B 09  BMI     HEXERR
C02C 81 46  CMP A  #'F    FIX A-F
C02E 2E 05  BGT     HEXERR
C030 80 07  SUB A  #7    CONVERT ASCII TO DIGIT
C032 84 0F  HEXRTS  AND A  #$0F
C034 39        RTS

C035 7E C0 AF  HEXERR  JMP     CTRL    RETURN TO CONTROL LOOP

```

## Lenguaje de alto nivel

- La mayoría de los lenguajes de programación actuales pertenecen a esta categoría (Java, Python, C#, C, C++, ...).
- Tienen una forma de programar más intuitiva y sencilla.
- Más cercano al lenguaje humano que al lenguaje máquina.
- Suelen tener librerías y funciones predeterminadas que solucionan algunos de los problemas que suelen presentarse al programador.
- En ocasiones, ofrecen frameworks para una programación más eficiente y rápida.

- Suelen trabajar con mucha abstracción y orientación a objetos. De esta manera, es más fácil la reutilización y la encapsulación de componentes.

```
import unidecode
import re

def eliminar_tildes_minuscula(texto):
    unidecode.unidecode(texto)
    return unidecode.unidecode(texto).lower()

def eliminar_hashtags_puntuacion(texto):
    texto = texto.replace("#", " ")
    return texto.replace(" ", "")

texto = 'El #terremoto que sacudió el sábado al suroeste de #Haití dejó [...]'
texto = eliminar_tildes_minuscula(texto)
texto = eliminar_hashtags_puntuacion(texto)

print(texto)
```

### 3.3.2- Fases en la obtención de código.



#### Caso práctico

Juan y María ya han decidido el Lenguajes de Programación que van a utilizar.

Saben que el programa que realicen pasará por varias fases antes de ser implementado en los equipos del cliente. Todas esas fases van a producir transformaciones en el código. ¿Qué características irá adoptando el código a medida que avanza por el proceso de codificación?

---

### 3.3.2.1.- Fuente.

El código fuente es el conjunto de instrucciones que la computadora deberá realizar, escritas por los programadores en algún lenguaje de alto nivel. Este conjunto de instrucciones no es directamente ejecutable por la máquina, sino que deberá ser traducido al lenguaje máquina (o ejecutable), que la computadora será capaz de entender y ejecutar.

Un aspecto muy importante en esta fase es la elaboración previa de un algoritmo . El algoritmo lo diseñamos en pseudocódigo  y con él, la codificación posterior a algún Lenguaje de Programación determinado será más rápida y directa.

Para obtener el código fuente de una aplicación informática:

1. Se debe partir de las etapas anteriores de análisis y diseño.
2. Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.
3. Se elegirá una Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar.
4. Se procederá a la codificación del algoritmo antes diseñado.

La culminación de la obtención de código fuente es un documento con la codificación de todos los módulos, funciones, bibliotecas y procedimientos necesarios para codificar la aplicación.

Puesto que, como hemos dicho antes, este código no es inteligible por la máquina, habrá que **TRADUCIRLO**, obteniendo así un código equivalente pero ya traducido a código binario que se llama código objeto (que explicaremos posteriormente).

Un aspecto importante a tener en cuenta es su licencia. Así, en base a ella, podemos distinguir dos tipos de código fuente:

- Código fuente abierto. Es aquel que está disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
- Código fuente cerrado. Es aquel que no tenemos permiso para editarlo.



#### Para saber más...

Puedes investigar en más profundidad sobre el software libre a través de este enlace <https://www.gnu.org/philosophy/free-sw.es.html#four-freedoms>.



#### Autoevaluación

**Para obtener código fuente a partir de toda la información necesaria del problema:**

- Se elige el Lenguaje de Programación más adecuado y se codifica directamente.
- Se codifica y después se elige el Lenguaje de Programación más adecuado.
- Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.

Incorrecta. La codificación directa nos llevará mucho tiempo y tendremos demasiados errores.

No es correcta. Antes de programar tenemos que saber qué Lenguajes de Programación vamos a utilizar.

Muy bien. El diseño del algoritmo (los pasos a seguir) nos ayudará a que la codificación posterior se realice más rápidamente y tenga menos errores.

## Solución

- 1. Incorrecto (#answer-49\_63)**
  - 2. Incorrecto (#answer-49\_120)**
  - 3. Opción correcta (#answer-49\_123)**
-

### 3.3.2.2.- Objeto.

El código objeto es un código intermedio. Es el resultado de traducir código fuente a un código equivalente formado por unos y ceros que aún no puede ser ejecutado directamente por la computadora. Es decir, es el código resultante de la compilación del código fuente.

Consiste en un bytecode (código binario) que está distribuido en varios archivos, cada uno de los cuales corresponde a cada programa fuente compilado. Sólo se genera código objeto una vez que el código fuente está libre de errores sintácticos y semánticos.

El proceso de traducción de código fuente a código objeto puede realizarse de dos formas:

- a. Compilación:** El proceso de traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá que enlazar. El software responsable se llama compilador.
- b. Interpretación:** El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente. No existe código objeto intermedio. El software responsable se llama intérprete. El proceso de traducción es más lento que en el caso de la compilación, pero es recomendable cuando el programador es inexperto, ya que da la detección de errores es más detallada.

**El código objeto es código binario, pero no puede ser ejecutado por la computadora**

---

El código objeto es código binario, pero no puede ser ejecutado por la computadora

---



#### Para saber más

En el siguiente enlace podrás visitar una página web, que te permitirá aprender más acerca de la generación de códigos objeto:

## Generación de código objeto.

---

---

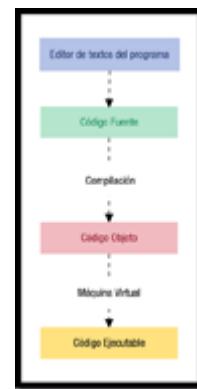
### 3.3.2.3.- Ejecutable.

El código ejecutable, resultado de enlazar los archivos de código objeto, consta de un único archivo que puede ser directamente ejecutado por la computadora. No necesita ninguna aplicación externa. Este archivo es ejecutado y controlado por el sistema operativo.

Para obtener un sólo archivo ejecutable, habrá que enlazar todos los archivos de código objeto, a través de un software llamado linker (enlazador) y obtener así un único archivo que ya sí es ejecutable directamente por la computadora.

En el esquema de generación de código ejecutable, vemos el proceso completo para la generación de ejecutables.

1. A partir de un editor, escribimos el lenguaje fuente con algún Lenguaje de programación. (En el ejemplo, se usa Java).
2. A continuación, el código fuente se compila obteniendo código objeto o bytecode.
3. Ese bytecode, a través de la máquina virtual (se verá en el siguiente punto), pasa a código máquina, ya directamente ejecutable por la computadora.



Generación de código ejecutable.



### Autoevaluación

Relaciona los tipos de código con su característica más relevante, escribiendo el número asociado a la característica en el hueco correspondiente.

#### Ejercicio de relacionar

Tipo de código.	Relación.	Características.
Código Fuente	<input type="text"/>	1. Escrito en Lenguaje Máquina pero no ejecutable.
Código Objeto	<input type="text"/>	2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable.
Código Ejecutable	<input type="text"/>	3. Escrito en Lenguaje Máquina y directamente ejecutable.

Tipo de código.	Relación.	Características.

Enviar

El código fuente escrito en algún lenguaje de programación de alto nivel, el objeto escrito en lenguaje máquina sin ser ejecutable y el código ejecutable, escrito también en lenguaje máquina y ya sí ejecutable por el ordenador, son las distintas fases por donde pasan nuestros programas.

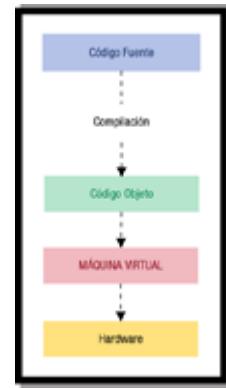
---

### 3.3.3- Máquinas virtuales.

Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

Esta capa de software desempeña un papel muy importante en el funcionamiento de los lenguajes de programación, tanto compilados como interpretados.

Con el uso de máquinas virtuales podremos desarrollar y ejecutar una aplicación sobre cualquier equipo, independientemente de las características concretas de los componentes físicos instalados. Esto garantiza la **portabilidad** de las aplicaciones.



Las funciones principales de una máquina virtual son las siguientes:

- Conseguir que las aplicaciones sean portables.
- Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.
- Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos.
- Cumplimiento de las normas de seguridad de las aplicaciones.

#### CARACTERÍSTICAS DE LA MÁQUINA VIRTUAL

Cuando el código fuente se compila se obtiene código objeto (bytecode, código intermedio).

Para ejecutarlo en cualquier máquina se requiere tener independencia respecto al hardware concreto que se vaya a utilizar.

Para ello, la máquina virtual aísla la aplicación de los detalles físicos del equipo en cuestión.

Funciona como una capa de software de bajo nivel y actúa como puente entre el bytecode de la aplicación y los dispositivos físicos del sistema.

La Máquina Virtual verifica todo el bytecode antes de ejecutarlo.

La Máquina Virtual protege direcciones de memoria.

**La máquina virtual actúa de puente entre la aplicación y el hardware concreto del equipo donde se instale.**



En el siguiente enlace te presentamos el proceso de instalación de la JVM (Máquina Virtual de Java) y su apariencia.

[PDF Guía rápida de instalación de JVM](#)

(<https://www.adictosaltrabajo.com/2009/03/03/instalacion-jvm/>)..

---

### 3.3.3.2.- Entornos de ejecución.

Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. En ocasiones pertenece al propio sistema operativo, pero también se puede instalar como software independiente que funcionará por debajo de la aplicación. Es decir, es un conjunto de utilidades que permiten la ejecución de programas.

**Se denomina runtime al tiempo que tarda un programa en ejecutarse en la computadora.**

Durante la ejecución, los entornos se encargarán de:

- **Configurar la memoria** principal disponible en el sistema.
- **Enlazar los archivos** del programa con las bibliotecas existentes y con los subprogramas creados. Considerando que las bibliotecas son el conjunto de subprogramas que sirven para desarrollar o comunicar componentes software pero que ya existen previamente y los subprogramas serán aquellos que hemos creado a propósito para el programa.
- **Depurar los programas**: comprobar la existencia (o no existencia) de errores semánticos del lenguaje (los sintácticos ya se detectaron en la compilación).



#### Funcionamiento del entorno de ejecución:

El Entorno de Ejecución está formado por la máquina virtual y los API's (bibliotecas de clases estándar, necesarias para que la aplicación, escrita en algún Lenguaje de Programación pueda ser ejecutada). Estos dos componentes se suelen distribuir conjuntamente, porque necesitan ser compatibles entre sí.

El entorno funciona como intermediario entre el lenguaje fuente y el sistema operativo, y consigue ejecutar aplicaciones.

Sin embargo, si lo que queremos es desarrollar nuevas aplicaciones, no es suficiente con el entorno de ejecución.

Adelantándonos a lo que veremos en la próxima unidad, para desarrollar aplicaciones necesitamos algo más. Ese "algo más" se llama entorno de desarrollo.



#### Autoevaluación

**Señala la afirmación falsa respecto de los entornos de ejecución:**

- Su principal utilidad es la de permitir el desarrollo rápido de aplicaciones.
- Actúa como mediador entre el sistema operativo y el código fuente.

- Es el conjunto de la máquina virtual y bibliotecas necesarias para la ejecución.

Muy bien, lo has entendido perfectamente.

Incorrecto, eso es verdad.

No es correcto, eso es verdad.

## Solución

- 1.** Opción correcta (#answer-66\_63)
  - 2.** Incorrecto (#answer-66\_74)
  - 3.** Incorrecto (#answer-66\_77)
-

### 3.3.3.3.- Java runtime environment.

En esta sección de va a explicar el funcionamiento, instalación, configuración y primeros pasos del Runtime Environment del lenguaje Java (se hace extensible a los demás lenguajes de programación).

#### Concepto.

Se denomina JRE al Java Runtime Environment (entorno en tiempo de ejecución Java). El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma.



#### Componentes.

JRE está formado por:

- Una Máquina virtual Java (JMV o JVM si consideramos las siglas en inglés), que es el programa que interpreta el código de la aplicación escrita en Java.
- Bibliotecas de clase estándar que implementan el API de Java.
- Las dos: JMV y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.

Lo primero es descargarnos el programa JRE. (Java2 Runtime Environment - consultar última versión-). Java es software libre, por lo que podemos descargarnos la aplicación libremente. Una vez descargado, comienza el proceso de instalación, siguiendo los pasos del asistente.



#### Debes conocer

El proceso de descarga, instalación y configuración del entorno de ejecución de programas. En el siguiente enlace, se explican los pasos para hacerlo bajo el sistema operativo Linux.

[Instalación y configuración del JRE de Java.](#)



#### Para saber más

En el siguiente enlace encontrarás un tutorial del lenguaje Java, con sus principales características y órdenes y comandos principales.

<https://www.w3schools.com/java/default.asp>

## 3.4.- Pruebas.



### Caso práctico

María reúne todos los códigos diseñados y los prepara para implementarlos en el equipo del cliente.

Juan se percata de ello, y le recuerda a su amiga que aún no los han sometido a pruebas. Juan se acuerda bien de la vez que le pasó aquello: *hace dos años, cuando fue a presentar una aplicación a sus clientes, no paraba de dar errores de todo tipo... los clientes, por supuesto, no la aceptaron y Juan perdió un mes de duro trabajo y estuvo a punto de perder su empleo...*

“—No tan deprisa María, tenemos que PROBAR la aplicación”.



---

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas.

Normalmente, éstas se realizan sobre un conjunto de datos de prueba, que consisten en un conjunto seleccionado y predefinido de datos límite a los que la aplicación es sometida.

**La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.**

#### Verificación

Consiste en comprobar los resultados de las distintas fases del desarrollo según unos parámetros de calidad que se definan. Responde a la pregunta de ¿El producto está correctamente construido?

#### Validación

Consiste en comprobar los requisitos de software, lo que el usuario espera que hayamos hecho. Responde a la pregunta de ¿El producto construido es correcto? La validación comenzará una vez la verificación se ha completado.

Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

- **PRUEBAS UNITARIAS**

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). JUnit es el entorno de pruebas para Java.

- **PRUEBAS DE INTEGRACIÓN**

Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.

La prueba final se denomina comúnmente Beta Test, ésta se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa).

El período de prueba será normalmente el pactado con el cliente.



## Autoevaluación

**Si las pruebas unitarias se realizan con éxito, ¿es obligatorio realizar las de integración?**

- Sí, si la aplicación está formada por más de cinco módulos diferentes.
- Sí, en cualquier caso.

Incorrecto. Una aplicación formada por dos módulos ya es susceptible de producir errores en su interrelación.

Muy bien, vas captando la idea.

## Solución

1. Incorrecto (#answer-73\_63)
2. Opción correcta (#answer-73\_82)



## Para saber más

Puedes visitar la siguiente página web, donde se detallan los tipos de pruebas que suelen hacer al software y la función de cada una.  
<https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing>

---

### 3.5.- Documentación.



#### Caso práctico

Ada ha quedado dentro de dos días con su cliente. Pregunta a María por todos los dossiers de documentación. La pálida expresión de la joven hace que Ada arda en desesperación: "—¿No habéis documentado las etapas? ¿Cómo voy a explicarle al cliente y sus empleados el funcionamiento del software? ¿Cómo vamos a realizar su mantenimiento?".



**Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.**

¿Por qué hay que documentar todas las fases del proyecto?

Para dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

Tenemos que ir documentando el proyecto en todas las fases del mismo, para pasar de una a otra de forma clara y definida. Una correcta documentación permitirá la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

Distinguimos tres grandes documentos en el desarrollo de software:

#### Documentos a elaborar en el proceso de desarrollo de software

	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
<b>Quedan reflejados:</b>	<ul style="list-style-type: none"> <li>• El diseño de la aplicación.</li> <li>• La codificación de los programas.</li> </ul>	<ul style="list-style-type: none"> <li>• Descripción de la funcionalidad de la aplicación.</li> <li>• Forma de comenzar a ejecutar la aplicación.</li> </ul>	<p>Toda la información necesaria para:</p> <ul style="list-style-type: none"> <li>• Puesta en marcha.</li> <li>• Explotación.</li> </ul>

	<b>GUÍA TÉCNICA</b>	<b>GUÍA DE USO</b>	<b>GUÍA DE INSTALACIÓN</b>
	<ul style="list-style-type: none"> <li>• Las pruebas realizadas.</li> </ul>	<ul style="list-style-type: none"> <li>• Ejemplos de uso del programa.</li> <li>• Requerimientos software de la aplicación.</li> <li>• Solución de los posibles problemas que se pueden presentar.</li> </ul>	<ul style="list-style-type: none"> <li>• Seguridad del sistema.</li> </ul>
<b>¿A quién va dirigido?</b>	Al personal técnico en informática (analistas y programadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
<b>¿Cuál es su objetivo?</b>	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

## 3.6.- Explotación.



### Caso práctico

Llega el día de la cita con la cadena hotelera. Ada y Juan se dirigen al hotel donde se va a instalar y configurar la aplicación. Si todo va bien, se irá implementando en los demás hoteles de la cadena.

Ada no quiere que se le pase ni un detalle: lleva consigo la guía de uso y la guía de instalación.

---

Después de todas las fases anteriores, una vez que las pruebas nos demuestran que el software es fiable, carece de errores y hemos documentado todas las fases, el siguiente paso es la explotación.

Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.

**La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.**

La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.

En el proceso de instalación, los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados.



Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación.

En este momento, se suelen llevar a cabo las Beta Test, que son las últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo.

Una vez instalada, pasamos a la fase de configuración.

En ella, asignamos los parámetros de funcionamiento normal de la empresa y probamos que la aplicación es operativa. También puede ocurrir que la configuración la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación. Y también, si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla. (Si el software es "a medida", lo más aconsejable es que la hagan aquellos que la han fabricado).

Una vez se ha configurado, el siguiente y último paso es la fase de producción normal. La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.

Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente: será el momento crítico del proyecto.

---



## Reflexiona

Realizas un proyecto software por vez primera y no te das cuenta de documentarlo. Consigues venderlo a buen precio a una empresa. Al cabo de un par de meses te piden que actualices algunas de las funciones, para tener mayor funcionalidad. Estás contento o contenta porque eso significa un ingreso extra. Te paras un momento... ¿Dónde están los códigos? ¿Qué hacía exactamente la aplicación? ¿Cómo se diseñó? No lo recuerdas... Probablemente hayas perdido un ingreso extra y unos buenos clientes.

---

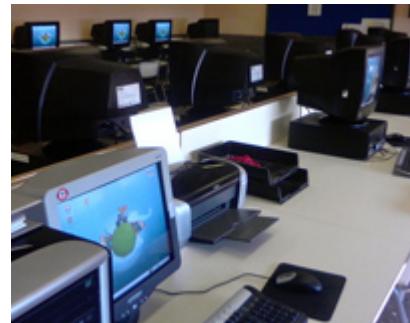
### 3.7.- Mantenimiento.



#### Caso práctico

Ada reúne por última vez durante estas semanas a su equipo. Todos celebran que el proyecto se ha implementado con éxito y que sus clientes han quedado satisfechos.

—Esto aún no ha terminado —comenta Ada—, nos quedan muchas cosas por hacer. Esta tarde me reúno con los clientes. ¿Cómo vamos a gestionar el mantenimiento de la aplicación?



---

Sería lógico pensar que con la entrega de nuestra aplicación (la instalación y configuración de nuestro proyecto en los equipos del cliente) hemos terminado nuestro trabajo.

En cualquier otro sector laboral esto es así, pero el caso de la construcción de software es muy diferente.

**La etapa de mantenimiento es la más larga de todo el ciclo de vida del software.**

Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo. Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó.

Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores.

Por todo ello, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

El mantenimiento se define como el proceso de control, mejora y optimización del software.

Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo.

Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

- **Perfectivos:** Para mejorar la funcionalidad del software.
- **Evolutivos:** El cliente tendrá en el futuro nuevas necesidades. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- **Adaptativos:** Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.

- **Correctivos:** La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).



## Autoevaluación

¿Cuál es, en tu opinión, la etapa más importante del desarrollo de software?

- El análisis de requisitos.
- La codificación.
- Las pruebas y documentación.
- La explotación y el mantenimiento.

Efectivamente. Si esta etapa no está lograda, las demás tampoco lo estarán.

Incorrecto, no necesariamente. Hoy día contamos con ayudas automatizadas (CASE).

No es correcto. Es importante, pero si el análisis no es correcto, no nos servirán de mucho.

No es cierto porque si el software no hace lo que se le pide, éstas no tendrán sentido.

## Solución

1. Opción correcta (#answer-85\_63)
2. Incorrecto (#answer-85\_88)
3. Incorrecto (#answer-85\_91)
4. Incorrecto (#answer-85\_94)

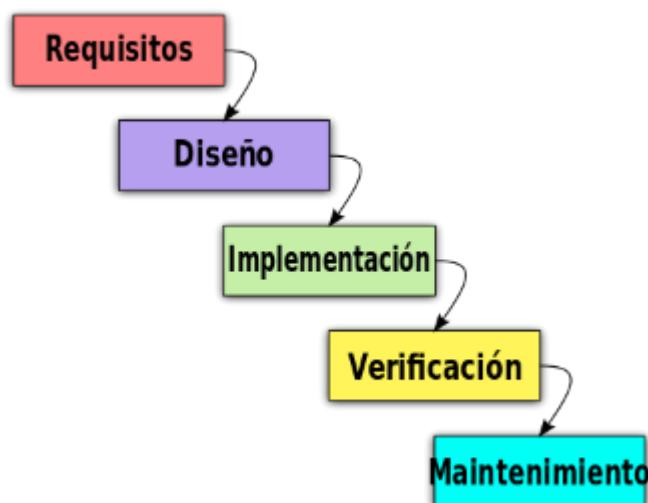
## 4.- Ciclos de vida del software.

Ya hemos visto que la serie de pasos a seguir para desarrollar un programa es lo que se conoce como Ciclo de Vida del Software. Diversos autores han planteado distintos modelos de ciclos de vida, pero los más conocidos y utilizados son los que aparecen a continuación. **Siempre se debe aplicar un modelo de ciclo de vida al desarrollo de cualquier proyecto software.**

### Modelo en Cascada

En este modelo de desarrollo las etapas van en un orden concreto, de tal forma que para empezar una etapa es necesario finalizar la anterior.

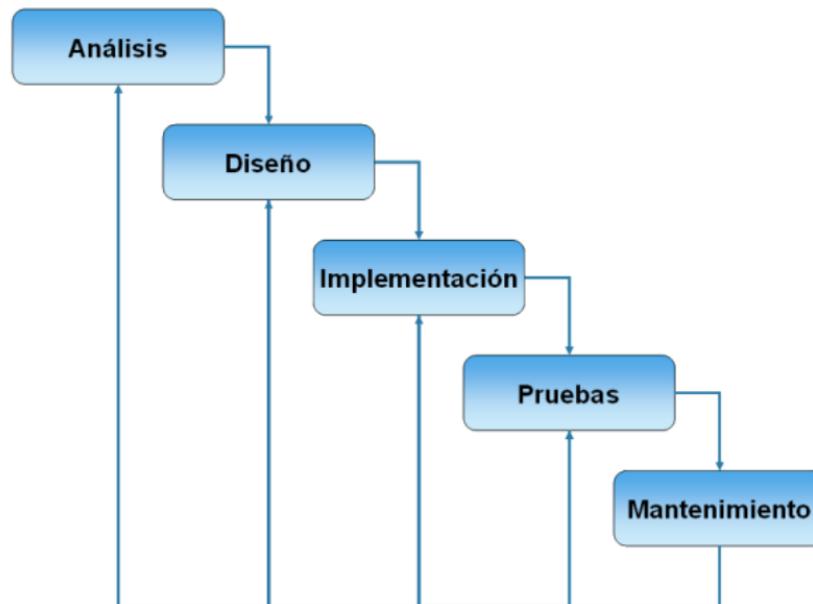
Se recomienda cuando el proyecto es similar a alguno que ya se haya realizado anteriormente con éxito, cuando los requisitos son estables y están bien definidos, y cuando los clientes no necesitan versiones intermedias durante el desarrollo.



### Modelo en Cascada con Realimentación

Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podemos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.

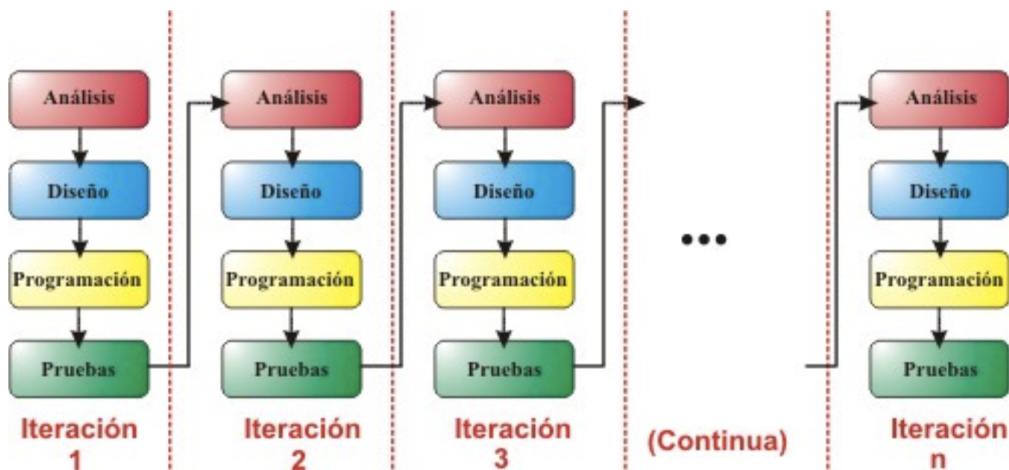
Al igual que el modelo en cascada clásico, es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.



### Modelo Iterativo Incremental

Este modelo entrega el software en partes pequeñas, pero utilizables, llamadas **incrementos**. En general, cada incremento se construye sobre aquél que ya ha sido entregado. Para el desarrollo de cada incremento se puede aplicar el modelo en cascada de forma **iterativa**.

Este modelo es recomendable en aquellos proyectos donde los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios. También cuando se está probando o introduciendo nuevas tecnologías.



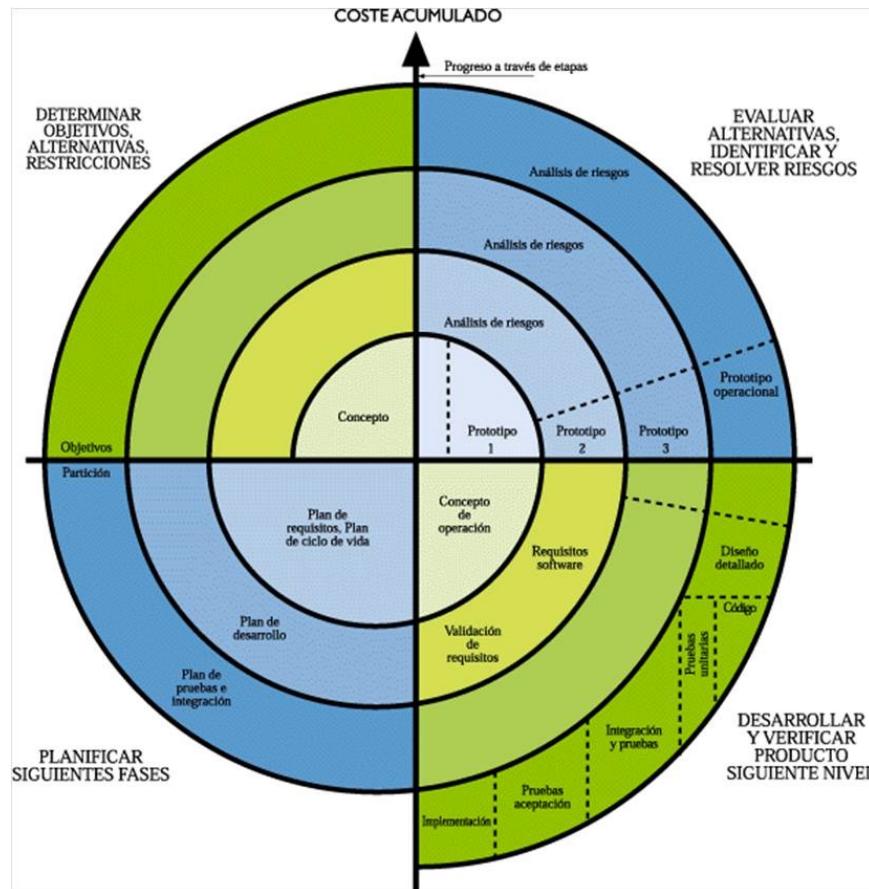
### Modelo en Espiral

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

Se recomienda en proyectos de gran tamaño y que necesitan constantes cambios, proyectos donde el factor de riesgo es alto.

Es un modelo bastante complejo, en el cual se diferencian las siguientes fases:

- 1. Determinar objetivos:** cada ciclo de la espiral comienza con la identificación de los objetivos y de las alternativas existentes para alcanzar dichos objetivos.
- 2. Análisis del riesgo:** se evaluará las alternativas en relación a los objetivos y a las limitaciones de dichos objetivos. Se pueden utilizar prototipos como mecanismo de reducción de riesgos.
- 3. Desarrollar y probar:** desarrollar la solución al problema en este ciclo, y verificar que es aceptable.
- 4. Planificación:** revisar y evaluar todo lo que se ha hecho, y con ello decidir si se continua, entonces hay que planificar las fases del ciclo siguiente.



## Autoevaluación

**Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?**

- Sí.
- No.

Incorrecto, si la aplicación no sufrirá grandes cambios, este modelo no es recomendable.

Efectivamente, por las características de esta aplicación, pensaríamos mejor en el modelo en cascada con realimentación.

## Solución

- 1. Incorrecto (#answer-14\_63)**
- 2. Opción correcta (#answer-14\_96)**

**Si queremos construir una aplicación cuya gestión de los riesgos es una prioridad para el cliente, el modelo más adecuado es en cascada con retroalimentación.**

- Si
- No

Aunque el modelo en cascada con retroalimentación gestiona los riesgos de una forma más eficiente que el modelo en cascada, la respuesta correcta es que para un proyecto con una fuerte gestión de riesgos debemos utilizar un modelo en espiral.

El modelo más adecuado es en espiral, ya que una de sus fortalezas es el análisis de riesgos.

## Solución

- 1. Incorrecto (#answer-14\_326)**
- 2. Opción correcta (#answer-14\_329)**



### Para saber más

En el siguiente enlace podrás ver cómo han ido surgiendo los diferentes modelos (metodologías) de desarrollo a lo largo del tiempo.

<https://www.timetoast.com/timelines/metodologias-para-el-desarrollo-del-software>

---

## 4.1.1.- Scrum

### Scrum

Scrum es un marco de trabajo para desarrollo ágil de software. El desarrollo ágil de software envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo **iterativo e incremental**, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

Scrum es simple pero no es fácil. Desde 2011 ha habido dos grandes cambios en la guía de Scrum, en 2013 y 2016. Primero, se eliminaron algunas cosas, como los diagramas de burndown. Se reforzó el énfasis en el Daily Scrum como elemento de planificación -y no solamente sincronización- y también se cambió el nombre de la reunión de grooming a refinement. Todos estos conceptos los veremos más adelante.

### Principios de Scrum

- El desarrollo incremental de los requisitos del proyecto en bloques temporales cortos y fijos (llamado Sprint ).
- Se da prioridad a lo que tiene más valor para el cliente.
- El equipo se sincroniza diariamente y se realizan las adaptaciones necesarias.
- Tras cada iteración (un mes o menos entre cada una) se muestra al cliente el resultado real obtenido, para que este tome las decisiones necesarias en relación a lo observado.
- Equipos pequeños (de 5 a 9 personas cada uno).



### Para saber más

Para entender los orígenes de Scrum, merece la pena leer el paper *The New New Product Development Game*, en el que se describen las reglas que dieron lugar al que hoy es el framework de desarrollo de Software ágil más usado en el mundo.

### Roles principales de la metodología

- **Product Owner:** se asegura de que el equipo Scrum trabaje de forma adecuada desde la perspectiva del negocio. El Product Owner ayuda al usuario a escribir las historias de usuario, las prioriza, y las coloca en el Product Backlog (posteriormente veremos que son las historias de usuario y el Product Backlog).

- **ScrumMaster:** eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El ScrumMaster no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El ScrumMaster se asegura de que el proceso Scrum se utiliza como es debido. El ScrumMaster es el que hace que las reglas se cumplan.
- **Equipo de desarrollo:** El equipo tiene la responsabilidad de entregar el producto. Es recomendable un pequeño equipo de 3 a 9 personas con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc).
- **Stakeholders** (Clientes, Proveedores, Vendedores, etc): Son las personas que hacen posible el proyecto y para quienes el proyecto producirá el beneficio acordado que justifica su desarrollo. Sólo participan directamente durante las revisiones del "sprint".
- **Administradores** (Managers): Son los responsables de establecer el entorno para el desarrollo del proyecto.

## Documentos principales de la metodología

- **Product backlog:** se trata como un documento de alto nivel para todo el proyecto. Contiene descripciones genéricas de funcionalidades deseables, y están priorizadas. Es abierto y solo puede ser modificado por el product owner. También refleja estimaciones realizadas a grandes rasgos, tanto del valor para el negocio, como del esfuerzo de desarrollo requerido.
- **Sprint backlog:** es el subconjunto de requisitos que serán desarrollados durante el siguiente sprint.
- **Burn down chart:** es una gráfica mostrada públicamente que mide la cantidad de requisitos en el Product backlog pendientes al comienzo de cada sprint.

## Desarrollo del proceso

- **Planificación del sprint:** Al comienzo de un sprint, el equipo de scrum tiene un evento de planificación de sprint. Scrum diario (Daily Standup). Cada día tiene lugar una reunión de estado del proyecto. Dura entre 5 y 15 minutos, se recomienda hacerla de pie para recordar que debe ser una reunión breve. No se interrumpe la dinámica del Standup para elaborar una discusión. Siempre será a la misma hora y en el mismo lugar.
- **Ejecución del sprint:** durante este periodo el equipo trabaja en el desarrollo del producto, finalizando las tareas fijadas.
- **Revisión de sprint:** se realiza al final del un sprint. Se presentan los trabajos completados y su duración no debería ser superior a 4 horas.
- **Retrospectiva del sprint:** los miembros del equipo dejan sus impresiones sobre el sprint recién superado. Tiene un tiempo fijo de 4 horas.

## 5.- Herramientas de apoyo al desarrollo del software.



### Caso práctico

En BK programación ya están manos a la obra. Ada reúne a toda su plantilla para desarrollar el nuevo proyecto.

Ella sabe mejor que nadie que no será sencillo y que habrá que pasar por una serie de etapas. Ana no quiere perderse la reunión, quiere descubrir por qué hay que tomar tantas anotaciones y tantas molestias antes incluso de empezar.

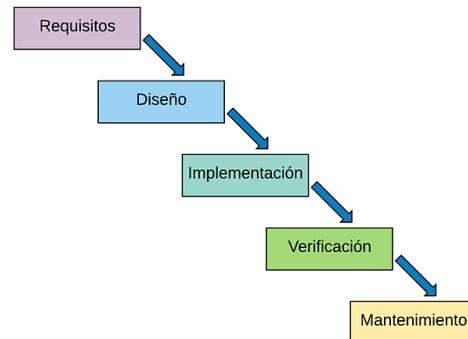


---

Entendemos por **Desarrollo de Software** todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

El proceso de desarrollo, que en un principio puede parecer una tarea simple, consta de una serie de pasos de obligado cumplimiento, pues sólo así podremos garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales (aquellos que van a utilizar el programa).

Como veremos con más detenimiento a lo largo de la unidad, el desarrollo de software es un proceso que conlleva una serie de pasos. Genéricamente, estos pasos son los que podemos ver en la imagen de la derecha. Según el orden y la forma en que se lleven a cabo las etapas hablaremos de diferentes ciclos de vida del software.



Q Etapas clásicas del desarrollo software.

La construcción de software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolle.

---



## Reflexiona

Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito. La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir. ¿Por qué el porcentaje de fracaso es tan grande? ¿Por qué piensas que estas causas son tan determinantes?

Mostrar retroalimentación

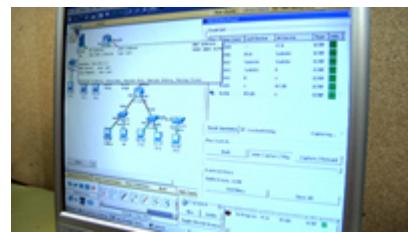
---

Porque los errores en la planificación inicial se propagarán en cascada al resto de etapas del desarrollo.

---

## 5.1.- Herramientas CASE

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.



Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis del mismo, que son las causas principales de los fallos del software.

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

¿En qué fases del proceso nos pueden ayudar?

En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores...

El desarrollo rápido de aplicaciones o **RAD** es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario o entornos de desarrollo integrado completos.

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

En concreto, estas herramientas permiten:

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

### CLASIFICACIÓN

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- **U-CASE**: ofrece ayuda en las fases de planificación y análisis de requisitos.
- **M-CASE**: ofrece ayuda en análisis y diseño.
- **L-CASE**: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

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



## Para saber más

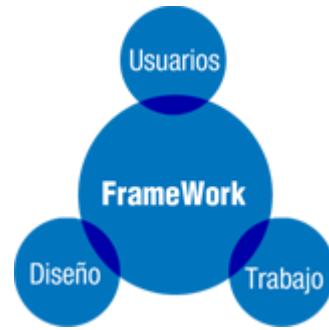
En el siguiente enlace se presenta una ampliación de los tipos y ayudas concretas de la herramientas CASE.

[Ayudas concretas de CASE.](#)

---

## 5.2.- Frameworks.

Un framework es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero. Se trata de una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto. Con el uso de framework podemos pasar más tiempo analizando los requerimientos del sistema y las especificaciones técnicas de nuestra aplicación, ya que la tarea laboriosa de los detalles de programación queda resuelta.



Ventajas de utilizar un framework:

- **Desarrollo rápido** de software.
- **Reutilización** de partes de código para otras aplicaciones.
- **Diseño** uniforme del software.
- **Portabilidad** de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual

Inconvenientes:

- **Gran dependencia del código** respecto al framework utilizado (sin cambiamos de framework, habrá que reescribir gran parte de la aplicación).
- La instalación e implementación del framework en nuestro equipo **consume bastantes recursos** del sistema.



### Para saber más

El uso creciente de frameworks hace que tengamos que estar reciclando constantemente. En el siguiente enlace puedes ver los frameworks más utilizados en nuestros días

<https://www.gajumedia.com/top-frameworks-de-desarrollo-web-en-el-2020/>

Ejemplos de Frameworks:

- **.NET** es un framework para desarrollar aplicaciones sobre Windows. Ofrece el "Visual Studio .net" que nos da facilidades para construir aplicaciones y su motor es el ".Net framework" que permite ejecutar dichas aplicaciones. Es un componente que se instala sobre el sistema operativo.
- **Spring** de Java. Son conjuntos de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones.

