UT1. INTRODUCCIÓN A LA PROGRAMACIÓN

Tabla de contenido

1 Concepto de programa informático. Instrucciones y datos.

Según la **RAE**, el **software** es un **conjunto de programas, instrucciones y reglas informáticas** que permiten ejecutar distintas tareas en una computadora.

Se considera que el software es el **equipamiento lógico e intangible** de un ordenador. En otras palabras, el concepto de software abarca a todas las **aplicaciones informáticas**, así como los procesadores de textos, las hojas de cálculo y los editores de imágenes.

Un **programa informático** es un conjunto de instrucciones escritas en algún lenguaje de programación. El programa debe ser compilado o interpretado para poder ser ejecutado y así cumplir su objetivo.

Un **programa** está formado por una serie de INSTRUCCIONES y de estructuras de DATOS. Un programa al ejecutarse en un ordenador, en general, acepta una serie de datos de ENTRADA y produce unos resultados de SALIDA, ejecutando para ello las instrucciones y manejando las estructuras de datos que componen el programa.

Definición de dato. Representación formal de hechos, conceptos o instrucciones adecuada para su comunicación, interpretación y procesamiento por seres humanos o medios automáticos.

Los tipos de datos se caracterizan por:

- **Dominio de posibles valores**: qué valores puede tomar.
- **Cómo se representan**: cuál es la representación interna y cómo se representan en el lenguaje de programación.
- **Operadores asociados**: qué operaciones o cálculos se pueden realizar con esos datos.

Algunos tipos de datos en Java:

- Entero (byte, short, int, long)
- Booleano (boolean)
- Carácter (char)
- Real (float, double)

ENTRADA → PROGRAMA →SALIDA

2 Ejecución de programas en ordenadores

2.1 Datos, algoritmos y programas.

PROGRAMA = DATOS + INSTRUCCIONES

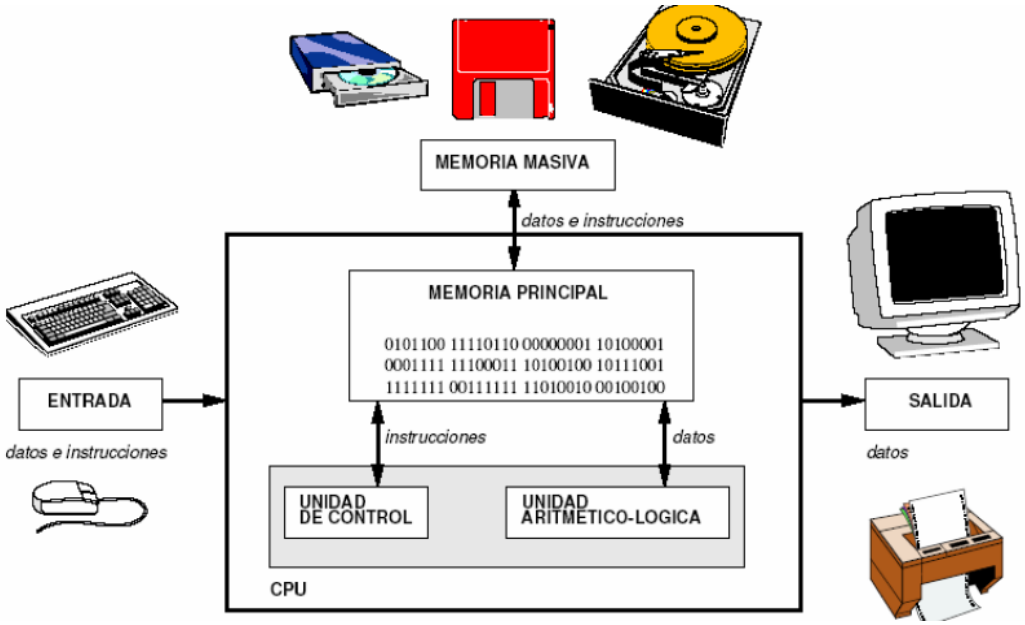
Por otra parte, el término **algoritmo** consiste en una lista ordenada de operaciones que tienen el propósito de buscar la solución a un problema en matemáticas, informática y disciplinas afines (también aplicable en la vida cotidiana).

*Un **programa** es el producto de la traducción de un algoritmo mediante un lenguaje de programación.*

Hardware vs. Software.

El ***Hardware*** son todos los componentes y dispositivos físicos y tangibles que forman una computadora, por ejemplo, la CPU o la placa base, mientras que el ***Software*** es el equipamiento lógico e intangible, como los programas y datos que almacena el ordenador.

2.2 Estructura funcional de un ordenador: procesador, memoria.



Los programas se almacenan en una memoria no volátil (por ejemplo, un disco), para que luego el usuario del ordenador, directa o indirectamente, solicite su ejecución. En el momento de dicha solicitud, el programa es cargado en la memoria de acceso aleatorio o RAM del equipo, bajo el control del software llamado **sistema operativo**, el cual puede acceder directamente al **procesador**. El procesador ejecuta el programa, instrucción por instrucción hasta que termina. La memoria se estructura en posiciones o palabras de memoria a las que se accede por su dirección.

A un programa en ejecución se le suele llamar también **proceso**. Un programa puede terminar su ejecución en forma normal o por causa de un error, dicho error puede ser de software o de hardware.

2.3 Ejecución de un programa

Una vez cargado el programa en la memoria se le cede el control del ordenador:

1. Se lee una instrucción del programa.
2. La unidad de control decodifica la instrucción.
3. La unidad de control envía las señales necesarias para ejecutar la instrucción:
    - Se leen los datos de entrada.
    - Se efectúa una operación con ellos en la ALU (UAL, Unidad Aritmético Lógica).
    - Se almacena el resultado.
4. Se determina cuál es la siguiente instrucción que se debe ejecutar.
5. Se vuelve al paso 1.

2.4 Tipos de software.

2.4.1 BIOS/UEFI

La BIOS (siglas en inglés de basic input/output system; en español "sistema básico de entrada y salida") es un software que localiza y reconoce todos los dispositivos necesarios para cargar el sistema operativo en la memoria RAM; es un software muy básico instalado en la placa base que permite que esta cumpla su cometido. Proporciona la comunicación de bajo nivel, el funcionamiento y configuración del hardware del sistema que, como mínimo, maneja el teclado y proporciona una salida básica (emitiendo pitidos normalizados por el altavoz de la computadora si se producen fallos) durante el arranque. La BIOS usualmente está escrita en lenguaje ensamblador.

La BIOS es un programa tipo *firmware*. El firmware no es más que un bloque de instrucciones para propósitos muy concretos, estas instrucciones están grabadas en una memoria de solo lectura o ROM. Establecen la lógica de más bajo nivel para poder controlar los circuitos electrónicos de un dispositivo de cualquier tipo.

UEFI es la versión avanzada y más actual de las BIOS antiguas.

2.4.2 Sistema Operativo

Un **Sistema Operativo** es el software encargado de ejercer el control y coordinar el uso del hardware entre diferentes programas de aplicación y los diferentes usuarios. Es un administrador de los recursos de hardware del sistema.

En una definición informal, es un **programa** que realiza una distribución ordenada y controlada de los procesadores, memorias y dispositivos de E/S entre los diversos programas que compiten por ellos.

A pesar de que todos nosotros usamos sistemas operativos casi a diario, es difícil definir qué es un sistema operativo. En parte, esto se debe a que los sistemas operativos realizan dos funciones diferentes:

- **Proveer una máquina virtual**, es decir, un entorno en el cual el usuario pueda ejecutar programas de manera conveniente, protegiéndolo de los detalles y complejidades del hardware.
- **Administrar eficientemente** los recursos del computador.

Ejemplos: Windows, Linux, Mac OS…

2.4.3 Aplicaciones

En Informática, una aplicación es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo.

Suele ser una solución informática para la automatización de ciertas tareas complicadas como pueden ser la contabilidad, la redacción de documentos, o la gestión de un almacén. Algunos ejemplos de **programas de aplicación** son los procesadores de textos, hojas de cálculo, y bases de datos.

Las aplicaciones desarrolladas "a medida" suelen ofrecer una gran potencia ya que están exclusivamente diseñadas para resolver un problema específico. Otros, llamados "paquetes integrados de software", ofrecen menos potencia, pero a cambio incluyen varias aplicaciones, como un programa procesador de textos, de hoja de cálculo y de base de datos.

Otros ejemplos de programas de aplicación pueden ser: programas de comunicación de datos, multimedia, para desarrollar presentaciones, para diseño gráfico, cálculo, finanzas, correo electrónico, compresión de archivos, presupuestos de obras, gestión de empresas, etc.

2.5 Código fuente, código objeto y código ejecutable. Máquinas virtuales.

Código fuente: El código fuente de un programa informático (o software) es un conjunto de líneas de texto que son las instrucciones que debe seguir el ordenador para ejecutar dicho programa. Se escribe en un lenguaje de programación determinado.

Código objeto: Se llama código objeto al código que resulta de la *compilación* del código fuente. Para ello, se usa un programa llamado compilador.
El código objeto consiste en lenguaje máquina o bytecode y se distribuye en uno o varios archivos que corresponden a cada código fuente compilado.

Código ejecutable: Para obtener un programa ejecutable se han de enlazar todos los archivos de código objeto con un programa llamado enlazador (linker).

Máquinas virtuales: En lo que se refiere al lenguaje de programación Java, una máquina virtual Java (en inglés Java Virtual Machine, JVM) es un máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java **bytecode**), el cual es generado por el compilador del lenguaje Java.

El código binario de Java no es un lenguaje de alto nivel, sino un intermedio de bajo nivel (para la JVM). Como todas las piezas del rompecabezas Java, fue desarrollado originalmente por Sun Microsystems. Actualmente el propietario es la empresa Oracle.

La máquina virtual Java es en última instancia la que convierte el código bytecode a código nativo del dispositivo final.

La gran **ventaja** de la máquina virtual java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas y así un programa (extensión .class) escrito en un entorno Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para dichos entornos.

*En **resumen**:*

Una de las características más relevante del lenguaje de programación Java es el hecho de que utiliza una máquina virtual para la ejecución de los programas, haciendo esta de intermediaria entre la máquina real (HW + SO) y los programas desarrollados por los programadores.

Esto supuso una gran ventaja a la hora de poder distribuir un mismo código ya compilado en diferentes tipos de máquinas físicas con diferentes sistemas operativos. Basta con compilar el código fuente de alto nivel a código intermedio (que es código máquina para la JVM), también llamado **bytecode**, para que pudiera ser empleado en máquinas virtuales que corren sobre máquinas físicas.

https://www.adictosaltrabajo.com/tutoriales/byte-code

3 Lenguajes de programación.

3.1 Tipos de lenguajes de programación.

Los lenguajes de programación se pueden clasificar según varios criterios. Se pueden encontrar hasta 11 criterios diferentes: Nivel de abstracción, propósito, evolución histórica, manera de ejecutarse, manera de abordar la tarea a realizar, paradigma de programación, lugar de ejecución, concurrencia, interactividad, realización visual y determinismo.

Hay que tener en cuenta también que, en la práctica, la mayoría de los lenguajes no pueden ser puramente clasificados en una categoría, pues surgen incorporando ideas de otros lenguajes y de otras filosofías de programación, pero no importa al establecer las clasificaciones, pues el auténtico objetivo de las mismas es mostrar los rangos, las posibilidades y tipos de lenguajes que hay.

1. Nivel de abstracción.

Según el nivel de abstracción, o sea, según el grado de cercanía a la máquina:

- **Lenguajes de bajo nivel:** La programación se realiza teniendo muy en cuenta las características del procesador. Ejemplo: Lenguajes tipo ensamblador:

  https://www.programacion.com.py/escritorio/ensamblador/ejemplos-de-programas-en-ensamblador-8086

- **Lenguajes de nivel medio:** Permiten un mayor grado de abstracción, pero al mismo tiempo mantienen algunas cualidades de los lenguajes de bajo nivel. Ejemplo: C puede realizar operaciones lógicas y de desplazamiento con bits.
- **Lenguajes de alto nivel:** Más parecidos al lenguaje humano. Manejan conceptos, tipos de datos, etc., de una manera más cercana al pensamiento humano ignorando (abstrayéndose) del funcionamiento de la máquina. Ejemplos: Java, Ruby, PHP, Pascal, LISP, ADA.

Hay quien sólo considera lenguajes de bajo nivel y de alto nivel, (en ese caso, C es considerado de alto nivel).

2. Propósito.

Según el propósito, es decir, el tipo de problemas a tratar con ellos:

- **Lenguajes de propósito general:** Aptos para todo tipo de tareas. Ejemplo: C.
- **Lenguajes de propósito específico:** Hechos para un objetivo muy concreto. Ejemplo: Csound (para crear ficheros de audio).
- **Lenguajes de programación de sistemas:** Diseñados para realizar sistemas operativos o drivers. Ejemplo: C.
- **Lenguajes de script:** Para realizar tareas varias de control y auxiliares. Antiguamente eran los llamados lenguajes de procesamiento por lotes (batch) o JCL ("Job Control Languages"). Se subdividen en varias clases (de shell, de GUI, de programación web, etc.). Ejemplos: Bash (shell), JavaScript (programación web).

3. Evolución histórica.

Con el paso del tiempo, se va incrementando el nivel de abstracción, pero en la práctica, los de una generación no terminan de sustituir a los de la anterior:

- **Lenguajes de primera generación (1GL)**: Código máquina.

- **Lenguajes de segunda generación (2GL)**: Lenguajes ensamblador.
- **Lenguajes de tercera generación (3GL)**: La mayoría de los lenguajes modernos, diseñados para facilitar la programación a los humanos. Ejemplos: C, Java.
- **Lenguajes de cuarta generación (4GL):** se ha dado este nombre a ciertas herramientas que permiten construir aplicaciones sencillas combinando piezas prefabricadas. Hoy se piensa que estas herramientas no son, propiamente hablando, lenguajes. Algunos proponen reservar el nombre de cuarta generación para la programación orientada a objetos.
- **Lenguajes de quinta generación (5GL):** La intención es que el programador establezca qué problema ha de ser resuelto y las condiciones a reunir, y la máquina lo resuelve. Se usan en inteligencia artificial. Ejemplo: Lisp, Prolog.

4. Manera de ejecutarse.

Según la manera de ejecutarse:

- Lenguajes **compilados**: Un programa traductor traduce el código del programa (código fuente) en código máquina (código objeto), generando un fichero con ese código. Otro programa, el enlazador, unirá los ficheros de código objeto del programa principal con los de las librerías para producir el programa ejecutable. Ejemplo: C, Pascal, Fortran.
- Lenguajes **interpretados**: Un lenguaje de programación interpretado es **aquel que el código fuente se ejecuta directamente, instrucción a instrucción**. Es decir, el código no pasa por un proceso de compilación, sino que tenemos un programa llamado intérprete que lee la instrucción en tiempo real, y la ejecuta. Ejemplo: Lisp, Javascript...
- También los hay **mixtos**, como Java, que primero pasan por una fase de compilación en la que el código fuente se transforma en "**bytecode**", y este "bytecode" puede ser ejecutado luego (interpretado) en ordenadores con distintas arquitecturas (procesadores) que tengan todos instalados la "máquina virtual" Java apropiada.

5. Manera de abordar la tarea a realizar.

Según la manera de abordar la tarea a realizar, pueden ser:

- Lenguajes **imperativos**: Indican cómo hay que hacer la tarea, es decir, expresan los pasos a realizar. Ejemplo: C.
- Lenguajes **declarativos**: Indican qué hay que hacer. Ejemplos: Lisp, Prolog. Otros ejemplos de lenguajes declarativos, pero que no son lenguajes de programación propiamente dicha, son HTML (para describir páginas web) o SQL (para consultar bases de datos).

6. Paradigma de programación.

El **paradigma de programación** es el estilo de programación empleado. Los principales son:

- Lenguajes de **programación procedural**: Divide el problema en partes más pequeñas, que serán realizadas por subprogramas (subrutinas, funciones, procedimientos), que se llaman unas a otras para ser ejecutadas. Ejemplos: C, Pascal.
- Lenguajes de **programación orientada a objetos**: Crean un sistema de clases y objetos siguiendo el ejemplo del mundo real, en el que unos objetos realizan acciones y se comunican con otros objetos. Ejemplos: C++, Java.
- Lenguajes de **programación funcional**: La tarea se realiza evaluando funciones (como en Matemáticas) de manera recursiva. Ejemplo: Lisp.
- Lenguajes de **programación lógica**: La tarea a realizar se expresa empleando lógica formal matemática. Ejemplo: Prolog.

7. Lugar de ejecución.

En **sistemas distribuidos**, según dónde se ejecute:

- Lenguajes de **servidor**: Se ejecutan en el servidor. Ejemplo: PHP es el más utilizado en servidores web.
- Lenguajes de **cliente**: Se ejecutan en el cliente. Ejemplo: JavaScript en navegadores web.

8. Concurrencia.

Según admitan o no concurrencia de procesos, esto es, la ejecución simultánea de varios procesos lanzados por el programa:

Un programa **concurrente** se formula utilizando bien un lenguaje concurrente (como Ada), o definiendo independientemente los procesos que constituyen el programa mediante un lenguaje secuencial (como el lenguaje C con Posix (librería pThread)), y haciendo uso de la capacidad de multiprocesado que proporciona un sistema operativo multiprocesador (como UNIX) o un software de comunicación (como MMS)

Un **Lenguaje de Programación será concurrente** si posee las estructuras necesarias para definir y manejar diferentes tareas (hilos de ejecución) dentro de un programa. Ejemplos: Java, Ada. El compilador y el SO serán los responsables de "mapear" la concurrencia lógica del programa sobre el hardware disponible. Incorporan características que permiten expresar la concurrencia directamente, sin recurrir a servicios del SO, bibliotecas, etc. Normalmente incluyen mecanismos de sincronización y comunicación entre procesos. ejemplos: Ada, Java, SR, Occam, PARLOG. El lenguaje C necesita incorporar alguna librería para crear hilos y permitir concurrencia.

https://dgvergel.blogspot.com/2016/10/programacion-concurrente-i.html

9. Interactividad.

Según la interactividad del programa con el usuario u otros programas:

- Lenguajes **orientados a sucesos (eventos)**: El flujo del programa es controlado por la interacción con el usuario o por mensajes de otros programas/sistema operativo, como editores de texto, interfaces gráficos de usuario (GUI) o kernels. Ejemplo: VisualBasic, lenguajes de programación declarativos.
- Lenguajes **no orientados a sucesos (eventos)**: El flujo del programa no depende de sucesos exteriores, sino que se conoce de antemano, siendo los procesos batch el ejemplo más claro (actualizaciones de bases de datos, colas de impresión de documentos, etc.). Ejemplos: Lenguajes de programación imperativos.

10. Realización visual.

Según la realización visual o no del programa:

- Lenguajes de **programación visual**: El programa se realiza moviendo bloques de construcción de programas (objetos visuales) en un interfaz adecuado para ello. No confundir con entornos de programación visual, como Microsoft Visual Studio y sus lenguajes de programación textuales (como Visual C#). Ejemplo: Mindscript, Alice, Scratch.
- Lenguajes de **programación textual**: El código del programa se realiza escribiendo cada instrucción. Ejemplos: C, Java, Lisp.
-

11. Determinismo.

Según se pueda predecir o no el siguiente estado del programa a partir del estado actual:

- Lenguajes **deterministas**. Ejemplos: Todos los anteriores.
- Lenguajes **probabilísticos** o **no deterministas:** Sirven para explorar grandes espacios de búsqueda, (como gramáticas), y en la investigación teórica de hipercomputación. Ejemplo: mutt (generador de texto aleatorio).

3.2 Características de los lenguajes más difundidos a lo largo del tiempo.

BASIC. Durante mucho tiempo se ha considerado un buen lenguaje para comenzar a aprender, por su sencillez, aunque se podía tender a crear programas poco legibles. A pesar de esta "sencillez" hay versiones muy potentes, incluso para programar en entornos gráficos como Windows.

COBOL. Fue muy utilizado para negocios (para crear software de gestión que tuviese que manipular grandes cantidades de datos). Sigue usándose en el campo de la Banca.

FORTRAN. Concebido para ingeniería, operaciones matemáticas, etc. También va quedando desplazado.

ENSAMBLADOR. Muy cercano al código máquina (es un lenguaje de "bajo nivel"), pero sustituye las secuencias de ceros y unos (bits) por palabras más fáciles de recordar, como MOV, ADD, CALL o JMP.

PASCAL. El lenguaje estructurado por excelencia, y que en algunas versiones tiene una potencia comparable a la del lenguaje C.

C. Uno de los mejor considerados en la historia de la programación (junto con C++ y Java, que mencionaremos a continuación), porque no es demasiado difícil de aprender y permite un grado de control del ordenador muy alto, combinando características de lenguajes de alto y bajo nivel. Además, es muy transportable: existe un estándar, el ANSI C, lo que asegura que se pueden convertir programas en C de un ordenador a otro o de un sistema operativo a otro con bastante menos esfuerzo que en otros lenguajes.

C++. Un lenguaje desarrollado a partir de C, que permite Programación Orientada a Objetos.

Java. Desarrollado a su vez a partir de C++, elimina algunos de sus inconvenientes y ha alcanzado una gran difusión gracias a su uso en Internet. Al igual que C++ es Orientado a Objetos.

Python. Es un lenguaje de programación de alto nivel, interpretado, orientado a objetos, usado en desarrollo web, Big Data, Inteligencia Artificial…

Kotlin. Lenguaje de programación que trabaja sobre la máquina virtual de Java y que ha sido elegido por Google como su lenguaje idóneo para programación en Android.

4 Proceso de obtención de código ejecutable a partir del código fuente, herramientas implicadas

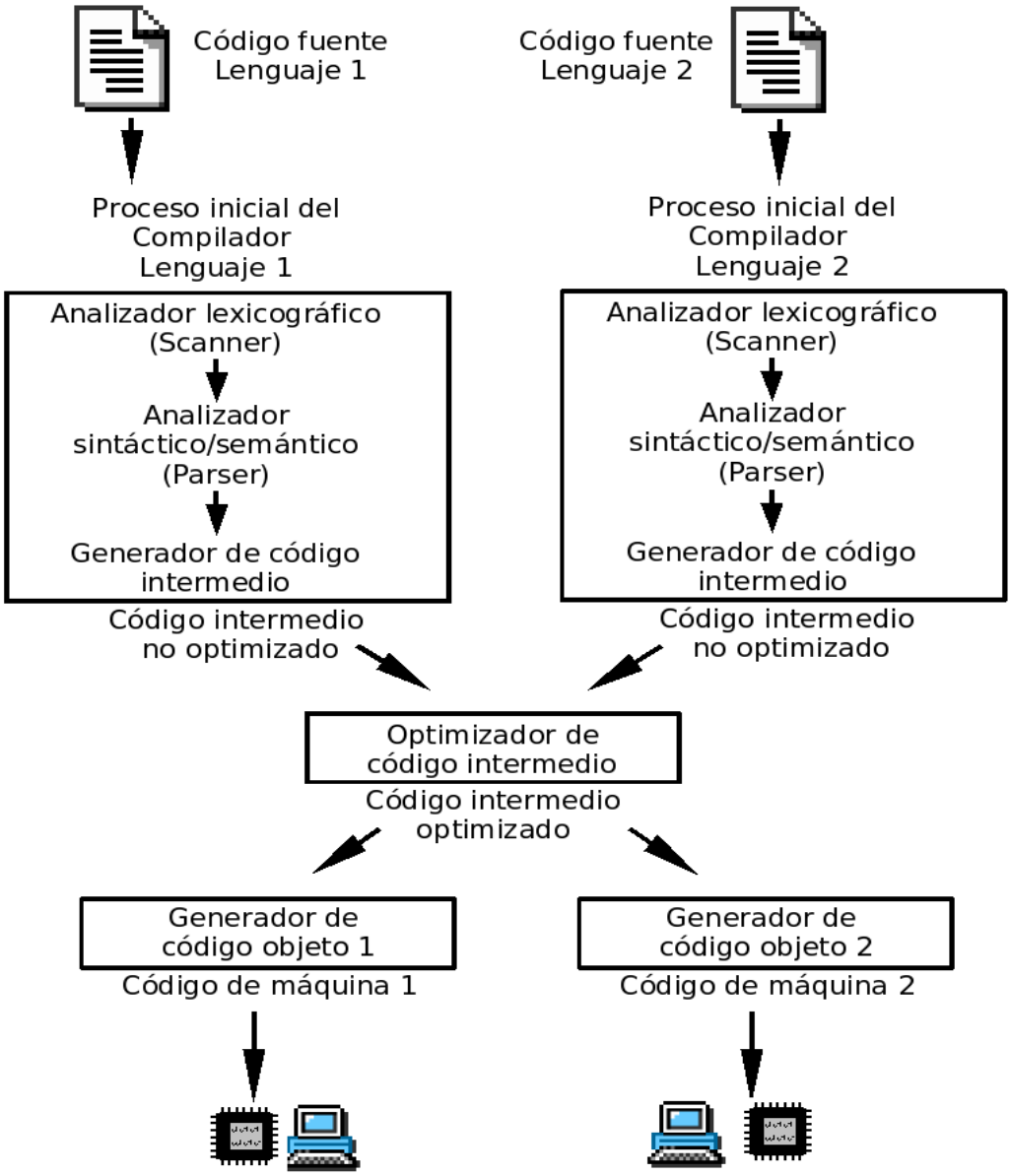
4.1 Editores

Un **editor** es un programa informático que nos permite crear, modificar (editar) y guardar archivos de texto en la memoria secundaria del ordenador.

Los editores de programas especializados nos ofrecen distintas facilidades para escribir el código: marcar las palabras reservadas en distinto color, terminar las palabras reservadas automáticamente, ayuda online, etc.

4.2 Compiladores

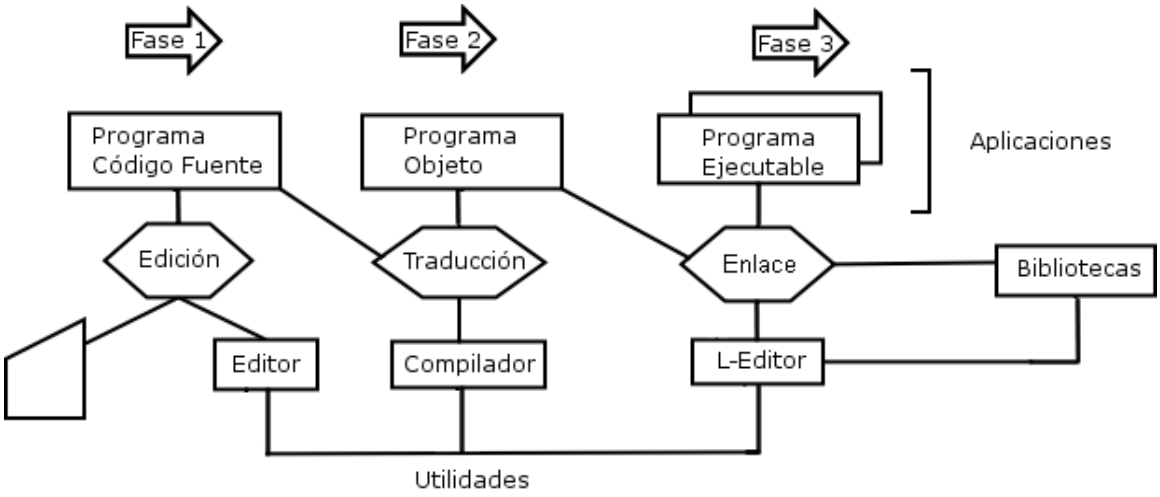
Un **compilador** es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un lenguaje intermedio (bytecode) o simplemente texto. Este proceso de traducción se conoce como **compilación**.



4.3 Enlazadores

En programación, un **enlazador** (en inglés, *linker*) es un módulo o programa que une los ficheros de código objeto (generados en la compilación), la información de todos los recursos necesarios (librerías/bibliotecas), elimina los recursos que no se necesitan y enlaza el código objeto con sus librerías. Finalmente produce el fichero ejecutable o una librería.

Existen programas que se enlazan dinámicamente, esto significa que este proceso se hace en el momento que se carga el programa.



5 Errores en el desarrollo de programas

En el proceso de desarrollo de un programa, así como durante todo el ciclo de vida de una aplicación, se pueden producir distintos errores. Estos errores pueden ser:

Errores sintácticos. Se detectarán en la fase de compilación del código.

Errores de ejecución. Se detectarán una vez compilado el programa, cuando intentemos ejecutarlo. Un ejemplo de esto es la división por cero. Suponga que tiene la instrucción siguiente:

```
velocidad = kilometros / horas
```

Si la variable horas tiene un valor de 0, se produce un error en tiempo de ejecución en la operación de división. El programa se debe ejecutar para que se pueda detectar este error y si horas contiene un valor válido, no se producirá el error.

Errores lógicos. Los errores lógicos son errores que impiden que un programa haga lo que estaba previsto. El código puede compilarse y ejecutarse sin errores, pero el resultado de una operación puede generar un resultado no esperado.

Por ejemplo, puede tener una variable llamada *nombre* y establecida inicialmente en una cadena vacía. Después, en el programa puede concatenar nombre con otra variable denominada *apellido* para mostrar un nombre completo. Si olvida asignar un valor a nombre, solo se mostrará el apellido, no el nombre completo como pretendía.

Los errores lógicos son los más difíciles de detectar y corregir, las herramientas de depuración facilitan el trabajo de encontrar los errores lógicos.

http://wiki.elhacker.net/bugs-y-exploits/introduccion/errores-de-programacion-comunes

6 Importancia de la reutilización de código

En programación, reutilización de código es el uso de software existente para desarrollar un nuevo software. La reutilización de código ha sido empleada desde los primeros días de la programación. Los programadores siempre han reusado partes de un código, hojas de cálculo, funciones o procedimientos.

La idea es que parte o todo el código de un programa informático escrito una vez, sea o pueda ser usado en otros programas. La reutilización de código es una técnica común que intenta ahorrar tiempo y energía, reduciendo el trabajo redundante.

Las bibliotecas o librerías de software son un buen ejemplo. Al utilizarlas se está reutilizando código.

El software más fácilmente reutilizable tiene ciertas características: modularidad, bajo acoplamiento, alta cohesión, ocultación de información, etc.

7 Introducción a la ingeniería del software.

7.1 Proceso software y ciclo de vida del software

El software de ordenadores es el producto que diseñan y construyen los ingenieros del software. El software es un elemento lógico, no físico. El software tiene unas características considerablemente distintas a las del hardware:

1. El software se desarrolla, no se fabrica.
2. La mayoría del software se realiza a medida.

La **Ingeniería del Software** es una disciplina o área de la Informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad.

La Ingeniería del Software aborda todas las fases de ciclo de vida de cualquier tipo de sistema de información.
Sistema de Información. 'Sistema de información' (SI) es un conjunto de elementos orientados al tratamiento y administración de datos e información, organizados y listos para su posterior uso, generados para cubrir una necesidad (objetivo). Dichos elementos formarán parte de alguna de estas categorías:

- Personas.
- Datos.
- Actividades o técnicas de trabajo.
- Recursos materiales en general, típicamente recursos informáticos y de comunicación, aunque no tienen por qué ser de este tipo obligatoriamente.

Existen múltiples definiciones de Ingeniería del Software, casi tantas como autores que han

escrito sobre el tema. Vamos a mencionar la de **IEEE** ("Institute of Electrical and Electronics Engineers"):

"Ingeniería del Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software. Es decir, la aplicación de la Ingeniería al software".

El **ciclo de vida** de un sistema abarca toda la vida del sistema, desde su concepción hasta que deja de utilizarse. Se habla también de ciclo de vida del desarrollo software, abarcando las siguientes etapas.

7.1.1 Fases del desarrollo de una aplicación

7.1.1.1 Análisis

Para comprender la naturaleza de los programas a construir, el analista, (= ingeniero del software) debe comprender el dominio de información del software, así como las funciones requeridas, comportamiento, rendimiento e interconexión.
En esta fase, el analista ha de recoger en colaboración con el cliente, toda la información relativa a los requisitos del *sistema a desarrollar*: su función, rendimiento e interfaces. Hay que analizar costes, recursos y definir tiempos para las siguientes tareas a realizar.

7.1.1.2 Diseño

El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de programa:

- Estructura de datos
- Arquitectura de software
- Representaciones de interfaz
- Algoritmo (detalle procedimental)

"El proceso de diseño traduce requisitos en una representación del software donde se pueda evaluar su calidad antes de que comience la codificación".

7.1.1.3 Codificación

El diseño se debe traducir en una forma legible por la máquina. Esta tarea se lleva a cabo en el paso de Generación de código (Codificación). Si el diseño se ha realizado de una forma detallada, la generación de código se realiza mecánicamente.

7.1.1.4 Pruebas

Cuando se ha finalizado la generación del código comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software. Se realizan

las pruebas para la detección de errores y para asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos.

7.1.1.5 Documentación

Cada una de las fases por las que pasa un proyecto de Software debe ir acompañada de la correspondiente documentación.

La documentación que se realiza de un software se divide en dos:

- Documentación para el usuario del SW
- Documentación para el equipo de desarrollo

7.1.1.6 Explotación

La etapa de explotación transcurre una vez que el sistema está a disposición del usuario.

7.1.1.7 Mantenimiento

El software una vez entregado sufrirá cambios. Los motivos de estos cambios pueden ser:

- Errores que pudieran encontrarse.
- El software debe adaptarse a cambios en su entorno externo (por ejemplo, un cambio de sistema operativo).
- El software requiere mejoras funcionales o de rendimiento.

El proceso de soporte y mantenimiento implica que el programa que se va a modificar tendrá que pasar de nuevo por cada una de las fases precedentes.

7.2 Modelos de proceso de desarrollo software

La estrategia de desarrollo (modelo o proceso) que se elige para realizar una aplicación concreta recibe el nombre de:

Ciclo de vida = modelo de desarrollo

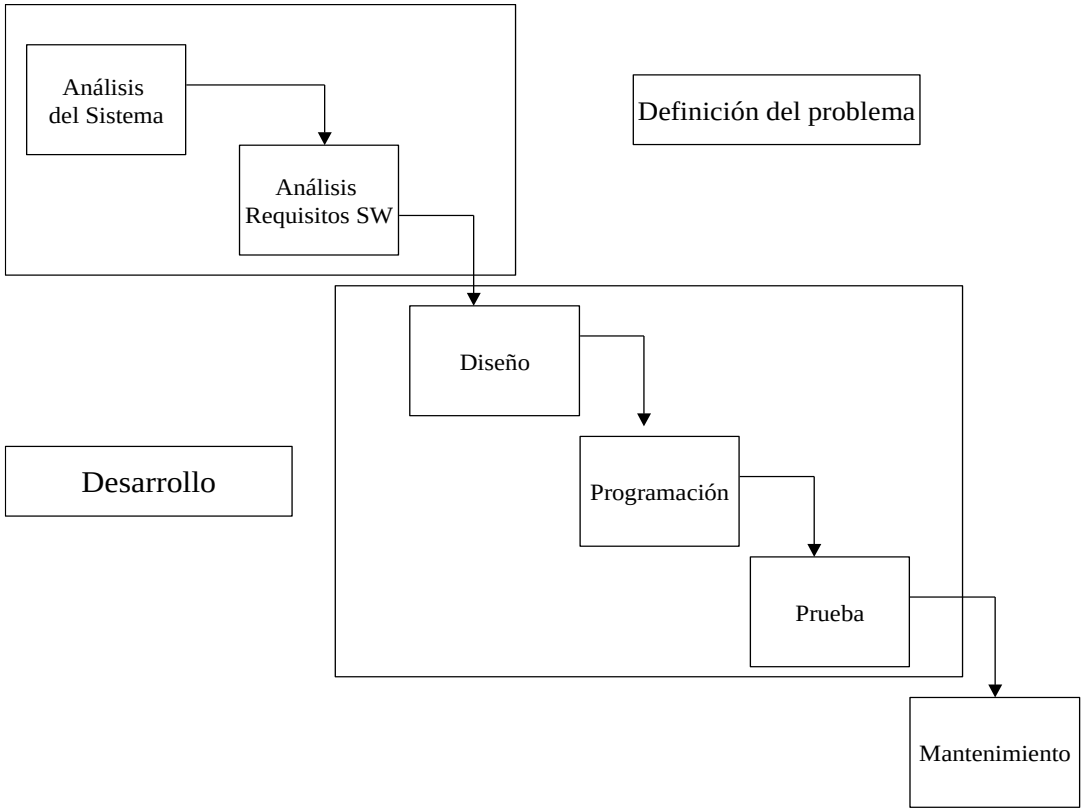
Cada modelo de desarrollo representa un intento de ordenar la actividad de desarrollo del software. Todos los modelos esperan contribuir al control y la coordinación de un proyecto de software real.
Dependiendo de la naturaleza de un proyecto de desarrollo, puede haber razones para

adoptar un modelo en lugar de otro o incluso una combinación de ellos.

7.2.1 Cascada (lineal secuencial)

En este modelo se distinguen una serie de fases, que pueden diferir en algunos autores y libros en nombre y número, pero todas tienen en común dos aspectos:

- Implantación ascendente.
- Progresión lineal y secuencial de una fase a otra.



En el modelo en cascada hasta que no acaba una fase no comienza la siguiente.

Razones por las que a veces no se debe utilizar el modelo lineal:

- A veces es difícil que el usuario declare explícitamente todos los requisitos al principio del proyecto. El modelo lineal secuencial requiere que esto ocurra y resulta complicado compaginarlo con la incertidumbre existente al comienzo de muchos proyectos.
- Una versión visible del trabajo realizado en el proyecto no estará disponible hasta que todo el trabajo esté realizado, haciendo que si ocurre un error ya sea demasiado tarde.

A pesar de estos problemas el modelo lineal secuencial proporciona una plantilla para el desarrollo del proyecto. Siempre es mejor que realizar un desarrollo al azar.

El modelo lineal es el modelo más antiguo y el que más se ha utilizado en la ingeniería del software.

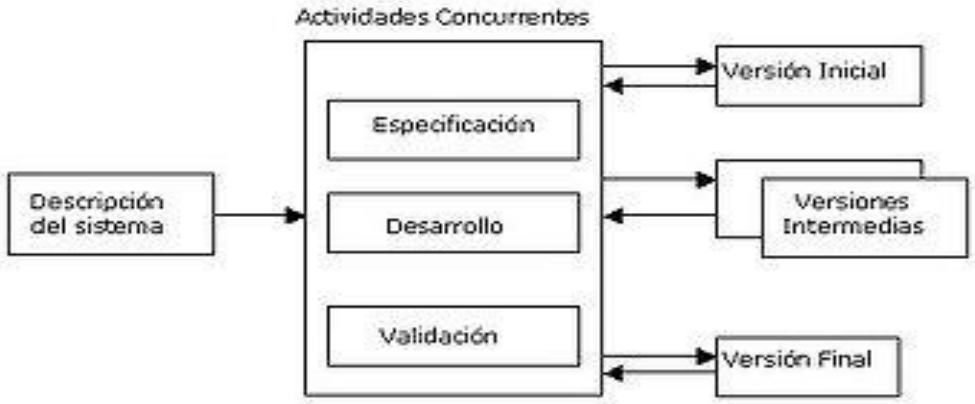
¿Cuándo se debe utilizar?:

"El modelo lineal (cascada) se puede utilizar y es aconsejable utilizarlo cuando los requisitos se han entendido y definido correctamente".

7.2.2 Evolutivo

En el desarrollo en cascada no se tiene en cuenta la naturaleza evolutiva del software. Se debe tener una especificación totalmente detallada de **TODOS** los requerimientos que debe satisfacer el software que desarrollemos para poder iniciar las diferentes etapas de desarrollo.

El **desarrollo evolutivo** se basa en la idea de desarrollar una implementación inicial e ir refinándola a través de diferentes versiones hasta desarrollar un sistema software que satisfaga todos los requerimientos del cliente.



Un enfoque evolutivo para el desarrollo de software suele ser más efectivo que el desarrollo en cascada ya que desde un principio se le entrega al cliente una versión que satisface los requerimientos principales.

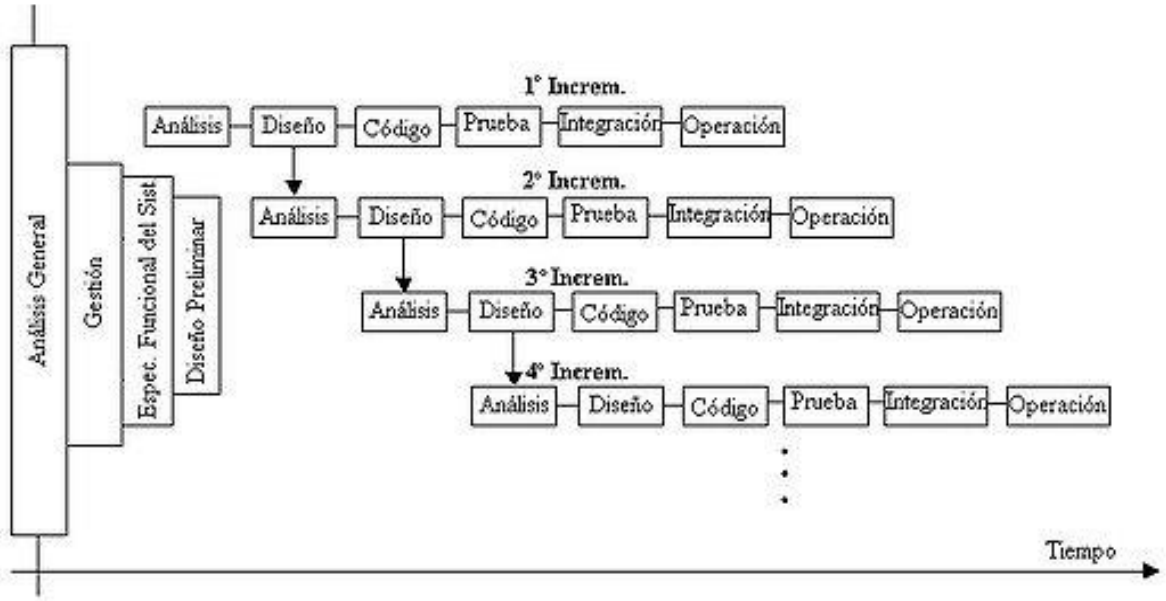
El modelo **iterativo incremental y el modelo en espiral** son dos modelos de tipo evolutivo.

7.2.2.1 Iterativo incremental

Como ya hemos dicho, el modelo en cascada requiere tener una especificación de requerimientos totalmente detallada para poder iniciar la etapa del diseño. Si a lo largo del proceso de desarrollo se cambian los requisitos hay que rehacer parte del trabajo de diseño e implementación para poder hacer frente a los nuevos cambios.

El desarrollo evolutivo permite que los requerimientos no estén totalmente especificados para comenzar con el desarrollo del software, esto hace que el software desarrollado pueda estar mal estructurado y sea difícil de mantener, pero permite adaptarse a los cambios en los requisitos.

Vamos a ver un modelo de proceso basado en el desarrollo evolutivo: el **modelo iterativo incremental.**



Siguiendo este modelo, los clientes especifican a grandes rasgos los servicios que tiene que proporcionar el sistema software que se quiere desarrollar. Entonces se definen varios incrementos para desarrollar cada uno un subconjunto de la funcionalidad del sistema software dando prioridad a los requerimientos más importantes.

Cada incremento puede desarrollarse siguiendo un modelo en cascada típico, por lo que al inicio de cada incremento se debe tener una especificación detallada de la funcionalidad que se pretende desarrollar en ese incremento.

Cuando el incremento se completa, se entrega y los clientes pueden utilizarlo a la espera del siguiente incremento. Esto significa que los clientes obtienen una parte de la funcionalidad del sistema software que necesitan de manera temprana, y se quedan a la espera de nuevos incrementos que mejoren la funcionalidad del sistema software.

Este proceso de desarrollo incremental tiene varias **ventajas** en contraste al desarrollo en cascada:

- Los clientes no tienen que esperar hasta la etapa final para sacar provecho del sistema software, ya que el primer incremento satisface los requerimientos principales.

- Los clientes pueden utilizar cada incremento para analizar nuevos requerimientos para incrementos posteriores.

- Existe un bajo riesgo de un fallo total del proyecto, ya que los errores encontrados en un incremento pueden arreglarse en el incremento posterior.

Este enfoque de desarrollo es aconsejable si el software que se pretende desarrollar posee una serie de funcionalidades bien definidas que se pueden desarrollar con total independencia. Los incrementos a considerar en cada vuelta ya vienen establecidos desde las etapas iniciales.

7.2.2.2 Iterativo en Espiral

Se trata de un modelo iterativo como el anterior, pero en este caso, **tras cada vuelta** se lleva a cabo un análisis de riesgos y se determinan los objetivos a conseguir en las siguientes iteraciones. Es un modelo muy abierto a cambios de requisitos, incluso una vez puesto en marcha el proyecto. El modelo en espiral corre el riesgo de alargarse excesivamente en el tiempo y aumentar el coste de su desarrollo.

En cada ciclo se realizan las siguientes cuatro etapas:

Planificación: determinar los objetivos principales a conseguir, discutir sobre las condiciones del proyecto y buscar las distintas alternativas que se pueden llegar a utilizar para llegar a cumplir esos objetivos.

Se debe realizar un calendario y un cronograma, determinar los recursos disponibles y estimar los costes, además de seleccionar las vías de comunicación entre desarrollador y cliente durante el proceso.

Análisis de riesgos: identificar todos los riesgos posibles para que el plan se ponga en marcha con las condiciones óptimas. Estos riesgos se deben registrar, evaluar y analizar, para posteriormente realizar simulaciones y utilizar prototipos que nos permitan estudiar su impacto y visibilizar las maneras de acabar con ellos.

Desarrollo: empezar a trabajar con las funcionalidades del *software*, ampliando el prototipo y llevando a cabo las acciones necesarias para llegar a los objetivos. En este proceso, es necesario hacer pruebas continuas para acreditar el funcionamiento deseado del programa, hasta que tengamos la seguridad de que puede ser utilizado en un entorno empresarial.

Evaluación: el cliente evalúa si los objetivos propuestos anteriormente se han cumplido, y se analiza si se están solucionando los riesgos identificados, realizando un seguimiento de los mismos.

A continuación, se empezará la planificación de la siguiente fase.

El modelo en espiral es un tipo de modelo basado en el desarrollo iterativo. Se **diferencia** del modelo iterativo incremental en que más que representarlo como una secuencia de actividades se representa como una espiral donde cada ciclo en la espiral representa una fase del proceso del software. La diferencia principal entre este modelo y los vistos hasta ahora es la evaluación del riesgo. El riesgo es todo aquello que pueda ir mal. Por ejemplo, si la intención es utilizar un lenguaje de programación, un riesgo posible es que los compiladores disponibles no produzcan código objeto eficiente. Los riesgos originan problemas en el proyecto como, por ejemplo, el exceso de costes. Por lo tanto, la disminución de los riesgos es una actividad muy importante.

https://www.hostingplus.cl/blog/metodologia-de-espiral-fases-y-desarrollo/

https://www2.deloitte.com/es/es/pages/technology/articles/que-es-el-desarrollo-en-espiral.html

7.2.2.3 Modelos ágiles.

Se trata de modelos que están ganando gran presencia en los desarrollos software. Muy centrados en la satisfacción del cliente, muestran gran flexibilidad a la aparición de nuevos requisitos, incluso durante el desarrollo de la aplicación. Al tratarse de metodologías evolutivas, el desarrollo es incremental, pero estos incrementos son cortos y están abiertos al solapado de unas fases con otras.

La comunicación entre los integrantes del equipo de trabajo y de estos con el cliente son constantes. Un ejemplo de metodología ágil es **Scrum**.

7.3 Roles que participan en el desarrollo software

7.3.1 Arquitecto de software

Es la persona encargada de decidir cómo va a realizarse el proyecto y cómo va a cohesionarse.
Tiene un conocimiento profundo de las tecnologías, frameworks, librerías… Decide la forma en la que se va a desarrollar el proyecto y los recursos con los que debe contar.

7.3.2 Jefe de proyecto

Dirige el proyecto. Muchas veces es el propio arquitecto, un analista con experiencia o una persona que se dedica exclusivamente a este puesto.
Su tarea es gestionar el equipo y los tiempos en los que se va desarrollando el proyecto, así como la relación con el cliente.

7.3.3 Analista de sistemas

Es un rol antiguo dentro del equipo de proyecto. Se trata de la persona que participa tanto en el análisis como en el diseño del proyecto. Tiene que ser una persona con experiencia ya que participa en el análisis de requisitos, en el diseño, y puede tener que relacionarse con el cliente.

7.3.4 Analista programador

Sería un programador senior, con experiencia, que en proyectos pequeños puede hacer las tareas de análisis y diseño. En proyectos grandes colaboraría en estas tareas junto a los principales responsables.

7.3.5 Programador

Su función es conocer el lenguaje de programación en el que se va a codificar el programa y programar las tareas que le han sido asignadas por los analistas.

Todas estas figuras son susceptibles de combinarse y recaer en la misma persona algunos de los roles.

7.4 Herramientas CASE (Computer Aided Software Engineering).

***C**omputer **A**ided **S**oftware **E**ngineering*

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software: Análisis, Diseño, Implementación e Instalación.

Una herramienta CASE suele incluir:

- Un **diccionario de datos** para almacenar información sobre los datos de la aplicación.
- **Herramientas de diseño** para dar apoyo al análisis de datos.
- **Herramientas** que permitan desarrollar el **modelo de datos** corporativo, así como los esquemas conceptual y lógico.
- **Herramientas** para desarrollar los **prototipos** de las aplicaciones.

El uso de las herramientas CASE puede mejorar la productividad en el desarrollo de una aplicación de bases de datos.

7.4.1 Ejemplos de Herramientas Case

ERwin:

ERwin es una herramienta para el diseño de bases de datos, que proporciona productividad en el diseño, generación, y mantenimiento de aplicaciones. Desde un modelo lógico de los requerimientos de información, hasta el modelo físico perfeccionado para las características específicas de la base de datos diseñada, además ERwin permite visualizar la estructura, los elementos importantes, y optimizar el diseño de la base de datos. Genera automáticamente las tablas y miles de líneas de procedimientos almacenados y disparadores para los principales tipos de base de datos.

EasyCASE

EasyCASE Profesional - Es un producto para la generación de esquemas de base de datos e ingeniería inversa - trabaja para proveer una solución comprensible para el diseño, consistencia y documentación del sistema en conjunto.

Esta herramienta permite automatizar las fases de análisis y diseño dentro del desarrollo de una aplicación, para poder crear las aplicaciones eficazmente – desde el procesamiento de transacciones a la aplicación de bases de datos de cliente/servidor, así como sistemas de tiempo real.

Oracle Designer:

Oracle Designer es un conjunto de herramientas para guardar las definiciones que necesita el usuario y automatizar la construcción rápida de aplicaciones cliente/servidor gráficas. Integrado con Oracle Developer, Oracle Designer provee una solución para desarrollar sistemas empresariales de segunda generación.

Todos los datos ingresados por cualquier herramienta de Oracle Designer, en cualquier fase de desarrollo, se guardan en un repositorio central, habilitando el trabajo fácil del equipo y la dirección del proyecto.

En el lado del Servidor, Oracle Designer soporta la definición, generación y captura de diseño de los siguientes tipos de bases de datos, por conexión de Oracle:

8 Frameworks

Un **framework** es una estructura de ayuda al programador, con la que 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 frameworks 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 casi resuelta.

Ventajas de utilizar un framework:

- Desarrollo rápido de software.
- Reutilización de partes de código para otras aplicaciones.
- Diseño uniforme del software.

Inconvenientes de utilizar un framework:

- Dependencia del framework.
- Posible dependencia del código respecto al framework utilizado (si cambiamos de framework, habrá que reescribir parte de la aplicación).
- Consumo de recursos. La instalación y uso del framework en nuestro equipo consume bastantes recursos del sistema.

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.
- **Spring** de Java. Es un conjunto de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones Java.
- **Angular**. Framework de Javascript para aplicaciones web.

9 Arquitectura del Software

La arquitectura del software es el diseño de más alto nivel en un sistema. Decide qué elementos van a formar el sistema y cómo se van a relacionar entre ellos.

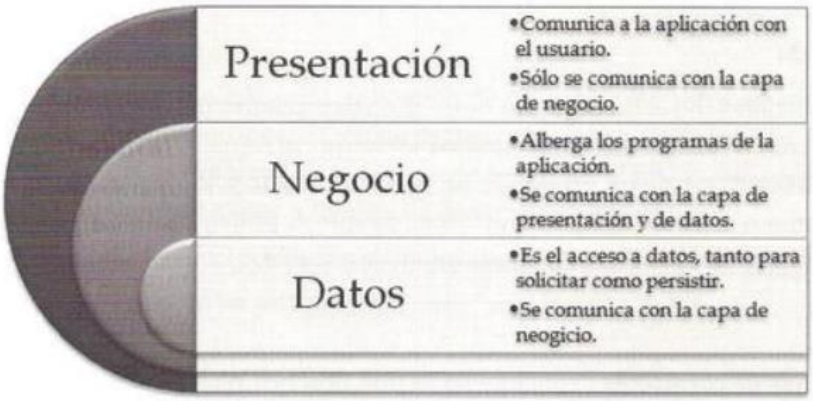
9.1 Patrones de diseño

Se les llama también patrones de desarrollo. Son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error. Los patrones de diseño ayudan a estar seguros de la validez del código, ya que son soluciones que funcionan y han sido probadas por muchísimos desarrolladores.

9.2 Desarrollo en tres capas

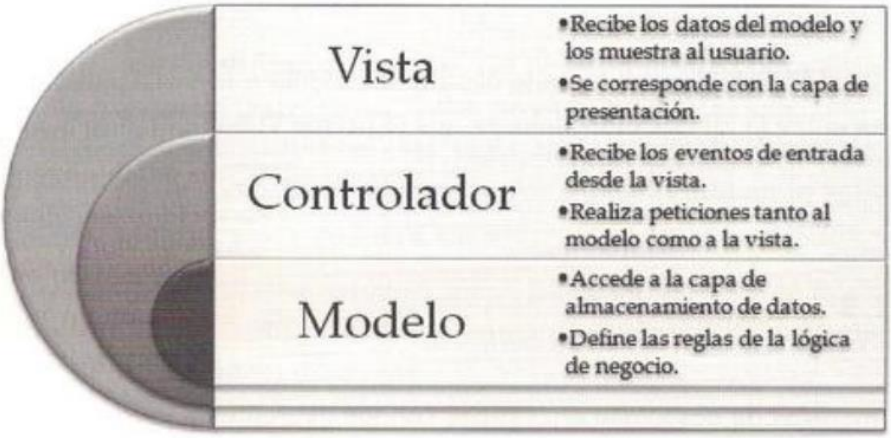
El desarrollo en capas surge para separar la lógica de la aplicación, el diseño y la presentación de los datos al usuario. Por tanto, tendremos 3 capas:

- Presentación al usuario
- Lógica de negocio
- Acceso a datos



9.2.1 Modelo Vista Controlador

Dentro del diseño por capas, nos encontramos con el modelo Vista Controlador **MVC**. Este modelo propone tres componentes para las capas del desarrollo del software, organiza el código basándose en la funcionalidad y no en las características del componente en sí mismo.



Para mostrar los datos del modelo en las vistas se realizan "bindings/enlaces" que relacionan diferentes componentes de la vista a propiedades y campos de las entidades de los datos a los que tiene acceso el modelo.

9.2.2 Modelo Vista VistaModelo

MVVM puede pensarse como una ampliación del MVC. Muchos frameworks actuales usan MVVM proveyendo al MVC framework un **viewmodel**.

De la misma manera que en MVC, los eventos de la vista son recogidos por el controlador, pero a diferencia del MVC los datos son obtenidos y actualizados a través del VistaModelo (viewmodel), ocultando así al modelo de la vista, dejando al modelo como una mera representación de las entidades para la persistencia de los datos.

Ejemplo: **MVVM Android.** En Android, MVC se refiere al **patrón predeterminado donde una actividad actúa como un controlador y los archivos XML son vistas**. **MVVM trata tanto las clases de actividad como los archivos XML como vistas**, y las clases de **ViewModel son donde escribe su lógica empresarial**.