1
Desarrollo de software

Índice:

1. El programa informático

Definición de programa informático:

"Un programa informático es un conjunto de instrucciones que se ejecutan de manera secuencial con el objetivo de realizar una o varias tareas en un sistema"

Un programa informático es creado por un programador en un lenguaje determinado y será compilado y ejecutado por un sistema. Cuando un programa es ejecutado, el procesador ejecuta el código compilado del programa instrucción a instrucción.

1.1. Interacción con el sistema

Para entender como un programa informático interactúa con un sistema, veremos el funcionamiento de una única instrucción en una máquina conocida como es el simulador de von Newmann.

El procesador ejecutará las instrucciones una a una y cada instrucción se realizará mediante una serie de microinstrucciones.

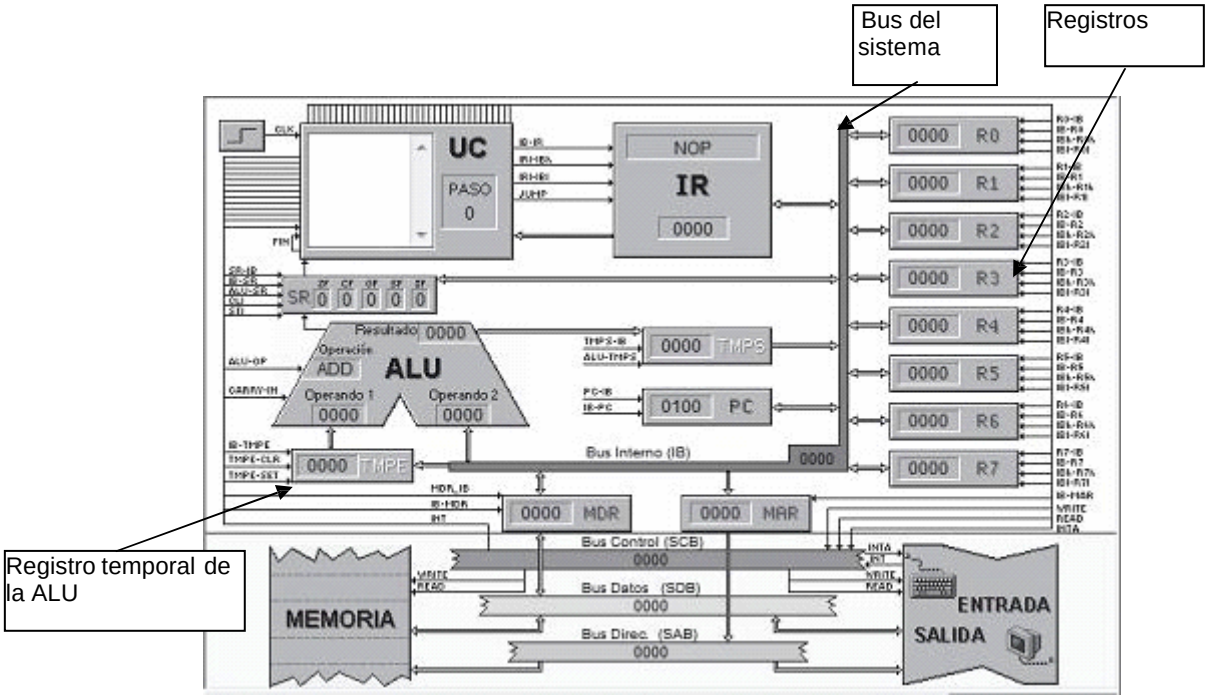


Figura 1.1. Máquina de von Newmann

Hagamos un recorrido que efectúa una instrucción de modo conceptual, más adelante profundizaremos en la interpretación que hace el sistema. Imaginemos que nuestro programa pide por teclado dos números y los suma. La instrucción que realizará nuestro programa será:

```
c = a + b;
```

El ordenador tendrá reservada una cantidad de posiciones de memoria definidas por el tipo de variable que se corresponden con las variables de

nuestra instrucción. Es decir, las variables “a”, “b” y “c” tendrán unas posiciones de memoria definidas que es donde el sistema almacenará el valor de las variables.

El procesador no puede ejecutar esa instrucción de un solo golpe, la ALU (Aritmetic Logic Unit) tiene un número limitado de operaciones (SUMA, RESTA, AND, OR, NOT…)

Las reglas del sistema son:

Y La ALU solo puede realizar una operación a la vez.

Y El registro temporal de la ALU, el bus y los registros solo almacenan un dato a la vez.

Si suponemos que las posiciones de las variables “a”, “b” y “c” se corresponden con los registros R1, R2 y R3, las microinstrucciones que tendrá que realizar nuestra máquina serán las siguientes:

```
R1-Bus; Bus-ALU_Temp; R2-Bus; ALU_SUMAR; ALU-Bus; Bus-R3
```

Hoy en día, con los procesadores modernos, el funcionamiento, aunque muy similar en esencia, puede variar sobre todo en lo que se refiere a las reglas. Sin embargo, hay cosas que no varían: el programa se sigue almacenando en memoria no volátil y se sigue ejecutando en memoria de acceso aleatorio, al igual que todas las variables utilizadas.

Veamos otras partes de la máquina de von Newmann:

Unidad Central de Proceso, conocida como CPU (Central Process Unit) o procesador. Es el principal elemento del ordenador, su trabajo consiste en coordinar y ejecutar todas las instrucciones que se leen de la memoria RAM. Sus elementos principales son:

- **UAL (Unidad Aritmético-Lógica) o ALU** (Arithmetic-Logical Unit). Componente de la **CPU** que realiza todas las operaciones aritméticas elementales (suma, resta, multiplicación, división) y las operaciones lógicas (por ejemplo, la comparación de dos valores).

- **UC (Unidad de Control).** Componente de la **CPU** encargado de controlar las acciones del resto de las unidades, interpretando y ejecutando las instrucciones en la secuencia adecuada.

- **Registros del microprocesador.** Son bancos de memoria de alta velocidad muy especializados, donde se almacenan algunos datos e instrucciones de un programa mientras se ejecuta.

  Generalmente, la ALU dispone de los registros de propósito general, que sirven para tareas como procesos de acumulación, contador de índices de bucles, transferencias de datos, o manipulación de bits.

  La UC también dispone de registros como: **Contador de programa (PC)**

que contiene la dirección de la siguiente instrucción que debe ejecutarse, **Registro de Estado (SR)** que guarda información sobre el estado actual de las operaciones que se están realizando, **Registro de instrucción (IR)** que contiene la instrucción que se está ejecutando y **Registro de puntero de pila** que mantiene la dirección necesaria para las operaciones de pila.

Memoria Central, también llamada **memoria principal o memoria interna.** Es un dispositivo de almacenamiento de información. Existen dos tipos principales de memoria central: la memoria ROM, de sólo lectura, donde se almacena software del sistema de forma permanente y la memoria RAM para almacenamiento temporal de información. Desde el punto de vista del programador, esta última es la más interesante. En ella se guardan todos los datos, tanto de entrada como resultados intermedios y definitivos de las operaciones realizadas durante la ejecución de los programas, así como las instrucciones que forman los propios programas, utilizando para ello los distintos segmentos ya nombrados.

El ordenador necesita obtener la información con la que trabajará de algún sitio y, además, poder comunicar los resultados de sus operaciones. Para ello dispone de las **Unidades de Entrada/Salida,** a las que se conectan todos los demás aparatos que transmiten la información entre el ordenador y su entorno.

Dispositivos o periféricos de entrada. Son los componentes hardware encargados de introducir la información desde el exterior para su posterior proceso. Un ejemplo de dispositivo que se utiliza para la entrada es el teclado, conocido comúnmente como *dispositivo estándar de entrada*

Dispositivos o periféricos de salida. Son los componentes hardware encargados de hacer llegar al exterior los resultados procedentes de los procesos realizados en el sistema informático. Un ejemplo de dispositivo utilizado para la salida es el monitor, conocido comúnmente como *dispositivo estándar de* salida

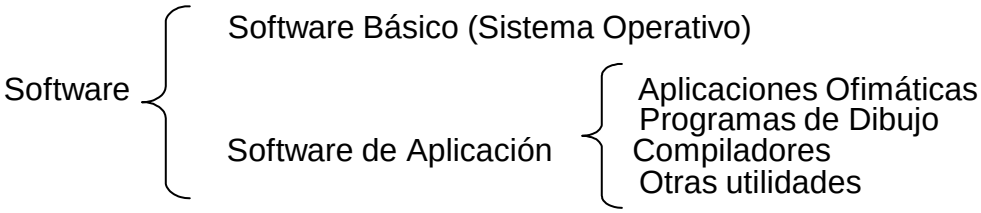
Existen, además, dispositivos que permiten la comunicación en ambos sentidos, y conocidos como dispositivos de entrada y salida, por ejemplo el módem

Dispositivo de almacenamiento auxiliar, a veces llamado dispositivo de almacenamiento secundario o memoria auxiliar. Son unidades de almacenamiento masivo de información mucho más lentas que la memoria central y con una capacidad mayor. Estas memorias son utilizadas para guardar datos y programas de forma permanente a diferencia de la memoria RAM, que se borra cuando se apaga el ordenador. Cuando la CPU requiera estos datos, deberán transferirse desde el dispositivo de almacenamiento auxiliar a la memoria central para su proceso. El principal elemento de almacenamiento auxiliar de información es el disco duro.

1.2. El elemento lógico (Software)

El software de un sistema informático es el conjunto de elementos lógicos, programas, datos, información, etc. que hacen posible el uso y funcionamiento de los ordenadores

Se puede decir que los elementos básicos del software son los datos y las órdenes o instrucciones. Si el software forma parte del sistema informático, deberá almacenarse en un soporte físico como la memoria central o la memoria secundaria.



Se puede clasificar el elemento lógico como **software básico** y **software de aplicación.**

El software básico es el conjunto de programas imprescindibles para gestionar el hardware del sistema informático. Este conjunto de programas es conocido con el nombre de **Sistema Operativo**

Sistema Operativo es la herramienta lógica del sistema informático que controla el funcionamiento del equipo físico y gestiona todos los recursos haciendo transparente al usuario las características físicas de la máquina, facilitando de este modo su uso y mejorando su eficacia. Son ejemplos de sistemas operativos, MS-DOS, UNIX, Linux, OS/2, OS-400, Windows XP, Mac OS-X

El **Software de Aplicación** está formado por un conjunto de programas diseñados con el objetivo de que los ordenadores realicen trabajos específicos, facilitando al usuario la realización de sus actividades. Son aplicaciones tales como herramientas ofimáticas (Microsoft Office, OpenOffice), programas de dibujo (CorelDraw, Microsoft Visio), programas para la realización de nóminas, o para llevar la contabilidad de la empresa (Contaplus, Contawin). Pertenecen también a este grupo de software de aplicación las herramientas de programación para los distintos lenguajes, necesarias para la realización de programas.

2. Lenguajes de programación

Definición de lenguaje de programación:

"Un lenguaje de programación es un conjunto de instrucciones, operadores y reglas de sintaxis y semánticas, que se ponen a disposición del programador para que éste pueda comunicarse con los dispositivos de hardware y software existentes".

El idioma artificial que constituyen los operadores, instrucciones y reglas tiene el objetivo de facilitar la tarea de crear programas, permitiendo con un mayor nivel de abstracción realizar las mismas operaciones que se podrían realizar utilizando código máquina.

En un principio, todos los programas eran creados por el único código que el ordenador era capaz de entender: el **código máquina**, un conjunto de 0s y 1s de grandes proporciones. Este método de programación, aunque absolutamente efectivo y sin restricciones, convertía la tarea de programación en una labor sumamente tediosa, hasta que se tomó la solución de establecer un nombre a las secuencias de programación más frecuentes (códigos pnemotécnicos), estableciéndolas en posiciones de memoria concretas, a cada una de estas secuencias nominadas se las llamó instrucciones, y al conjunto de dichas instrucciones, **lenguaje ensamblador**.

Más adelante, empezaron a usar los ordenadores científicos de otras ramas, con muchos conocimientos de física o química, pero sin nociones de informática, por lo que les era sumamente complicado el uso del lenguaje ensamblador; como un modo de facilitar la tarea de programar, y no como un modo de facilitar el trabajo al programador informático, nace el concepto de **lenguaje de alto nivel** con Fortran (FORmula TRANslation) como primer debutante.

Los lenguajes de alto nivel son aquellos que elevan la abstracción del código máquina lo más posible, para que programar sea una tarea más liviana, entendible e intuitiva. No obstante, nunca hay que olvidar que, usemos el lenguaje que usemos, el compilador hará que de nuestro código solo lleguen 1s y 0s a la máquina.

2.1. Clasificación y características

La cantidad de lenguajes de programación es elevadísima, cada uno con unas características y objetivos determinados, tal cantidad de lenguajes hace necesario establecer unos criterios para clasificarlos. Los criterios que clasifican los lenguajes de programación se corresponden con sus características principales.

Se pueden clasificar mediante una gran variedad de criterios, se podrían establecer hasta once criterios válidos diferentes con los que catalogar un lenguaje de programación. Algunos de dichos criterios pudieran ser redundantes, puesto que se encuentran incluidos explícitamente dentro de otros, como el determinismo o el propósito.

Nosotros vamos a clasificar los lenguajes de programación siguiendo 3 criterios globales y reconocidos: el nivel de abstracción, la forma de ejecución y el paradigma.

Nivel de abstracción

Llamamos nivel de abstracción al modo en que los lenguajes se alejan del

código máquina y se acercan cada vez más a un lenguaje similar a los que utilizamos diariamente para comunicarnos. Cuanto más alejado esté del código máquina, de mayor nivel será el lenguaje. Dicho de otro modo, podría verse el nivel de abstracción como la cantidad de "capas" de ocultación de código máquina que hay entre el código que escribimos y el código que la máquina ejecutará en último término.

Lenguajes de bajo nivel

- **Primera generación**: solo hay un lenguaje de primera generación: el *código máquina*. Cadenas interminables de secuencias de 1s y 0s que conforman operaciones que la máquina puede entender sin interpretación alguna.

Lenguajes de medio nivel

- **Segunda generación**: los lenguajes de segunda generación tienen definidas unas instrucciones para realizar operaciones sencillas con datos simples o posiciones de memoria. El lenguaje clave de la segunda generación es sin duda el *lenguaje ensamblador*.

  Aunque en principio pertenecen a la tercera generación, y por tanto serían lenguajes de alto nivel, algunos consideran a ciertos lenguajes de programación procedimental lenguajes de medio nivel, con el fin de establecerlos en una categoría algo inferior que los lenguajes de programación orientada a objetos, que aportan una mayor abstracción.

Lenguajes de alto nivel

- **Tercera generación**: la gran mayoría de los lenguajes de programación que se utilizan hoy en día pertenecen a este nivel de abstracción, en su mayoría, los lenguajes del paradigma de programación orientada a objetos, son lenguajes de propósito general que permiten un alto nivel de abstracción y una forma de programar mucho más entendible e intuitiva, donde algunas instrucciones parecen ser una traducción directa del lenguaje humano. Por ejemplo, nos podríamos encontrar una línea de código como ésta: IF contador = 10 THEN STOP. No parece que esta sentencia esté muy alejada de cómo expresaríamos en nuestro propio lenguaje Si el contador es 10, entonces para.

- **Cuarta generación**: son lenguajes creados con un propósito específico, al ser un lenguaje tan específico permite reducir la cantidad de líneas de código que tendríamos que hacer con otros lenguajes de tercera generación mediante procedimientos específicos. Por ejemplo, si tuviésemos que resolver una ecuación en un lenguaje de tercera generación, tendríamos que crear diversos y complejos métodos para poder resolverla, mientras que un lenguaje de cuarta generación dedicado a este tipo de problemas ya tiene esas rutinas incluidas en el propio lenguaje, con lo que solo tendríamos que invocar la instrucción que realiza la operación que necesitamos.

- **Quinta generación**: también llamados lenguajes naturales, pretenden abstraer más aún el lenguaje utilizando un lenguaje natural con una base de conocimientos que produce un sistema basado en el conocimiento. Pueden establecer el problema que hay que resolver y las premisas y condiciones que hay que reunir para que la máquina lo resuelva. Este tipo de lenguajes los podemos encontrar frecuentemente en inteligencia artificial y lógica.

Forma de ejecución

Dependiendo de cómo un programa se ejecute dentro de un sistema, podríamos definir tres categorías de lenguajes:

- **Lenguajes compilados**: un programa traductor (compilador) convierte el código fuente en código objeto y otro programa (enlazador) unirá el código objeto del programa con el código objeto de las librerías necesarias para producir el programa ejecutable.

- **Lenguajes interpretados**: ejecutan las instrucciones directamente, sin que se genere código objeto, para ello es necesario un programa intérprete en el sistema operativo o en la propia máquina donde cada instrucción es interpretada y ejecutada de manera independiente y secuencial. La principal diferencia con el anterior es que se traducen a tiempo real solo las instrucciones que se utilicen en cada ejecución, en vez de interpretar todo el código, se vaya a utilizar o no.

- **Lenguajes virtuales**: los lenguajes virtuales tienen un funcionamiento muy similar al de los lenguajes compilados, pero, a diferencia de éstos, no es código objeto lo que genera el compilador, sino un **bytecode** que puede ser interpretado por cualquier arquitectura que tenga la máquina virtual que se encargará de interpretar el código bytecode generado para ejecutarlo en la máquina; aunque de ejecución lenta (como los interpretados), tienen la ventaja de poder ser multisistema y así un mismo código bytecode sería válido para cualquier máquina.

Paradigma de programación

El paradigma de programación es un enfoque particular para la construcción de software, un estilo de programación que facilita la tarea de programación o añade mayor funcionalidad al programa dependiendo del problema que haya que abordar. Todos los paradigmas de programación pertenecen a lenguajes de alto nivel, y es común que un lenguaje pueda usar más de un paradigma de programación.

- **Paradigma imperativo**: describe la programación como una secuencia de instrucciones que cambian el estado del programa, indicando cómo realizar una tarea. (C, Basic)

- **Paradigma declarativo**: especifica o declara un conjunto de premisas y condiciones para indicar qué es lo que hay que hacer y no necesariamente cómo hay que hacerlo. (SQL)

- **Paradigma procedimental**: el programa se divide en partes más pequeñas, llamadas funciones y procedimientos, que pueden comunicarse entre sí. Permite reutilizar código ya programado y solventa el problema de la programación spaghetti. (C, Cobol, Pacal)

- **Paradigma orientado a objetos**: encapsula el estado y las operaciones en objetos, creando una estructura de clases y objetos que emula un modelo del mundo real, donde los objetos realizan acciones e interactúan con otros objetos. Permite la herencia e implementación de otras clases, pudiendo establecer tipos para los objetos y dejando el código más parecido al mundo real con esa abstracción conceptual. (C++, Java, Smalltalk)

- **Paradigma funcional**: evalúa el problema realizando funciones de manera recursiva, evita declarar datos haciendo hincapié en la composición de las funciones y en las interacciones entre ellas. (Lisp, Scheme)

- **Paradigma lógico**: define un conjunto de reglas lógicas para ser interpretadas mediante inferencias lógicas. Permite responder preguntas planteadas al sistema para resolver problemas. (Prolog)

3. Obtención de código ejecutable

Como se ha venido comentando a lo largo de todo el capítulo, nuestro programa, esté programado en el lenguaje que esté y se quiera ejecutar en la arquitectura que sea, necesita ser traducido para poder ser ejecutado (con la excepción del lenguaje máquina). Por lo que, aunque tengamos el código de nuestro programa escrito en el lenguaje de programación escogido, no podrá ser ejecutado a menos que lo traduzcamos a un idioma que nuestra máquina entienda.

3.1. Tipos de código (fuente, objeto y ejecutable)

El código de nuestro programa es manejado mediante programas externos comúnmente asociados al lenguaje de programación en el que está escrito nuestro programa, y a la arquitectura en donde queremos ejecutar dicho programa.

Para ello, deberemos definir los distintos tipos de código por los que pasará nuestro programa antes de ser ejecutado por el sistema.

- **Código fuente**: el código fuente de un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación de alto nivel. Es decir, es el código en el que nosotros como programadores escribimos nuestro programa mediante un editor de texto.

- **Código objeto**: el código objeto es el código binario resultante de compilar el código fuente. El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora. Si se trata de un lenguaje de programación compilado, el código objeto pasará a ser código

máquina, mientras que si se trata de un lenguaje de programación virtual, será código bytecode.

- **Código ejecutable**: el código ejecutable es el resultado obtenido de enlazar nuestro código objeto con las librerías o bibliotecas. Este código ya es nuestro programa ejecutable, programa que se ejecutará directamente en nuestro sistema o sobre una máquina virtual en el caso de los lenguajes de programación virtuales. También es conocido como código máquina y ya sí es directamente inteligible por la computadora.

Cabe destacar que, si nos encontrásemos programando en un lenguaje de programación interpretado, nuestro programa no pasaría por el compilador y el enlazador, sino que solo tendríamos un código fuente que pasaría por un intérprete interno del sistema operativo o de la máquina que realizaría la compilación y ejecución línea a línea en tiempo real.

3.2. Compilación

Aunque el proceso de obtener nuestro código ejecutable pase tanto por un compilador como por un enlazador, se suele llamar al proceso completo "compilación". Todo este proceso se lleva a cabo mediante dos programas: el compilador y el enlazador. Mientras que el enlazador solamente une el código objeto con las librerías, el trabajo del compilador es mucho más completo.

Fases de compilación

- **Análisis lexicográfico:** se leen de manera secuencial todos los caracteres de nuestro código fuente, buscando palabras reservadas, operaciones, caracteres de puntuación y agrupándolos todos en cadenas de caracteres que se denominan lexemas.

- **Análisis sintáctico-semántico:** agrupa todos los componentes léxicos estudiados en el análisis anterior en forma de frases gramaticales. Con el resultado del proceso del análisis sintáctico, se revisa la coherencia de las frases gramaticales, si su "significado" es correcto, si los tipos de datos son correctos, si los arrays tienen el tamaño y tipo adecuados, y así consecutivamente con todas las reglas semánticas de nuestro lenguaje.

- **Generación de código intermedio:** una vez finalizado el análisis, se genera una representación intermedia a modo de pseudoensamblador con el objetivo de facilitar la tarea de traducir al código objeto.

- **Optimización de código**: revisa el código pseudoensamblador generado en el paso anterior optimizándolo para que el código resultante sea más fácil y rápido de interpretar por la máquina.

- **Generación de código:** genera el código objeto de nuestro programa en un código de lenguaje máquina relocalizable, con diversas posiciones de memoria sin establecer, ya que no sabemos en qué parte de la memoria volátil se va a ejecutar nuestro programa.

- **Enlazador de librerías:** como se ha comentado anteriormente, se enlaza nuestro código objeto con las librerías necesarias, produciendo en último término nuestro código final o código ejecutable.

Veamos una ilustración que muestra el proceso de compilación:

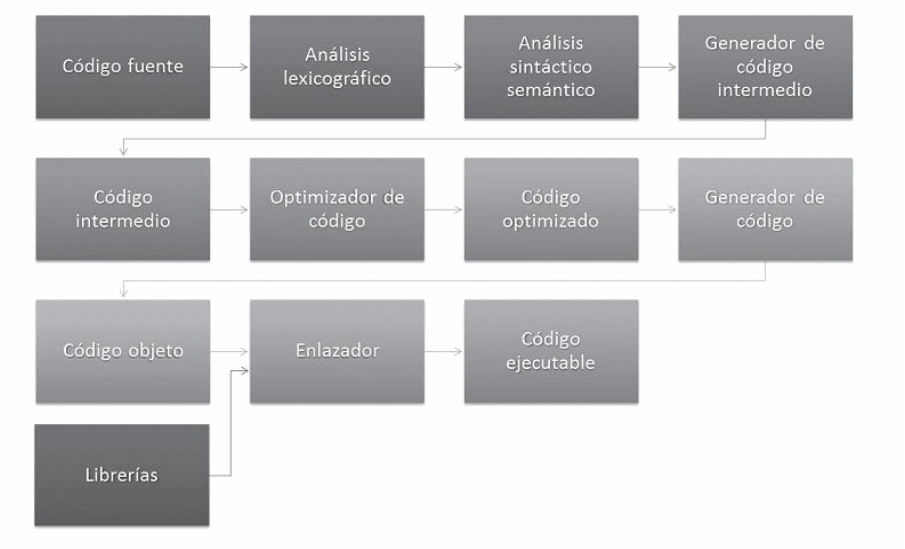


Figura 1.2. Obtención de código ejecutable

4. Procesos de desarrollo

El desarrollo de un software o de un conjunto de aplicaciones pasa por diferentes etapas (llamadas ciclo de vida del programa) desde que se produce la necesidad de crear un software hasta que se finaliza y está listo para ser usado por un usuario. No en todos los programas ni en todas las ocasiones el proceso de desarrollo llevará fielmente las mismas etapas en el proceso de desarrollo; no obstante, son unas directrices muy recomendadas.

Hay más de un modelo de etapas de desarrollo, que de modo recurrente suelen ser compatibles y usadas entre sí, sin embargo vamos a estudiar uno de los modelos más extendidos y completos, el modelo en cascada.

Figura 1.3. Modelo en cascada

4.1. Análisis

La fase de análisis es la primera fase del proyecto, define los requisitos del software que hay que desarrollar.

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.

Inicialmente, esta etapa comienza con una entrevista al cliente, que establecerá lo que quiere o lo que cree que necesita, lo cual nos dará una buena idea global de lo que necesita, pero no necesariamente del todo acertada. Aunque el cliente crea que sabe lo que el software tiene que hacer, es necesaria una buena habilidad y experiencia para reconocer requisitos incompletos, ambiguos, contradictorios o incluso necesarios.

Es importante que en esta etapa del proceso de desarrollo se mantenga una comunicación bilateral, aunque es frecuente encontrarse con que el cliente pretenda que dicha comunicación sea unilateral, es necesario un contraste y un consenso por ambas partes para llegar a definir los requisitos verdaderos del software.

¿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.
>
> **Requisitos no 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), acompañado del diagrama de clases o de Entidad/Relación.

En este documento quedan especificados:

> La planificación de las reuniones que van a tener lugar.
>
> Relación de los objetivos del usuario cliente y del sistema.
>
> Relación de los requisitos funcionales y no funcionales del sistema.
>
> Relación de objetivos prioritarios y temporización.

Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.

Como ejemplo de requisitos funcionales, en la aplicación para nuestros clientes de las tiendas de cosmética, habría que considerar:

Si desean que la lectura de los productos se realice mediante códigos de barras.

Si van a detallar las facturas de compra y de qué manera la desean.

Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.

Si van a operar con tarjetas de crédito.

Si desean un control del stock en almacén.

Etc.

4.2. Diseño

En esta fase se pretende determinar el funcionamiento de una forma global del sistema, sin entrar en detalles. Uno de los objetivos principales es establecer las consideraciones de los recursos del sistema, tanto físicos como lógicos. Se define por tanto el entorno que requerirá el sistema, aunque también se puede establecer en sentido contrario, es decir, diseñar el sistema en función de los recursos de los que se dispone.

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es ¿Cómo hacerlo?

Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas.

Decidir qué hará exactamente cada parte.

En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.

En este punto, se deben tomar decisiones importantes, tales como:

Entidades y relaciones de las bases de datos.

Selección del lenguaje de programación que se va a utilizar

Selección del Sistema Gestor de Base de Datos.

Formato de la información de entrada y salida.

Etc.

En la fase de diseño se crearán los diagramas de casos de uso y de secuencia para definir la funcionalidad del sistema. Además con todos esos diagramas e información se obtendrá el cuaderno de carga.

4.3 Codificación

La fase más obvia en el proceso de desarrollo de software es sin duda la codificación. Es más que evidente que una vez definido el software que hay que crear haya que programarlo.

Gracias a las etapas anteriores, el programador contará con un análisis completo del sistema que hay que codificar y con una especificación de la estructura básica que se necesitará, por lo que en un principio solo habría que traducir el cuaderno de carga en el lenguaje deseado para culminar la etapa de codificación, pero esto no es siempre así, las dificultades son recurrentes mientras se modifica. Por supuesto que cuanto más exhaustivo haya sido el análisis y el diseño, la tarea será más sencilla, pero nunca está exento de necesitar un reanálisis o un rediseño al encontrar un problema al programar el software.

El programador tendrá que elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente, cumpliendo exhaustivamente con todos los requisitos 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.

4.4 Pruebas

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas. Las pruebas buscan confirmar que la codificación ha sido exitosa y el software no contiene errores, a la vez que se comprueba que el software hace lo que debe hacer, que no necesariamente es lo mismo.

No es un proceso estático, y es usual realizar pruebas después de otras etapas, como la documentación. Generalmente, las pruebas realizadas posteriormente a la documentación se realizan por personal inexperto en el ámbito de las pruebas de software, con el objetivo de corroborar que la documentación sea de calidad y satisfactoria para el buen uso de la aplicación.

En general, las pruebas las realiza, preferentemente personal diferente al que codificó la aplicación, con una amplia experiencia en programación, personas capaces de saber en qué condiciones un software puede fallar de

antemano sin un análisis previo.

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.

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.

4.5 Documentación

Por norma general, la documentación que se realiza de un software tiene dos caras: la documentación disponible para el usuario y la documentación destinada al propio equipo de desarrollo.

La documentación para el usuario debe mostrar una información completa y de calidad que ilustre mediante los recursos más adecuados cómo manejar la aplicación. Una buena documentación debería permitir a un usuario cualquiera comprender el propósito y el modo de uso de la aplicación sin información previa o adicional.

Por otro lado, tenemos la documentación técnica, destinada a equipos de desarrollo, que explica el funcionamiento interno del programa, haciendo especial hincapié en explicar la codificación del programa. Se pretende con ello permitir a un equipo de desarrollo cualquiera poder entender el programa y modificarlo si fuera necesario. En casos donde el software realizado sea un servicio que pueda interoperar con otras aplicaciones, la documentación técnica hace posible que los equipos de desarrollo puedan realizar correctamente el software que trabajará con nuestra aplicación.

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 del software:

Documentos a elaborar en el proceso de desarrollo de software			
	GUÍA TÉCNICA	**GUÍA DE USO**	**GUÍA DE INSTALACIÓN**
Quedan reflejados:	• El diseño de la aplicación. • La codificación de los programas. • Las pruebas realizadas.	• Descripción de la funcionalidad de la aplicación. • Forma de comenzar a ejecutar la aplicación. • Ejemplos de uso del programa. • Requerimientos software de la aplicación. • Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: • Puesta en marcha. • Explotación. • Seguridad del sistema.
¿A quién va dirigido?	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).
¿Cuál es su objetivo?	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.

4.6 Explotación

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.

Una vez que tenemos nuestro software, hay que prepararlo para su distribución. Para ello se implementa el software en el sistema elegido o se prepara para que se implemente por sí solo de manera automática.

Cabe destacar que en caso de que nuestro software sea una versión sustitutiva de un software anterior es recomendable valorar la convivencia de sendas aplicaciones durante un proceso de adaptación.

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. Se procede a 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 llevan 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.

4.7 Mantenimiento

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.

Son muy escasas las ocasiones en las que un software no vaya a necesitar de un mantenimiento continuado. En esta fase del desarrollo de un software se arreglan los fallos o errores que suceden cuando el programa ya ha sido implementado en un sistema y se realizan las ampliaciones necesitadas o

requeridas.

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).

5. Roles que interactúan en el desarrollo

A lo largo del proceso de desarrollo de un software deberemos realizar, como ya hemos visto anteriormente, diferentes y diversas tareas. Es por ello que el personal que interviene en el desarrollo de un software es tan diverso como las diferentes tareas que se van a realizar.

Los roles no son necesariamente rígidos y es habitual que participen en varias etapas del proceso de desarrollo.

- **Analista de sistemas**

– Uno de los roles más antiguos en el desarrollo del software. Su objetivo

consiste en realizar un estudio del sistema para dirigir el proyecto en una dirección que garantice las expectativas del cliente determinando el comportamiento del software.

– Participa en la etapa de análisis.

- **Diseñador de software**

– Nace como una evolución del analista y realiza, en función del análisis de un software, el diseño de la solución que hay que desarrollar.

– Participa en la etapa de diseño.

- **Analista programador**

– Comúnmente llamado “desarrollador”, domina una visión más amplia de la programación, aporta una visión general detallada del proyecto diseñando una solución más amigable para la codificación y participando activamente en ella.

– Participa en las etapas de diseño y codificación.

- **Programador**

– Se encarga de manera exclusiva de crear el resultado del estudio realizado por analistas y diseñadores. Escribe el código fuente del software.

– Participa en la etapa de codificación.

- **Arquitecto de software**

– Es la argamasa que cohesiona el proceso de desarrollo. Conoce e investiga los frameworks y tecnologías revisando que todo el procedimiento se lleva a cabo de la mejor forma y con los recursos más apropiados.

– Participa en las etapas de análisis, diseño, documentación y explotación.