Software

Ramon Costa i Pujol
Jeroni Vivó i Plans

PID_00153112

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

El software de un sistema informático está constituido por el conjunto de programas ejecutables en aquel sistema.

Los sistemas informáticos, es decir, la combinación del hardware, como son los ordenadores, la infraestructura de las redes y los sistemas de comunicación (cableado, etc.), con el software, han permitido mejorar muchos aspectos de la sociedad, ya que la aplicación de estos sistemas engloba todos los ámbitos, tanto genéricos como, por ejemplo, de:

- Negocios.
- Medicina.
- Educación.
- Ciencia y tecnología.
- Ingeniería y arquitectura.

O si se trata de ámbitos propios de las unidades funcionales o departamentales de las organizaciones, como, por ejemplo:

- Producción.
- Logística y distribución.
- Atención al cliente.
- Formación y gestión de los RRHH.
- Gestión de la planificación.

Todo este conjunto de software se desarrolla utilizando unos lenguajes de programación y siguiendo unas pautas y metodologías conocidas como *ingeniería del software*.

La construcción de programas de ordenador utilizando directamente el lenguaje nativo de la máquina, tal como se hacía en los inicios de la informática, tiene muchos inconvenientes. La única manera de superar estos inconve-

nientes consiste en desarrollar lenguajes adecuados para programar cualquier ordenador y que se puedan traducir al lenguaje que entienda una máquina concreta.

Uno de los apartados de este módulo consistirá en analizar qué caracteriza un lenguaje de programación.

Un lenguaje de programación

Un lenguaje de programación es un conjunto de normas sintácticas englobadas bajo un nombre (como puede ser C, Pascal, Java, etc.) que especifican cómo y cuándo se puede utilizar un conjunto de instrucciones determinado.

A partir de los lenguajes de programación se pueden diseñar programas, que después se convierten a códigos máquina por parte del compilador correspondiente del lenguaje para su ejecución, y ofrecer una visión general de cómo se produce la traducción de programas escritos en un lenguaje de alto nivel al lenguaje máquina de un ordenador.

No pretendemos, en esta asignatura, conocer con detalle los lenguajes de programación, sino ofrecer una visión abstracta de estos lenguajes que ayude a la comprensión de sus aspectos más interesantes (como los ámbitos de aplicación).

El diseño, el desarrollo y la implantación de un sistema informático engloba tanto los aspectos de software (diseño y desarrollo de aplicaciones, parametrización de programas existentes, etc.), como de hardware (instalación y configuración de ordenadores y redes, parametrización de servidores de correo, de ficheros, de red, etc.).

La gestión de todas estas tareas se realiza siguiendo los parámetros de la gestión de proyectos.

Una parte importante de todo sistema informático es la adquisición del software requerido para implantar la solución pedida (servidores de redes, de ficheros, programas de desarrollo, aplicaciones ya hechas, etc.).

Implantar, poner en marcha una instalación informática

También se utiliza esta palabra para referirse a la fase dentro del ciclo de vida de un programa durante la cual se codifica en un lenguaje de programación determinado.

Esta adquisición del programa se debe realizar atendiendo unas políticas de compra y uso de licencias marcadas por los fabricantes.

En definitiva, el objetivo de este módulo es introducir al estudiante en los conceptos principales:

- del software y sus aplicaciones,
- el desarrollo del software y los lenguajes de programación,
- la gestión de proyectos,
- el uso de las licencias de software.

Objetivos

Los **objetivos generales** que el estudiante puede alcanzar son los siguientes:

1. Conocer el concepto de software y sus principales aplicaciones.
2. Conocer las principales herramientas disponibles para el desarrollo de software, en especial por lo que respecta a lenguajes de programación y a gestión de proyectos.
3. Conocer las tipologías de licencias de software y sus implicaciones prácticas.

Estos objetivos generales se desglosan en los **objetivos específicos** siguientes:

1. Definir el concepto de software y enumerar sus principales características.
2. Enumerar las principales aplicaciones del software en función de su finalidad.
3. Describir el concepto de desarrollo de software para crear programas y aplicaciones.
4. Enumerar las principales características y diferencias entre los lenguajes máquina, ensamblador y de alto nivel.
5. Conocer un conjunto de los principales lenguajes de programación y sus diferencias.
6. Describir las principales características de una herramienta CASE.
7. Conocer los principales conceptos y elementos de la ingeniería del software.
8. Identificar los principales conceptos de la gestión de proyectos en general y enumerar las principales fases.
9. Describir los principales conceptos de la planificación de un proyecto y su seguimiento: tareas, recursos, herramientas y técnicas.
10. Identificar las principales características de un proyecto informático y enumerar sus principales fases y tareas.

11. Identificar y enumerar los principales papeles de los miembros de un equipo de trabajo en un proyecto informático.

12. Describir el concepto de licencias de software.

13. Listar diferentes posibilidades y opciones en la compra y uso de licencias de software.

14. Enumerar las principales características de las licencias "*Freeware*" y "*Shareware*".

1. Software: una visión general

El software de un sistema informático, más conocido como software, está constituido por el conjunto de programas ejecutables y ficheros necesarios por su funcionamiento.

Dentro del software, se incluyen herramientas tan variadas como:

- El sistema operativo.
- Las interfaces de usuario (sistema de ventanas, por ejemplo).
- Lenguajes de programación (Pascal o el entorno Visual Studio, por ejemplo).
- Herramientas o utilidades (compresores de ficheros, antivirus).
- Aplicaciones de cualquier especialidad y tipo (procesadores de texto, herramientas de diseño y gestión de imágenes).
- Ficheros de datos y bases de datos con los que trabajan estos programas; contienen los datos y las estructuras de datos necesarios para el funcionamiento del software y las generadas por su propio uso.

Una parte importante del software de un sistema lo forman las herramientas o utilidades. Dentro de este paquete de programas, se puede incluir software tan variado como programas de hoja de cálculo, tratamiento de textos como aplicaciones de uso común, los gestores de base de datos, los paquetes de todo en uno (herramientas conocidas como de productividad personal), los paquetes de gestión, así como otros paquetes de utilidades, ya mencionados anteriormente.

El **software** es todo aquel conjunto de programas o utilidades que se ejecutan en un ordenador.

Son muchas las empresas que se dedican a desarrollar este tipo de aplicaciones. Se trata de un mercado muy dinámico y con una competencia feroz, lo que provoca que la aparición de nuevas versiones y revisiones sea constante. Estas nuevas versiones quieren corregir errores antiguos y, a menudo, también presentan nuevas funcionalidades, muchas de ellas demandadas por los propios usuarios.

Los computadores tienen la capacidad de realizar muchas y distintas tareas, siempre y cuando tengan el software adecuado.

Los ordenadores permiten realizar trabajos que antes necesitaban un personal muy especializado en varios campos (delineación, analistas financieros, programadores) para poder llevarlos a cabo, facilitando la automatización y simplificación de las tareas de soporte de estos profesionales. Los software no pueden sustituir el conocimiento de los delineantes, analistas financieros o programadores; lo que pueden hacer es automatizar procesos rutinarios y dar elementos para facilitar el resto, de manera que el trabajo se haga de una forma más eficiente.

En la actualidad, la gran mayoría de estos trabajos se pueden realizar más fácilmente mediante un ordenador personal, el software adecuado y una mínima formación en el uso y el trabajo de este software (siempre que el usuario tenga un nivel de competencia adecuado en el ámbito temático de utilización).

Además, el software de inteligencia artificial permite ir más allá de la simple automatización a la hora de ayudar a resolver problemas complejos.

Una primera clasificación del software permitiría diferenciar dos grandes categorías: el software de sistema y el software de aplicación. Es decir, las aplicaciones y utilidades que sirven para controlar el hardware (sistemas operativos, servidores de bases de datos, servidores de correo electrónico, servidores de ficheros e impresoras, etc.) y los programas que trabajan en este software y hardware (aplicaciones empresariales, personales, etc.).

Una posible organización del software podría ser distinguir entre el software de sistema y el software de aplicación.

En esta unidad trataremos de las diferentes aplicaciones del software, proponiendo una clasificación posible de estos programas.

1.1. Aplicaciones del software

El software se puede aplicar a cualquier situación en la que se haya definido previamente un conjunto específico de pasos procedimentales, lo que se conoce como *algoritmos* (un problema determinado se intenta descomponer en un conjunto de problemas menores, lo cual permite llegar a crear algoritmos muy potentes).

Con el fin de determinar la naturaleza del software, hay que tener en cuenta tanto el contenido como la información gestionada.

Por una parte, el contenido tiene que ver con el significado y la forma de la información de entrada y de salida.

Aplicaciones bancarias

Por ejemplo, muchas aplicaciones bancarias utilizan unos datos de entrada muy estructurados (una base de datos) y producen "informes" con formatos determinados.

El software que controla una máquina automática (por ejemplo, un control numérico) actúa sobre elementos de datos discretos con una estructura limitada y produce órdenes concretas para la máquina en una rápida sucesión.

Por otra parte, en cambio, la gestión de la información también ha de tener en cuenta la predictibilidad de la orden y del tiempo de llegada de los datos.

Un programa de ingeniería acepta datos que están en un orden predefinido, ejecuta el algoritmo sin interrupción y produce datos resultantes en un informe o formato gráfico.

Programa de ingeniería

Ejemplos de este tipo de aplicaciones son:

- Consolidación de movimientos mensuales de los pedidos de una empresa.
- Gestión de las nóminas.

En cambio, un sistema operativo multiusuario recibe entradas que tienen un contenido variado o que se producen en instantes arbitrarios, ejecuta algoritmos que se pueden interrumpir por condiciones externas y produce una salida que depende de una función del entorno y del tiempo. Las aplicaciones con estas características se denominan indeterminadas.

Aplicaciones indeterminadas

Otros ejemplos de aplicaciones indeterminadas serían:

- Sistemas de monitoreo de plantas industriales.
- Aplicaciones de control de alarmas y seguridad en un edificio.

Con el fin de determinar la naturaleza del software, hay que tener en cuenta tanto el contenido, como la información que gestiona.

Es muy difícil establecer categorías genéricas para las aplicaciones del software. A medida que aumenta la complejidad del software, se hace más complicado establecer fronteras nítidamente separadas.

Con el fin de repasar las posibles aplicaciones del software, se ha decidido categorizarlas en los apartados siguientes:

- Software de sistemas.
- Software de tiempo real.
- Herramientas de uso personal.
- Software de ingeniería y científico.
- Herramientas de gestión y rediseño de procesos organizativos.

- Aplicaciones de inteligencia artificial.

Se habría podido clasificar el software según otras taxonomías o detallar más esta organización (gestión del conocimiento, control de la producción, sistemas de gestión de información gráfica, etc.) o por su aplicación en el ámbito de Internet y las relaciones entre empresas y usuarios (B2B[1], B2C[2], etc.), pero se ha optado por introducir, en este módulo, estos seis tipos de aplicaciones.

(1)B2B es el acrónimo de las palabras inglesas *business to business* ('negocio a negocio'). Esta abreviatura se utiliza mucho en el ciberespacio para referirse a todas las relaciones comerciales que establecen las empresas entre sí por medio de Internet.

(2)B2C es el acrónimo de las palabras inglesas *business to consumer* ('negocio a consumidor'). Esta abreviatura se utiliza mucho en el ciberespacio para referirse a las relaciones comerciales que establecen las empresas con los consumidores finales por Internet.

1.1.1. Software de sistemas

Son un conjunto de programas que se han escrito para servir a otros programas.

Algunos programas de sistemas como los compiladores o los editores y las utilidades de gestión de archivos procesan estructuras de información complejas pero determinadas.

Otras aplicaciones de sistemas son, por ejemplo, determinados componentes del sistema operativo, utilidades de gestión de los periféricos o procesadores, de telecomunicaciones que procesan datos muy indeterminados.

En cualquier caso, el área del software de sistemas se caracteriza por:

- Una fuerte interacción con el hardware de la computadora.
- Concurrencia de usuarios.

Estos dos elementos combinados hacen que este tipo de software tenga que prever el compartimento de recursos y haya de hacer, a la vez, una gestión de los procesos muy cuidadosa para dejar satisfechos a todos los usuarios, entendiendo como usuarios los programas que se ejecutan por encima del software de sistema.

1.1.2. Software de tiempo real

El software que mide, analiza y controla sucesos del mundo real a medida que ocurren se denomina *de tiempo real*.

Entre los elementos del software de tiempo real se incluyen:

- Un componente de adquisición de datos que recoge y da formato a la información recibida del entorno externo.
- Un componente de análisis que transforma la información según lo requiera la aplicación.
- Un componente de control/salida que responda al entorno externo.
- Un componente de monitoreo que coordina todos los demás componentes, de manera que se pueda mantener la respuesta en tiempo real (típicamente en el rango de un milisegundo a un minuto).

Cabe tener en cuenta que el concepto "tiempo real" significa que el sistema ha de responder dentro de unos vínculos estrictos de tiempo para evitar que se produzca algún desastre.

Ejemplos de software de tiempo real

Algunos ejemplos son los sistemas de monitoreo de una planta industrial que controlan todas las máquinas que intervienen en la cadena de producción, o el sistema de control de las alarmas de un edificio o de una planta nuclear o, más comunes para el usuario de a pie, el software de un cajero automático o el de un peaje.

1.1.3. Herramientas de uso personal

Desde la aparición de los ordenadores personales (PC) a finales de la década de los setenta e inicios de los ochenta se han desarrollado aplicaciones y programas pensados y diseñados para ayudar al trabajo personal y las tareas cotidianas, tanto en la oficina como en el hogar:

- Procesadores de textos.
- Hojas de cálculo.
- Gráficas por ordenador.
- Reproductores multimedia de vídeo y sonido.
- Juegos y aplicaciones de entretenimiento.
- Gestión de agendas personales.
- Aplicaciones financieras, de negocios y personal.

Éstas, y muchas más, son algunas de los centenares de aplicaciones diseñadas para ayudar a las personas en sus tareas cotidianas.

1.1.4. Software de ingeniería y científico

El software de ingeniería y científico está caracterizado por los algoritmos optimizados para la manipulación matemática de los datos.

Las aplicaciones de esta tipología van desde:

- Soporte a la observación astronómica.
- Predicción meteorológica.
- Monitorización de procesos industriales.
- Aplicaciones de soporte a la diagnosis médica.
- Aplicaciones de soporte al I+D.
- Etc.

O por aplicaciones más próximas a la gente de la calle como:

- El control y sincronización de semáforos.
- Cálculo de estructuras de edificios.
- Control del flujo de la corriente eléctrica.
- Etc.

En los últimos años, las aplicaciones de ingeniería/ciencia se han alejado de los algoritmos convencionales numéricos y el diseño asistido por ordenador (del inglés CAD), la simulación de sistemas u otras aplicaciones interactivas también tienen características del software de tiempo real e, incluso, del software de sistemas. En estos momentos muchas aplicaciones científicas permiten tratar, en tiempo real, medidas y datos como, por ejemplo, el seguimiento de la situación de los elementos meteorológicos.

1.1.5. Herramientas de gestión y rediseño de procesos organizativos

La gestión de información comercial y corporativa constituye la mayor de las áreas de aplicación del software.

Sistemas como la gestión de nóminas, la compatibilidad, la gestión de inventarios, etc. han evolucionado hacia el software de sistemas de información (SI), que accede a una o más bases de datos grandes que contienen información de la organización.

Las aplicaciones en esta área reestructuran los datos existentes para facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamiento de datos, las aplicaciones de software de gestión también realizan cálculos interactivos (por ejemplo, el procesamiento de transacciones en puntos de venta).

La evolución y la aplicación de este software ha ido dando apoyo, cada vez más, a otras funciones de la organización y sus departamentos, por lo que se han convertido en herramientas de gestión de la mayor parte de los procesos de las compañías:

- el proceso comercial,
- el proceso de compras,
- las operaciones de producción y fabricación,
- la gestión de los recursos humanos.

Para estas tareas han aparecido soluciones específicas como:

- SCM (*supply chain management*, 'gestión de la cadena de suministro'),
- CRM (*customer relationship management*, 'gestión de las relaciones con el cliente'),
- BPR (*business process reengineering*, 'reingeniería de procesos de negocio'),
- e-learning (formación por medio de herramientas telemáticas),
- e-rrhh (soluciones de apoyo a la gestión del departamento de recursos humanos),
- *Content management* ('gestión de contenidos y documentos'),
- *Knowlegde management* ('gestión del conocimiento').

1.1.6. Aplicaciones de inteligencia artificial

El software de inteligencia artificial (IA) utiliza algoritmos no numéricos para resolver problemas complejos para los cuales no es adecuado el cálculo o el análisis directo.

Diferentes áreas de la IA son los sistemas expertos, también denominados sistemas basados en el conocimiento: reconocimiento de patrones, reconocimiento de imágenes, de voz y de escritura.

Hay diferentes formas de implementar aplicaciones de inteligencia artificial; la que más se aproxima al funcionamiento del cerebro humano y de los animales son los algoritmos que implementan redes neuronales que simulan, hasta donde la tecnología y los conocimientos humanos permiten, la estructura de procesos del cerebro (las funciones de la neurona biológica) y, a la larga, pueden llevar a una clase de software que reconozca patrones complejos y a aprender "de experiencias" pasadas.

Las aplicaciones de inteligencia artificial se usan en ámbitos como la ingeniería, medicina, militar y la economía. También se han usado para juegos de estrategia, como el ajedrez, y para hacer videojuegos.

2. Desarrollo de software

Todo software debe haber sido diseñado y haber sido creado, o como se dice en términos informáticos, desarrollado.

Para desarrollar estos programas y aplicaciones, los "programadores" disponen de un conjunto de lenguajes de programación o desarrollo.

Existe una gran cantidad de lenguajes de desarrollo. Algunos se han diseñado para crear programas de un tipo específico o para un tipo de aplicaciones concretas (aplicaciones científicas y técnicas, inteligencia artificial, gestión, etc.) y otros son de aplicación "general".

Los programas desarrollados en estos lenguajes se deben traducir, posteriormente, al código fuente (un lenguaje inteligible por el ordenador y su procesador). Hay dos tipos de traductores, los compiladores y los intérpretes.

Con el fin de optimizar este proceso de desarrollo y minimizar, por otra parte, los posibles errores en la producción del software, se ha desarrollado lo que se denomina *ingeniería del software*, que engloba un conjunto de métodos, herramientas y procedimientos que facilitan el trabajo a los programadores.

Los objetivos de este apartado son:

- Introducir al alumno en los principales conceptos de los lenguajes de programación.
- Hacer un repaso de las "generaciones" de lenguajes.
- Enumerar los principales lenguajes de desarrollo existentes y su aplicación.
- Indicar las principales diferencias entre el lenguaje máquina, el lenguaje ensamblador y los lenguajes de alto nivel y sus características.
- Introducir las herramientas CASE.
- Enumerar los principales conceptos de la ingeniería del software.

2.1. Lenguajes de programación

Los desarrolladores de software, o "programadores", utilizan lenguajes de programación para crear y desarrollar los paquetes de software o programas como, por ejemplo, los procesadores de textos, las hojas de cálculo o los sistemas de mensajería o las aplicaciones web que los diferentes usuarios utilizan cuando trabajan con sus ordenadores.

Toda aplicación informática ha sido desarrollada utilizando un lenguaje de programación.

Se componen de los siguientes elementos:

- un léxico, que es el conjunto de símbolos permitidos o vocabulario,
- una sintaxis, que son las reglas que indican cómo realizar las instrucciones del lenguaje,
- y una semántica, que corresponde a las reglas que permiten determinar el significado de cualquier construcción del lenguaje.

Los programas que puede ejecutar el ordenador se han de "redactar" en el lenguaje nativo del procesador.

Es decir, cada instrucción debe estar en código binario y directamente relacionada con los circuitos del procesador.

Ahora bien, programar instrucciones completamente en código binario, es un proceso demasiado lento y sujeto a errores.

Por ello, los lenguajes de programación se han diseñado para permitir que el desarrollador escriba las instrucciones parecidas a un lenguaje humano, casi siempre en inglés.

Estas instrucciones se convierten, después, en el código binario correspondiente mediante unos programas denominados *compiladores*.

Tal como veremos más adelante, los compiladores traducen las instrucciones de un lenguaje de programación de alto nivel a código binario de bajo nivel, como una persona podría traducir de un idioma a otro, por ejemplo, del catalán al castellano.

De esta manera, un compilador o traductor convierte los programas que el desarrollador ha programado utilizando un lenguaje de alto nivel a un código fuente y un código objeto.

El código fuente es el conjunto de instrucciones escritas en un lenguaje de programación, mientras que el código objeto es el conjunto de instrucciones binarias que puede ejecutar el ordenador.

Los programas que el ordenador puede ejecutar han de estar "redactados" en el lenguaje nativo de sus procesadores. Por ello, los programas desarrollados utilizando los lenguajes de programación de alto nivel se han de traducir a un código fuente y un código objeto, inteligibles para el procesador del ordenador.

2.1.1. Lenguajes máquina, ensamblador y alto nivel

En este apartado, se profundiza en las principales características del lenguaje máquina, ensamblador y los de alto nivel.

Lenguaje máquina

Tal como se ha mencionado en la introducción, el lenguaje máquina es el único lenguaje que entiende directamente el ordenador, o mejor dicho, el procesador, ya que su estructura está totalmente adaptada a los circuitos de la máquina.

Por lo tanto, su forma de expresión está muy alejada de la comprensión y el análisis de las personas.

La programación con este tipo de lenguajes es muy complicada, ya que requiere un conocimiento amplio de la arquitectura física del ordenador por parte del programador.

Por contra, el código máquina permite que el desarrollador pueda utilizar la totalidad de los recursos que ofrece el ordenador, de manera que se pueden obtener programas muy eficientes, tanto en tiempo de ejecución como en ocupación de memoria.

Estos lenguajes constituyen la primera generación de lenguajes de programación.

Sus principales características son las siguientes:

- Las instrucciones se expresan en código binario, codificadas como cadenas de ceros (0) y unos (1), lo que hace que sea muy difícil de entender y modificar (mantener).

- Los datos se referencian por medio de las direcciones de memoria donde se encuentran y no por nombres de variables o constantes como pasa en los lenguajes de alto nivel.

- Cada una de las instrucciones realiza operaciones muy simples, de manera que el programador ha de saber cómo combinar el número reducido de instrucciones de las que dispone con el fin de realizar los algoritmos buscados.

- Las instrucciones tienen un formato muy rígido con respecto a la posición de los diferentes campos (por ejemplo, CÓDIGO OPERACIÓN - PRIMER OPERANDO - SEGUNDO OPERANDO), por lo que la redacción de las instrucciones es muy poco flexible.

- Tal como ya hemos mencionado, el lenguaje máquina o lenguaje nativo depende de la CPU (procesador) de cada ordenador y está ligado a ella, por lo que los programas en código máquina no se pueden transferir de un modelo de ordenador a otro y sólo pueden ejecutarse en el procesador con el cual está vinculado el código máquina correspondiente.

- En un programa escrito en lenguaje máquina no se pueden incluir comentarios que ayuden a su seguimiento y comprensión, y que faciliten la modificación posterior (mantenimiento).

El lenguaje máquina es lo único que entiende directamente el ordenador (su procesador). La programación con este tipo de lenguajes es muy complicada, pero permite que el desarrollador pueda utilizar la totalidad de los recursos que ofrece el ordenador, obteniendo programas muy eficientes, tanto en tiempo de ejecución como en ocupación de memoria.

Lenguaje ensamblador

La mayoría de las limitaciones comentadas de los lenguajes máquina las resuelve, parcialmente, el lenguaje ensamblador y casi totalmente los lenguajes de alto nivel o también conocidos como *simbólicos*.

Para evitar que los programadores deban programar directamente en código binario o máquina, se desarrollaron unos programas que permitieran traducir las instrucciones a código máquina.

Estos programas son la segunda generación de lenguajes de programación y se denominaron *ensambladores*.

Podían leer las instrucciones que las personas "podían entender" mejor, redactadas en lenguaje ensamblador, y las convertían en una instrucción de lenguaje máquina.

Aun así, este lenguaje también es de bajo nivel, ya que cada una de las instrucciones corresponde a una instrucción del lenguaje máquina.

Cada procesador tiene, por lo tanto, su lenguaje ensamblador, que traduce el código fuente, línea por línea, a código de máquina y crea el fichero ejecutable de la aplicación.

Código fuente

Listado de texto que contiene las instrucciones que componen una aplicación en un determinado lenguaje de programación que una vez compilado o parecido con la correspondiente herramienta da lugar a un programa funcional ejecutable bajo un sistema operativo.

Como principales características se puede enumerar que:

- Estos lenguajes utilizan una notación simbólica o mnemotécnica para representar el código de operación, lo que evita los código numéricos tan difíciles de utilizar. Estos códigos mnemotécnicos están constituidos por abreviaturas de las operaciones en inglés. Por ejemplo, la instrucción de sumar se representa en la mayoría de los lenguajes ensamblador con las letras ADD.

- En lugar de utilizar direcciones binarias absolutas en la memoria, los datos se pueden identificar por nombres, lo que se conoce como *dirección simbólica*.

- El programador puede utilizar comentarios entre las líneas de instrucciones de los programas, lo que los convierte en más inteligibles y simplifica su seguimiento y posterior modificación.

- El lenguaje ensamblador continúa siendo muy importante, ya que ofrece al programador el control total de la máquina y, por lo tanto, le permite generar un código compacto, rápido y eficiente.

A pesar de todo, el lenguaje ensamblador presenta la mayoría de los inconvenientes que también tiene el lenguaje máquina, como, por ejemplo:

- Un repertorio muy reducido de instrucciones.

- Un formato muy rígido para las instrucciones.

- La baja portabilidad entre procesadores y su fuerte dependencia del hardware.

Para solucionar de alguna manera la limitación que implica disponer de un repertorio de instrucciones tan reducido, se han desarrollado unos ensambladores especiales denominados *macroensambladores*.

Los lenguajes que traducen los macroensambladors tienen "macroinstrucciones", cuya traducción da lugar a varias instrucciones máquina y no sólo a una única.

Ejemplos tipos macroinstrucciones

Por ejemplo, hay macroinstrucciones para multiplicar, dividir, transferir bloques de memoria principal a disco, etc.

Debido a estas limitaciones los programador no suelen desarrollar los programas en lenguaje ensamblador, sino que lo hacen con lenguajes de más alto nivel.

Los lenguajes ensambladores se definieron para evitar que los programadores hubieran de programar directamente en código binario o máquina. Cada procesador tiene su propio lenguaje ensamblador que traduce el código fuente, línea por línea, a código máquina, lo que hace que también sean unos lenguajes de bajo nivel.

Lenguajes de alto nivel

La tercera generación de lenguajes de programación tenía como objetivo facilitar la tarea del desarrollador al permitir crear programas independientes de la máquina utilizando una sintaxis muy parecida al inglés.

Ejemplos de lenguajes

Ejemplos de estos lenguajes son el Cobol, el Basic, el Fortran, el C o el Pascal.

Con estos lenguajes los programadores pueden escribir en una sola instrucción el equivalente a varias instrucciones complicadas de bajo nivel.

Para saber más

Para más información sobre lenguajes de programación, podéis ver también el apartado "Principales lenguajes de desarrollo" de este mismo módulo.

Tienen unas instrucciones más fáciles de entender y proporcionan facilidades para expresar alteraciones del flujo de control de una manera bastante intuitiva.

Ejemplo de rutina de código

Por ejemplo, una rutina de código desarrollada en un lenguaje de alto nivel tendría la estructura siguiente:

```
A= 0

FOR i = 1 TO 10

A = A + (i * i)

NEXT i
```

En este caso, esta rutina calcula la suma de los primeros 10 cuadrados de los números naturales (1 * 1 + 2 * 2 + 3 * 3 + ... + 10 * 10)

Independientemente del lenguaje de alto nivel en el que se escriba el programa, lo debe traducir un compilador al lenguaje máquina para que lo pueda ejecutar el procesador correspondiente.

Sus características principales son las siguientes:

- Son independientes de la arquitectura física del ordenador, por lo cual se pueden utilizar los mismos programas en ordenadores de arquitecturas diferentes, sin necesidad de conocer el hardware específico de la máquina. Sólo es necesario "compilar"[3]el programa con el compilador correspondiente.

Para saber más

Para más información sobre los compiladores, podéis ver el apartado "compiladores" de este mismo módulo.

(3)"Compilar"

Acción de traducir código escrito en formato ASCII a lenguaje máquina (código máquina) mediante un compilador.

- Como ya hemos mencionado, la ejecución de un programa en un lenguaje de alto nivel requiere una traducción al lenguaje máquina del ordenador en el que se ejecutará. Una sentencia en un lenguaje de alto nivel corresponde, una vez traducida, a varias instrucciones en lenguaje máquina.

- Utilizan notaciones próximas a las utilizadas por las personas, con lo cual se pretende una aproximación mayor al lenguaje natural o algebraico, de manera que las instrucciones están expresadas en texto y es posible introducir comentarios en las líneas de código de los programas y su escritura está, normalmente, basada en reglas parecidas a las humanas.

- Ofrecen instrucciones potentes como, por ejemplo, funciones matemáticas (seno, coseno, conversión de enteros a reales, etc.), operadores específicos de entrada o salida como, por ejemplo, "PRINT" u operadores de tratamiento de cadenas de caracteres como, por ejemplo, "extraer un conjunto de caracteres de una línea de texto" o "buscar un subtexto en una cadena". También ofrecen la posibilidad de crear tus propias funciones.

- A diferencia de los lenguajes máquina y ensamblador, éstos son menos eficientes.

Todas estas características ponen de manifiesto un acercamiento a las personas y un alejamiento de las máquinas.

Por ello, todos los programas escritos en lenguaje de alto nivel no se pueden tratar directamente por el ordenador y es necesario traducir al lenguaje máquina para ser ejecutado.

Estos programas que realizan estas traducciones se denominan traductores y han sido desarrollados para cada ordenador específico.

Una evolución de estos lenguajes fueron los enmarcados dentro de la cuarta generación (4GL), que permiten generar de manera automática la mayor parte de los procedimientos (rutinas o conjuntos de instrucciones de código en lenguaje de alto nivel) de un programa. De esta manera, el desarrollador sólo debe indicar qué quiere hacer, pero no cómo.

Un programador que trabaje con un lenguaje de tercer nivel como, por ejemplo, el C, escribe instrucciones de lo que se ha de hacer e indica cómo hacerlo por medio de los algoritmos redactados.

En cambio, con los lenguajes 4GL (que permiten generar código sin tener que desarrollarlo, a partir de especificaciones), los desarrolladores o usuarios (analistas funcionales, expertos no técnicos) pueden escribir programas de manera sencilla, por ejemplo, para consultar una base de datos o para crear sistemas de información personal o departamentales.

Muchos de estos lenguajes disponen de una interfaz gráfica.

Estos lenguajes convierten las especificaciones indicadas por el desarrollador o bien en lenguajes de tercer nivel, que el mismo programador puede refinar, o bien generan las instrucciones en código máquina directamente.

Ejemplos de lenguajes 4GL son la mayoría de las herramientas CASE de diseño de bases de datos, modelización de procesos, etc.

Finalmente, hay que indicar que la quinta generación de los lenguajes de programación son los denominados *lenguajes naturales*, que si bien están en sus inicios, facilitan una aproximación total al lenguaje humano.

Con los lenguajes de alto nivel, los programadores pueden escribir en una sola instrucción el equivalente a varias instrucciones complicadas de bajo nivel. Ahora bien, estos programas han de ser, posteriormente, traducidos a un lenguaje inteligible por el procesador.

2.1.2. Traductores

Tal como ya se ha comentado anteriormente, dado que un ordenador sólo puede interpretar y ejecutar el código máquina, existen unos programas especiales, denominados *traductores*, que traducen programas escritos en un lenguaje de programación al lenguaje máquina del ordenador.

Un traductor es un metaprograma que tiene como entrada un programa (o parte de un programa) escrito en lenguaje simbólico, alejado de la máquina, denominado *programa fuente* (*código fuente*) y proporciona como salida otro programa semánticamente equivalente, escrito en un lenguaje comprensible por el hardware del ordenador denominado *programa objeto* (*código objeto*).

En este material se tratarán dos tipos de traductores, los compiladores y los intérpretes, que representan dos aproximaciones muy diferentes a la tarea de permitir el funcionamiento de los programas escritos en un determinado lenguaje de programación de alto nivel.

Compiladores

Un compilador traduce completamente un programa fuente y genera un programa objeto (semánticamente equivalente) escrito en lenguaje máquina.

Como parte importante de este proceso de traducción, el compilador informa al desarrollador de la presencia de errores en el programa fuente y pasa a la creación del programa objeto (en lenguaje máquina) sólo en caso de que no se hayan detectado errores (generalmente, se suele cancelar la compilación cuando se detectan errores).

El programa fuente suele estar contenido en un fichero y el programa objeto se puede almacenar como otro fichero en memoria masiva (disco duro, CD, etc.) para ser ejecutado posteriormente sin que sea necesario volver a traducirlo.

Para saber más

Podéis ver los dispositivos externos de memoria masiva en el apartado 1.4 del módulo "Ordenadores y sistemas operativos".

Las principales ventajas de los compiladores frente a los intérpretes es que sólo se ha de efectuar la traducción una vez. Una vez traducido un programa, su ejecución es independiente de su compilación. Así, los compiladores permiten realizar optimizaciones de código al leer y traducir todo el texto.

Intérpretes

Un programa intérprete permite que un programa fuente escrito en un lenguaje determinado se vaya traduciendo y ejecutando directamente, sentencia a sentencia, por el ordenador.

El intérprete revisa una sentencia fuente, la analiza, la interpreta y da lugar a su ejecución inmediata.

A diferencia del proceso de "compilación", en este caso no se crea ningún fichero o programa objeto almacenable en memoria masiva para posibles ejecuciones futuras.

Si se utiliza un intérprete para traducir un programa, cada vez que se ejecuta este último se vuelve a analizar (ya que no se genera un fichero objeto).

En cambio, con un compilador, aunque resulte más lenta, sólo se ha de realizar la traducción una vez.

Además, los traductores no permiten realizar optimizaciones del código (que elimina órdenes innecesarias compactando el código) más allá del contexto de cada sentencia del programa.

La principal ventaja de los intérpretes sobre los compiladores es que resulta más fácil localizar y corregir errores de los programas (depuración de programas), ya que la ejecución de un programa bajo un intérprete se puede interrumpir en cualquier momento para conocer los valores de las diferentes variables y la instrucción fuente que se acaba de ejecutar en aquel momento.

Por este motivo, los intérpretes resultan más pedagógicos para aprender a programar, ya que el alumno puede detectar y corregir más fácilmente sus errores. Son famosos los intérpretes del lenguaje de programación Basic, un lenguaje muy utilizado para enseñar a programar.

2.1.3. Principales lenguajes de desarrollo

En este apartado se enumeran, brevemente, algunos de los principales lenguajes de programación.

El objetivo es tener una visión de diferentes tipos de lenguajes de desarrollo y su principal aplicación.

Se trata de conocer los principales conceptos de su utilidad y principal funcionalidad, pero no de saber la sintaxis y la manera de trabajar.

Dado que los lenguajes de alto nivel son los más utilizados, tal como se ha comentado en los apartados anteriores, la clasificación y presentación se centrará en estos lenguajes.

Los lenguajes de alto nivel se pueden clasificar en dos divisiones, los lenguajes de propósito general, que se pueden utilizar en cualquier tipo de aplicaciones (como, por ejemplo, el lenguaje C o el Pascal) y los lenguajes de propósito especial.

Una primera clasificación de los lenguajes de desarrollo se puede establecer entre los lenguaje de propósito general y los de propósito especial.

Aun así, al ser tan general esta clasificación, también se podrían intentar subclasificar desde el punto de vista del campo de aplicación al que pertenece el lenguaje:

- **Aplicaciones científicas**: predominan las operaciones numéricas o matriciales propias de algoritmos matemáticos. Un lenguaje adecuado para este tipo de aplicaciones es el Fortran, por ejemplo.

Para saber más al respecto

Para más información sobre lenguaje de base de datos SQL podéis consultar el módulo 2, "Bases de datos".

Fortran

El Fortran se diseñó para ser utilizado en aplicaciones científicas y técnicas (matemáticas, ciencia e ingeniería). Se caracteriza por la potencia en sus cálculos matemáticos, pero está limitado en todo lo que hace referencia al tratamiento de datos no numéricos, por lo que no resulta adecuado para aplicaciones de gestión, gestión de ficheros y el tratamiento de caracteres y edición de informes.

A pesar de haberse creado en 1953, de la mano de John Backups, un empleado de IBM, sigue siendo un lenguaje común en aplicaciones de investigación, ingeniería y educación.

Fortran significa ***for**mula **tra**nslator* ('traductor de fórmulas') y se considera el primer lenguaje de alto nivel; la primera versión apareció en 1957.

- **Aplicaciones de procesamiento de datos**: en estas aplicaciones son frecuentes las operaciones de creación, mantenimiento y consulta sobre ficheros y bases de datos. Dentro de este campo, se encontrarían las aplicaciones de gestión empresarial, como los programas de nóminas, contabilidad, facturación, control de inventario, etc. Algunos de los lenguajes aptos para este tipo de aplicaciones son el Cobol y el lenguaje de bases de datos SQL.

Cobol

El Cobol es el lenguaje más utilizado en aplicaciones de gestión. Lo creó en 1960 un comité patrocinado por el Departamento de Defensa de Estados Unidos con la finalidad de disponer de un lenguaje universal para aplicaciones comerciales, y hoy día existen millones de líneas de código escritas en este lenguaje.

El nombre Cobol proviene de la frase *common business oriented language* ('lenguaje general para los negocios') y a lo largo de su existencia ha sufrido distintas actualizaciones.

Las características más interesantes de este lenguaje son que se parece al lenguaje natural (se utiliza mucho el inglés sencillo), es autodocumentado y ofrece grandes facilidades en la gestión de ficheros y también en la edición de informes escritos.

En cambio, sus rígidas reglas de formato de escritura, la necesidad de escribir todos los elementos al máximo detalle, la extensión excesiva de sus sentencias (a veces resulta demasiado enrevesado y reiterativo) y la inexistencia de funciones matemáticas son sus inconvenientes principales.

- **Aplicaciones de tratamiento de textos**: estas aplicaciones se encuentran asociadas al manejo de textos en lenguaje natural. El lenguaje C sería adecuado para este tipo de aplicaciones.

Lenguaje C

El lenguaje C lo creó en 1972 Dennis Ritchie (que, junto con Ken Thompson, había diseñado anteriormente el sistema operativo Unix), con la intención de conseguir un lenguaje idóneo para la programación de sistemas que fueran independientes de la máquina para utilizarlos en la implementación del sistema operativo Unix.

Desde entonces, tanto Unix como C han tenido un enorme desarrollo y proliferación, hasta convertirse en un estándar industrial para el desarrollo del software.

El C es un lenguaje moderno de propósito general que combina las características de un lenguaje de alto nivel (programación estructurada, tipo y estructuras de datos, recursividad, etc.) con una serie de características más propias de lenguajes de bajo nivel.

Esta cualidad del C posibilita que el programador utilice la programación estructurada para resolver tareas de bajo nivel, obteniendo un código ejecutable veloz y eficiente.

Por ello, mucha gente lo considera como un lenguaje de nivel medio.

- **Aplicaciones en inteligencia artificial**: destacan las aplicaciones en sistemas expertos, juegos, visión artificial y robótica; los lenguajes más populares dentro de este ámbito son el Lisp y el Prolog.

Lisp

El Lisp fue diseñado en 1959 por John McCarthy en el MIT (Instituto Tecnológico de Massachusetts) para utilizarlo en el ámbito de la inteligencia artificial. Este lenguaje toma su nombre del inglés ***list p****rocessing* ('procesamiento de listas').

Lisp está pensado para resolver problemas de manipulación de símbolos (que son de gran interés en inteligencia artificial). Los símbolos son elementos básicos de este lenguaje y representan objetos arbitrarios del dominio de interés que se esté tratando.

Resulta un lenguaje funcional, ya que todo programa (o subprograma) en Lisp se puede ver como una función de alto nivel que se aplica sobre otras funciones de más bajo nivel para obtener determinados resultados. Para realizar operaciones elementales se pueden utilizar funciones de una biblioteca.

Este lenguaje no se parece nada a otros lenguajes de programación. Aun así, resulta un lenguaje fácil de entender y es el más común dentro de las aplicaciones en inteligencia artificial.

Uno de sus principales problemas era que no se podía ejecutar de manera eficiente en muchos ordenadores. En la actualidad, existen versiones estándar de Lisp, Common Lisp, DG Common Lisp y Lisp portable estándar.

Prolog

El Prolog (***pro****gramming* ***logi****c*) se desarrolló a partir del trabajo realizado en la década de los años setenta, principalmente en universidades europeas, y ha sido el lenguaje más utilizado en el ámbito de la inteligencia artificial en Europa.

Se basa en la lógica y ha sido utilizado para desarrollar un gran número de aplicaciones en bases de datos e inteligencia artificial.

El Prolog permite que el programador exprese una serie de tareas basadas en la descripción de los objetos que intervienen (hechos y reglas) y las relaciones lógicas que existen entre ellos (predicados), en lugar de hacerlo mediante un algoritmo.

Lleva incorporada la programación de operaciones y todo el esfuerzo de programación consiste en especificar adecuadamente los hechos y las reglas para después establecer preguntas que se podrán inferir de manera automática.

El Prolog permite desarrollar sistemas expertos a personas sin mucha idea de programación, ya que no requiere programar ningún algoritmo.

Vista esta facilidad de uso, una importante aplicación del Prolog es la educación, donde se puede utilizar para enseñar lógica, técnicas de resolución de problemas y se suele emplear en un gran número de bases de datos educativas.

- **Aplicaciones de programación de sistemas**: en este campo se incluirían la programación de software de interfaz entre el usuario y el hardware, como son los módulos de un sistema operativo y los traductores. Tradicionalmente, tal como se ha mencionado con anterioridad, para estas aplicaciones se utilizaba el lenguaje ensamblador. En la actualidad se muestran muy adecuados los lenguajes como el Ada, el Modula-2 y el C, por ejemplo.

Ada

El lenguaje Ada nació con el objetivo de obtener un único lenguaje para todo tipo de aplicaciones (un auténtico lenguaje de propósito general).

Lo encargó el Departamento de Defensa de Estados Unidos y su estandarización fue publicada en 1983. El nombre de Ada proviene de Augusta Ada Byron, condesa de Lovelace, considerada la primera programadora de la historia.

Entre las características del lenguaje se incluye la compilación separada, la programación concurrente, la programación estructurada, la buena mantenibilidad, características de tiempo real, etc.

Sin embargo, el principal inconveniente de este lenguaje es su gran extensión, que puede complicar su uso.

Modula-2

El mismo Nicklaus Wirth, creador del lenguaje Pascal, dirigió, a finales de los años setenta, el desarrollo del lenguaje Modula-2 (denominado en un principio simplemente Modula), con la intención de incluir las necesidades de la programación de sistemas y responder a las críticas recibidas con respecto a las carencias del lenguaje Pascal.

Además de las principales características del Pascal, este lenguaje incorpora funcionalidades como la posibilidad de compilación separada, la creación de bibliotecas, la programación concurrente, la mejor gestión de cadenas de caracteres, los procedimientos de entrada/salida y de gestión de la memoria y, además, aporta grandes facilidades para la programación de sistemas.

Este lenguaje también posee cualidades didácticas, por lo que ha sido sobradamente aceptado dentro de la comunidad universitaria como herramienta idónea para enseñar la programación.

Otra clasificación según el tipo de aplicación del lenguaje permitiría clasificarlos en aplicaciones científicas, de procesamiento de datos, de tratamiento de textos, de inteligencia artificial y de programación de sistemas.

Otra manera de clasificar los lenguajes de alto nivel puede ser según el estilo de programación que fomentan, es decir, la filosofía de construcción de programas:

- Lenguajes imperativos o procedimentales: establecen cómo se debe ejecutar una tarea, dividiéndola en partes que especifican cómo realizar cada una de las subtareas asociadas. Estos lenguajes se fundamentan en el uso de variables para almacenar valores y el uso de instrucciones que indican las operaciones que debe realizar sobre los datos almacenados. La mayoría de los lenguajes de alto nivel son de este tipo: Fortran, Basic, Pascal, Ada, Modula-2, Algol, RPG, PL/1, Simula, Smalltalk y Eiffel.

Basic

El lenguaje Basic se diseñó en 1965 con el objetivo de que lo utilizaran los principiantes en su introducción al mundo de la programación.

Resulta un lenguaje fácil de aprender, como indica su nombre: ***b**eginner's **a**ll-purpose **s**ymbolic **i**nstruction **c**ode* ('código simbólico de propósito general para principiantes'); al principio se enfocó a enseñar a estudiantes a los que se pretendía introducir en el mundo de la programación y se convirtió en el lenguaje educativo más popular del mundo.

Las principales aportaciones de Basic son que es un lenguaje interpretado (traducido por un programa intérprete) y que es de uso interactivo.

Con los años, este lenguaje ha evolucionado hacia versiones preparadas para desarrollar aplicaciones por Internet, de gestión y soportar características y métodos orientados a objetos, como el Visual Basic.

Pascal

El lenguaje Pascal lo desarrolló en 1970 el matemático suizo Nicklaus Wirth con el objetivo de proporcionar un lenguaje adecuado para la enseñanza de los conceptos y las técnicas de programación, y permitió el desarrollo de aplicaciones fiables y eficientes en los ordenadores disponibles en aquel momento.

El lenguaje recibe su nombre en honor del filósofo y matemático francés Blaise Pascal, que inventó la primera máquina de tipo mecánico para sumar y, con el tiempo, ha llegado a ser un lenguaje muy utilizado en todo tipo de aplicaciones, y se emplea mucho en las universidades para aplicaciones científicas y de ingeniería.

Pascal se ha diseñado para ilustrar conceptos clave en programación, como los de tipo de datos, programación estructurada (es un lenguaje muy estructurado) y diseño descendente. El Pascal trata de proporcionar un mecanismo para implementar cada uno de los conceptos de programación.

Este lenguaje se ha convertido en el predecesor de otros lenguajes más modernos, como el Modula-2 y el Ada.

Algol

El Algol (***algo****rithmic* ***l****anguage* o ***alg****ebraic* o ***o****rented* ***l****anguage*) es un lenguaje algorítmico u orientado al álgebra, importante por haber introducido por primera vez conceptos que se han revelado básicos en la programación, principalmente la idea de estructuración. Se llegó a utilizar como un lenguaje internacionalmente adoptado para la descripción de algoritmos matemáticos. Se desarrolló en la década de los sesenta.

RPG

El RPG (*report program generator*) es en realidad un generador de programas que lee unas especificaciones y adapta los módulos de programa para la realización de las entradas y salidas indicadas en las instrucciones.

Apareció con los pequeños ordenadores para gestión como el Sistema/3 de IBM a finales de los años sesenta. Modificaciones posteriores (las tarjetas de instrucciones tipos C y las versiones RPG III y RPG 400) lo configuran con la posibilidad de desarrollar algoritmos específicos, mediante el uso de variables binarias conocidas como *interruptores*, y también de gestionar bases de datos.

PL/1

El PL/1 (*program language 1*) es un lenguaje diseñado para que se utilice tanto en los problemas de tipo científico, como en los de gestión. Su diseño se basa mucho en otros lenguajes como el Fortran, el Cobol y el Algol.

Del Algol adopta la estructura de bloques y las instrucciones de control de secuencias estructuradas; del Cobol, la riqueza en la gestión de ficheros y la declaración de datos de tipo PICTURE, y del Fortran, la forma concisa y simple de las instrucciones y el mecanismo de transmisión de parámetros, entre otros elementos.

Simula

Es el antecesor de los lenguajes orientados a objetos. Construido en torno a ideas ya presentes en el Algol, en 1962, el Simula incluye, además, los conceptos de encapsulación y herencia, y fue pionero a la hora de utilizar los conceptos de clase, objetos, mensajes y tipos abstractos de datos, tan corrientes, tiempo después, en todos los lenguajes orientados a objetos.

Uno de los aspectos más importantes de Simula, concebido como un lenguaje que describe sistemas y elabora simulaciones de estos sistemas por ordenador, es que introdujo la idea de construir simulaciones de sistemas complejos procurando reflejar el vocabulario del sistema real y del dominio del problema.

Smalltalk

Se puede considerar el primer lenguaje orientado a objetos realmente "puro", en el sentido de que es el primero que se plantea como un lenguaje de nuevo acuñamiento para adoptar íntegramente la nueva orientación a objetos. En el Smalltalk todo se trata como un objeto, desde los números enteros hasta las clases.

Concebido como un intérprete, se adelantó a su tiempo en la utilización de una interfaz gráfica y el uso interactivo. Su influencia ha resultado decisiva en el resto de lenguajes orientados a objetos.

Eiffel

Eiffel se publicó en 1986 y se presentó como un lenguaje orientado a objetos que se traduce primero a otro lenguaje de programación, lo cual, como en el caso de C++, favorece la portabilidad.

El lenguaje Eiffel se considera, en el mundo académico, como el lenguaje orientado a objetos mejor diseñado y completo, aunque no siempre resulte el más eficiente.

En realidad, es un entorno de desarrollo con herramientas auxiliares, como sistemas gráficos de exploración de estructuras, depuradores, apoyo de persistencia, gestión de configuraciones, etc.

- Lenguajes declarativos: en este caso, el proceso por el cual se ejecuta el programa no aparece de manera explícita en el programa. El programador no necesita indicar el proceso detallado de cómo realizar la tarea. En estos lenguajes, los programas se construyen mediante descripciones de funciones (lenguajes funcionales, como Lisp) o expresiones lógicas que indican las relaciones entre determinadas estructuras de datos (lenguajes de programación lógica, como Prolog).

- Lenguajes orientados a objetos: el diseño de los programas se centra más en los datos y su estructura. Los programas consisten en descripciones de unidades denominadas objetos que encapsulan los datos y las operaciones que actúan sobre éstos. El lenguaje más utilizado dentro de este tipo es el C++ .

- Lenguajes orientados al problema: este tipo de lenguajes están diseñados para problemas específicos, principalmente de gestión. En estos lenguajes, los programas están formados por sentencias que ordenan qué se quiere realizar. Generalmente, estos lenguajes suelen ser generadores de aplicaciones que permiten automatizar tanto como pueden la tarea de desarrollo del software de aplicación de gestión.

Finalmente, si se tiene en cuenta la manera de programar, utilizando estos lenguajes se pueden categorizar entre los procedimentales, los declarativos, los orientados a objetos y los orientados al problema.

Java

Es un lenguaje de programación orientado a objetos creado por Sun Microsystems, a principios de los años noventa. Es similar al lenguaje C++ pero no tiene elementos de bajo nivel, por lo que resulta conceptualmente menos abstracto.

Es un lenguaje interpretado; el lenguaje de alto nivel es compilado en bytcodes, los cuales son transversales a cualquier arquitectura; por lo tanto, el programa será exportable a todas las arquitecturas de microprocesadores. Para que funcionen, sólo hace falta que la plataforma tenga una máquina virtual Java, es decir, el intérprete de los bytecodes adecuado a cada plataforma.

Javascript

Es un lenguaje orientado a objetos de forma amplia utilizado para dotar de dinamismo a las paginas webs escritas en código HTML (*hiper text marked language*). De sintaxis muy similar a Java, pero de complejidad mucho menor.

Fue inventado por Brendan Ecih mientras formaba parte de Netscape Comunications. Actualmente, todos los navegadores web soportan este lenguaje inevitablemente, ya que un porcentaje elevadísimo de las páginas webs lo usan.

Lenguajes propietarios

Algunas grandes aplicaciones se pueden personalizar (customizar) según la empresa que las compre para adaptarlas a su negocio y a su particular forma de operar, a la hora de adaptar la aplicación al negocio y después personalizarla; según sus flujos operativos particulares, hace falta algún lenguaje que haga este paso. Estos lenguajes no sirven para crear aplicaciones desde cero, sino para personalizar y adaptar aplicaciones de carácter genérico como SAP (ERP) o Siebel (CRM).

ERP

Enterprise resource planning (planificación de los recursos de la empresa).

CRM

Customer relationship mangement (gestión de la relación con el cliente).

2.2. Herramientas CASE

El término CASE (*computer aided software engineering*) se popularizó en la década de los ochenta para designar una tecnología que no era nueva: las ayudas automatizadas de las metodologías de desarrollo de software, y representaba la evolución de las primeras ayudas (editores, compiladores, etc.) que formaron los entornos de desarrollo de software de los años setenta.

2.2.1. Principales conceptos e historia

CASE, la ingeniería de software asistida por ordenador, incluye el conjunto de herramientas y métodos asociados que pueden servir de ayuda en el proceso de construcción del software a lo largo de su ciclo de vida.

La novedad del CASE se hizo evidente cuando se incorporaron a las herramientas de ayuda al diseño informático nuevas funcionalidades gráficas con las cuales se pudo representar, modificar y mantener actualizados los diagramas gráficos que se han convertido en clásicos y se utilizan en la mayoría de las metodologías de desarrollo de software.

Una herramienta CASE permite ayudar al desarrollador y al equipo de proyectos a construir el programa dándoles herramientas para que puedan generar documentación, código, diagramas, etc.

Las verdaderas herramientas CASE surgieron cuando los diagramas se incorporaron a una base de datos global de diseño (diccionario de datos o repositorio), que también mantenía detalles de los elementos de datos y de la lógica de los procesos.

Este diccionario de datos es, en realidad, una base de datos que incorpora un modelo en el que se incluyen restricciones de integridad de los datos.

Ejemplos de herramientas CASE son los sistemas Excelerator, System Architect, EasyCASE, IEF (*information enineering facility*), IEW (*information engineering workbench*), Teamwork, VAW (*visible analyst workbench*), así como todos los aparecidos más adelante que aprovechan al máximo todas las prestaciones de la interfaz gráfica.

Dos herramientas o entornos muy conocidos hoy día de herramientas CASE son el programa Microsoft Visio y el paquete Rational Rose, que incorpora aplicaciones para generar documentación, código, probar los programas, etc.

Si bien existen diferentes subdivisiones, una clasificación de las herramientas CASE se basa en CASE alto (*upper CASE*), medio (*middle CASE*) y bajo (*lower CASE*) de acuerdo con el nivel de abstracción y generalidad.

Otra distinción posible se establece entre herramientas (*toolkits*), supuestamente aplicables a una única tarea del desarrollo del software, y talleres de trabajo (*workbench*), que se presentan como una colección de herramientas integradas que ayudan a todo el proceso de desarrollo de software: análisis, diseño e implementación.

Existe gran infinidad de herramientas CASE en el mercado y se clasifican por CASE alto, medio y bajo o también por herramientas y talleres de trabajo.

2.2.2. Aplicación del CASE y Herramientas

La mayoría de las herramientas CASE se aplican a las fases de análisis y diseño, pero algunos sistemas más completos incorporan generadores de lenguajes e incluyen la posibilidad de generar el código fuente de la aplicación de una manera automática.

De esta manera, ayudan tanto al proceso de construcción del software, como también a la elaboración y al mantenimiento de la documentación, con lo que simplifican buena parte del trabajo de programar.

A veces, las herramientas CASE son simplemente un generador que utiliza las especificaciones procedentes de otra herramienta CASE, como puede suceder con la familia de generadores del APS Development Center, que puede generar el Cobol a partir, por ejemplo, de las especificaciones que le proporciona Excelerator.

Con respecto al análisis y diseño de sistemas, predominan las herramientas que implementan los diagramas de flujo de datos (DFD, *data flow diagrams*) del análisis y diseño estructurado de Constantine, Yourdon y De Marco y también en la variante de Gane/Searson.

En cuanto al diseño de datos, se utiliza a menudo el diagrama de entidad-relación de Chen (ERS, *entity- relationship diagrams*) o la versión más clásica de los diagramas de Bachman para convertirlos finalmente en su equivalente en el modelo de bases de datos relacionales.

También, para nuevas metodologías como MERISE, se implementan los diagramas de la historia de vida de una entidad (ELH, *entity-life-history-diagrams*).

Otras técnicas de diseño muy populares y utilizadas por las herramientas CASE son los diagramas de descomposición y jerarquía de funciones (HD, *hierarhical diagram*), los diagramas de estructura de módulos de la programación modular y el diseño estructurado (SC, *structured chart*), los diagramas de estructura de datos de la metodología Jackson (DSD, *data structure diagrams*) o los grafos de diseño de diálogos (IDS, *invocation dialog structure*) y los diseños de pantallas y listados entre otras modelizaciones gráficas muy utilizadas en la ingeniería del software.

Objetivo del apartado

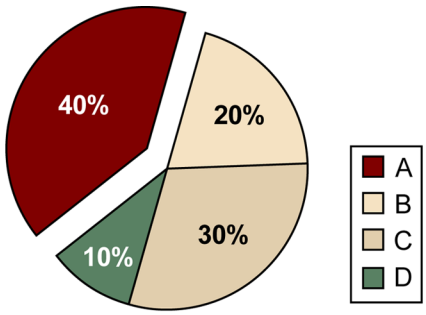
El objetivo de este apartado no es profundizar en técnicas y herramientas de diseño y desarrollo de aplicaciones. Aun así, se enumeran todas estas técnicas para que resulten familiares al estudiante.

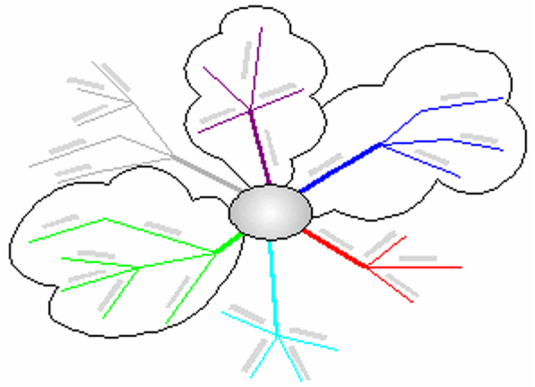
Las herramientas CASE pueden tener diferentes aplicaciones. Desde el análisis y diseño hasta la generación del programa y su documentación.

Algunos de los modelos que permiten generar estas herramientas son los siguientes:

- Diagramas genéricos. Utilidades para crear diseños y esquemas para planificación, comunicación y generación de documentación de apoyo.

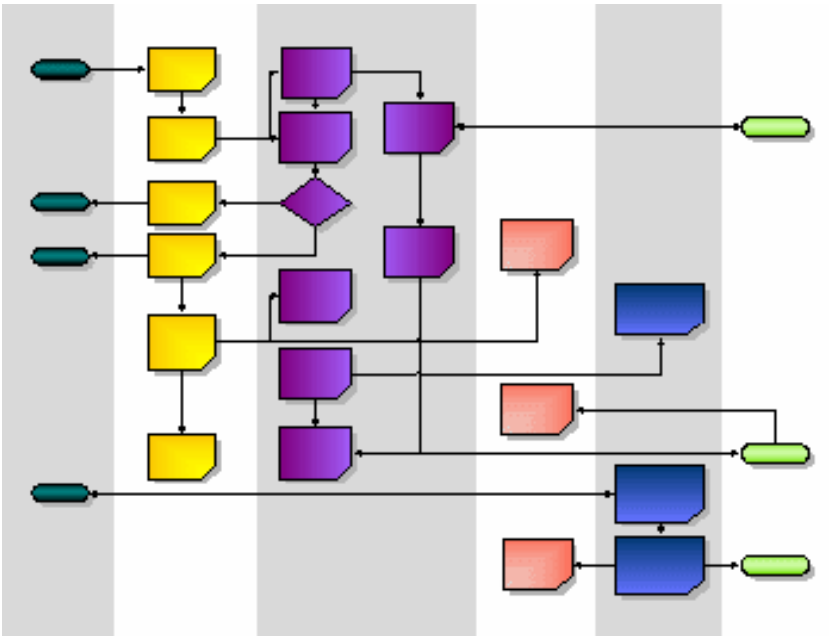
Diagramas genéricos





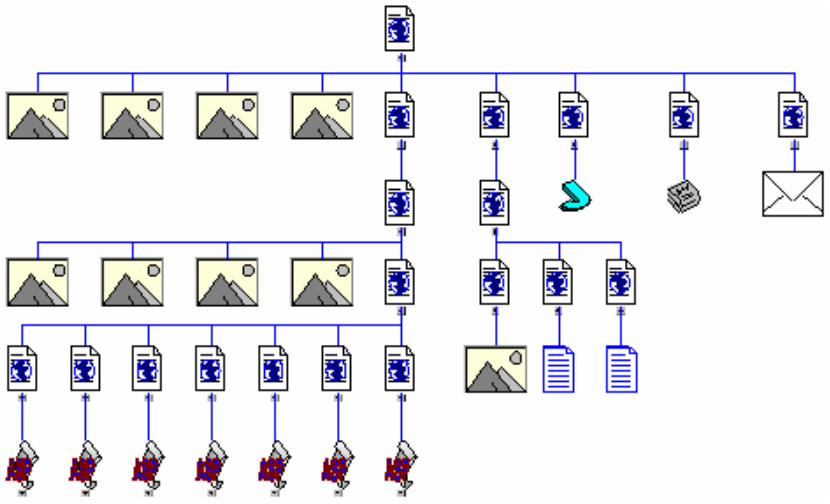
- Diagramas de auditoría. Utilizados para documentar y analizar procesos.

Diagramas de auditoría



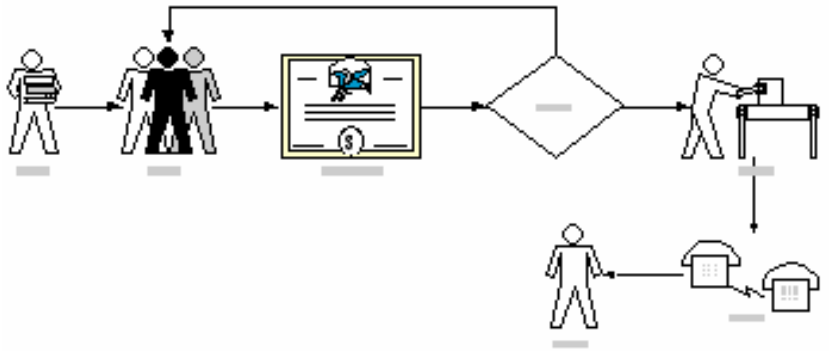
- Diagramas "conceptuales de sitios web". Utilizados en el diseño y desarrollo de webs.

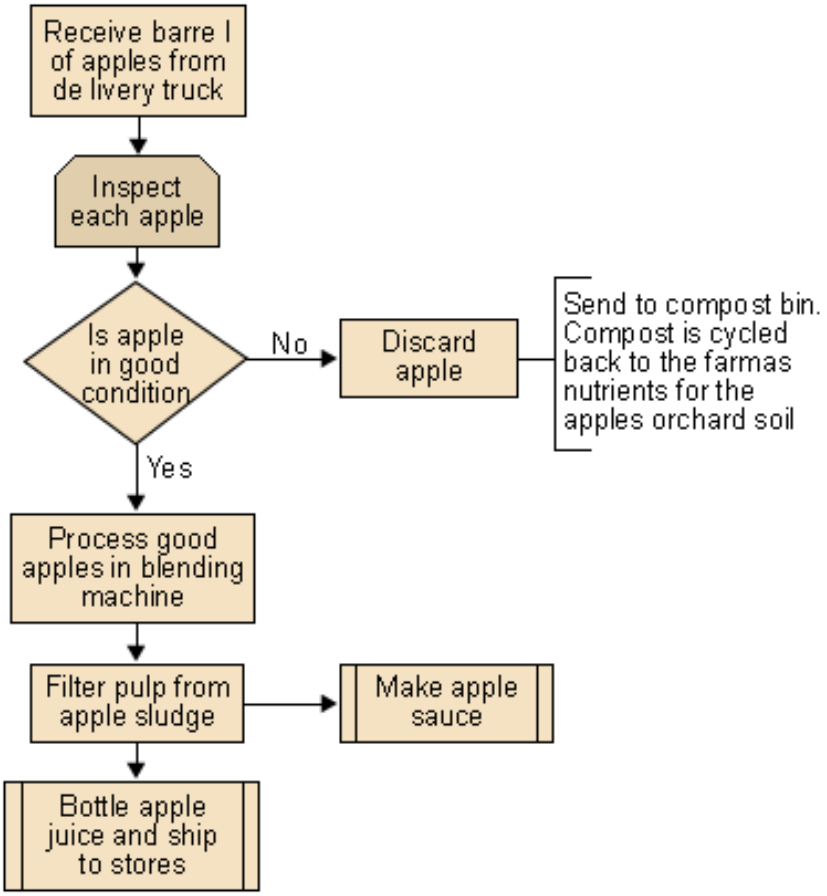
Diagramas "conceptuales de sitios web"



- Diagramas de procesos y flujos de datos. Permiten diseñar y reflejar los procesos de negocio y los flujos de datos y documentos de los diferentes departamentos de una compañía y su interacción.

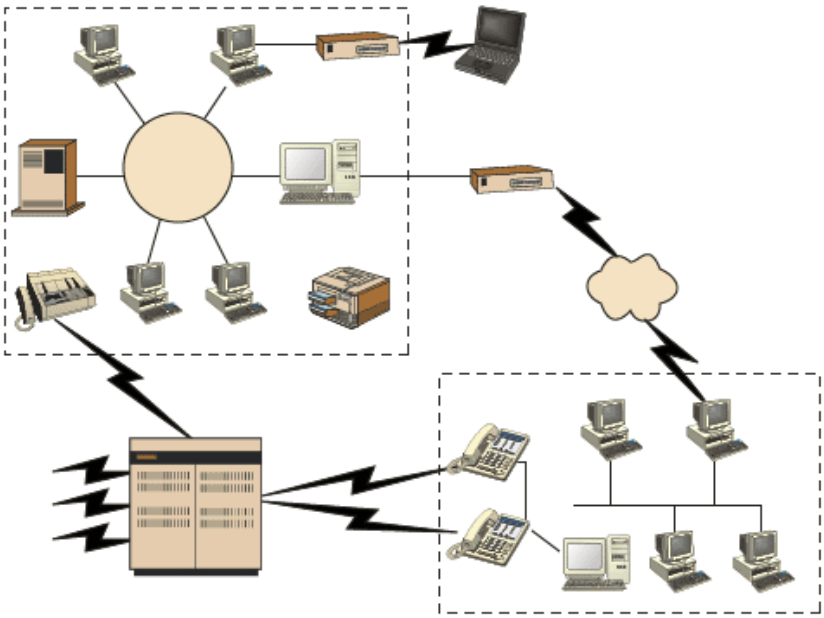
Diagramas de procesos y flujos de datos





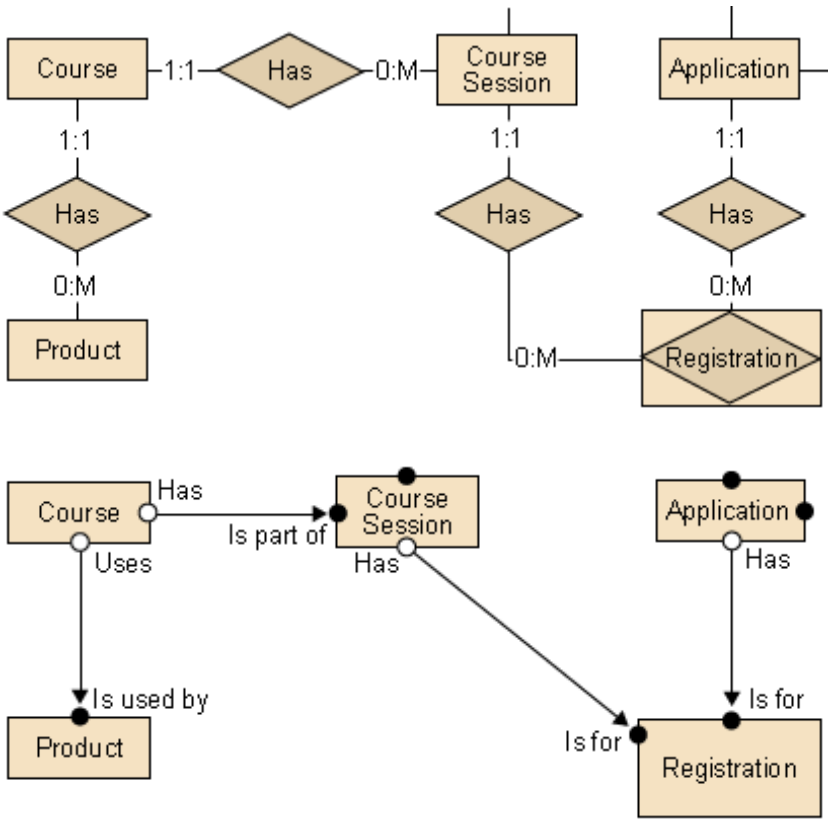
- Diagramas de redes. Permiten crear y diseñar el diagrama tanto lógico como físico de una red de ordenadores y de sus dispositivos.

Diagramas de redes



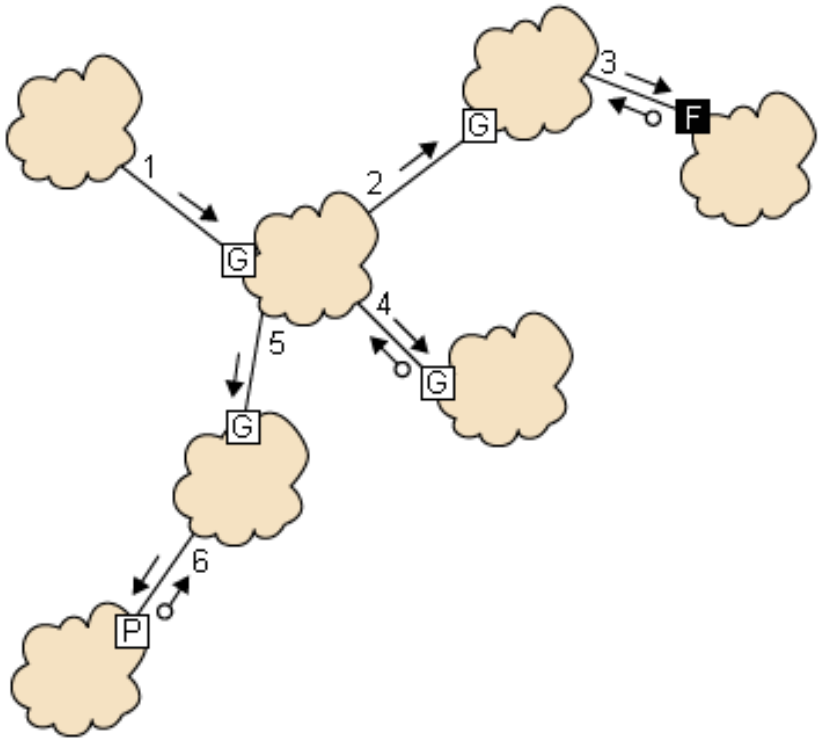
- Diagramas de modelos conceptuales de BD.

Diagramas de modelos conceptuales de BD



- Diagramas de modelos "orientados a objetos". A partir de diferentes notaciones (Booch OOD, los casos de uso de Jacobson, la notación ERD Martin, el lenguaje de modelización ROOM o la metodología OMT de Rumbaugh), este tipo de diagramas permiten diseñar aplicaciones orientadas a objetos. La mayoría de las herramientas CASE también poseen funciones para generar código de programación a partir de estos diseños.

Diagramas de modelos "orientados a objetos"



2.3. Introducción a la ingeniería del software

El objetivo de este material no consiste en cubrir y tratar la ingeniería del software, sino en ofrecer una breve introducción y visión de conjunto de sus principales conceptos.

La ingeniería del software nació de la necesidad de estandarizar el proceso de producción del software y minimizó el riesgo de errores en su desarrollo y mantenimiento.

A continuación, se describen las principales fases de la producción del software, la importancia del mantenimiento de los programas y los principales elementos de la ingeniería del software.

2.3.1. La producción de software

Tal como se ha mencionado en los apartados anteriores, el objetivo y el uso de los lenguajes de programación es la construcción de software, es decir, aplicaciones que se ejecutarán en un sistema informático (hardware, red, etc.) con el fin de llevar a cabo un conjunto de funcionalidades y operaciones.

El proceso de producción del software tiene una estructura general donde se podrían distinguir tres fases genéricas: la definición, el desarrollo y el mantenimiento.

Las tareas que se realizan durante cada una de estas fases de producción son las siguientes:

- **Definición**: esta fase se focaliza en el qué, es decir, "qué cosa" se ha de desarrollar, y en ella se define qué información se debe procesar, qué función y rendimiento se desea del futuro programa, qué restricciones se han de imponer y qué criterios de validación son necesarios para que el futuro sistema desarrollado se pueda considerar correcto.

Toma de requerimientos

Dentro de la definición, está la toma de requerimientos al solicitante del software. A menudo, quien pide el nuevo software conoce perfectamente el problema que tiene y que quiere resolver con el nuevo software, pero no tiene claro la solución que resuelve su problema. Para evitar que la fase de definición se convierta en algo imposible de abarcar, hace falta que el ingeniero de software conozca muy bien la tecnología con la que quiere implementar la solución y que hable con todos los implicados en el problema, desde los usuarios a los niveles jerárquicos más elevados. Una vez se han recogido todos los datos y deseos de todos los implicados, cabe presentar una propuesta que contente a todas las partes, solucione el problema y sea implementable técnicamente dentro del tiempo pactado y dentro del presupuesto que se tenga. Prometer soluciones poco realistas hará fracasar el proyecto.

- **Desarrollo**: la segunda fase se centra en el cómo. Así, durante esta fase es necesario decidir cómo transformar aquello definido anteriormente en un producto software, cómo se han de diseñar las estructuras de datos y la arquitectura de software, cómo se implementan los diferentes detalles procedimentales, cómo se ha de trasladar el diseño a un lenguaje de programación y cómo se debe realizar la prueba de validación una vez generado el programa.

La tecnología

Acertar la tecnología con la que se quiere desarrollar el software puede ayudar mucho al éxito del mismo. Un buen desarrollo siempre es más ágil si detrás hay un buen análisis y las herramientas de desarrollo adecuadas, tanto de software como de hardware. Disfrutar de los perfiles oportunos es también de vital importancia para esta fase.

- **Mantenimiento**: finalmente, y una vez que el programa se ha desarrollado, se entra en la fase de realizar y llevar a cabo cambios asociados con la corrección de errores, la adaptación a nuevos entornos y el aumento de la funcionalidad.

Tipo de mantenimiento

Hay dos tipos básicos de mantenimiento: el correctivo y el evolutivo. El mantenimiento correctivo tiene una gran actividad en los primeros meses de existencia del software, pero poco a poco tiende a ir desapareciendo a medida que se van abarcando todos los errores. El mantenimiento evolutivo, en cambio, tiene una gran actividad durante todo el ciclo de vida del software, ya que consiste en corregir, pero también en evolucionar y mejorar, el software a medida que surgen nuevas necesidades o ideas. A menudo, algunos cambios del mantenimiento evolutivo necesitan ser tratados como un pequeño proyecto de software, con su toma de requerimientos, análisis, presupuesto, etc.

Las tres principales fases del desarrollo del software son la definición, el desarrollo y el mantenimiento posterior.

2.3.2. El mantenimiento del software

El software no se deteriora, como sí que le sucede al hardware.

El hardware presenta relativamente muchos errores al principio de su vida (construcción), a menudo atribuibles a defectos de diseño o de fabricación o de especificaciones.

Cuando se corrigen los defectos, disminuyen los errores hasta el nivel más bajo, donde quedan estacionarios durante un determinado período de tiempo, hasta que llega un momento de su vida en el que empieza a mostrar de nuevo errores como consecuencia del paso del tiempo y de otros daños externos que deterioran los componentes.

Aun así, la realidad es que el software queda obsoleto antes de que se deteriore.

Una de las razones de esta obsolescencia se debe a la rapidez con la que evoluciona el software, que provoca que cada vez se exijan más recursos de hardware.

Otra razón es la evolución del entorno empresarial, científico, docente, etc., que hace que los flujos de trabajo vayan variando y que el software ya no responda a todas las nuevas necesidades surgidas.

Con frecuencia, al solventar los errores iniciales, se incluyen modificaciones que hacen que aparezcan o salgan a la luz nuevos errores. Para evitarlo, hay que enfatizar la rigurosidad de los análisis y las fases de test.

En general, el mantenimiento del software tiene una complejidad considerablemente mayor que el mantenimiento del hardware.

Un error en el software se debe a incoherencias en las especificaciones, a un error en el diseño o a un desarrollo mal resuelto. No existen piezas de recambio, como en el caso del hardware; eso obliga a que toda solución se tenga que resolver con más desarrollo, algo también sometido a la posibilidad de contener errores. Así, en el software no existen piezas de recambio, pero un buen diseño para ayudar a simularlas. Todas las tareas del software se pueden diseñar e implementar como módulos individuales que interactúan entre ellas, de manera que si una parte queda obsoleta por la tecnología se puede rehacer sólo el módulo implicado. Tener el código modularizado también facilita el mantenimiento y sus reos en diferentes proyectos.

El software no se deteriora como el hardware. Aun así, un error en el diseño y desarrollo de un programa obliga a su reprogramación, ya que no hay piezas de recambio.

2.3.3. Principales elementos de la ingeniería del software

Vista la importancia de garantizar el menor número de errores en la producción del software y, al mismo tiempo, la necesidad de estandarizar el desarrollo de las aplicaciones, apareció la ingeniería del software.

La ingeniería del software engloba un conjunto de tres elementos clave (métodos, herramientas y procedimientos) que facilitan que el gestor controle el proceso de desarrollo del software y que suministran las bases para construir software de alta calidad de una manera productiva.

- Los métodos de la ingeniería del software indican cómo construir técnicamente el software e incluyen un amplio espectro de tareas, desde la planificación y estimación de proyectos, hasta el análisis de los requerimientos, el diseño, la codificación, la prueba y el mantenimiento.

- Las herramientas de la ingeniería del software suministran un apoyo automático o semiautomático para los métodos. En la actualidad, existen herramientas para soportar cada uno de los métodos mencionados anteriormente. La ingeniería del software asistida por ordenador, conocida normalmente como CASE (*computer aided software engineering*) (podéis ver "Herramientas CASE"), es una herramienta que permite la automatización del proceso de producción de desarrollo del software. La idea básica de una herramienta CASE es proporcionar un conjunto de herramientas integradas que ahorren trabajo enlazando y automatizando todas las fases del ciclo de vida del software.

- Los procedimientos de la ingeniería del software son el nexo de unión entre los métodos y las herramientas, y facilitan un desarrollo racional y oportuno del software. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas (documentos, informes, códigos, etc.) que se requieren, los controles que ayudan a asegurar la calidad y coordinar los cambios, así como las guías que facilitan a los gestores de los software establecer su desarrollo.

La ingeniería del software incluye y define los métodos, las herramientas y los procedimientos para el desarrollo del software.

3. Gestión y planificación de proyectos informáticos

3.1. La gestión de proyectos

Si se quisiera definir qué es un proyecto, se encontrarían distintos conceptos y definiciones.

Según la Asociación Francesa de Normalización, un proyecto es "un proceso específico que permite estructurar metódica y progresivamente una realidad futura".

Otra definición sería considerar un proyecto como "un conjunto de recursos humanos y materiales necesarios para conseguir un objetivo delimitado por unos costes, unos plazos y unos resultados".

Y, finalmente, atendiendo al Project Management Institute (PMI), un proyecto se define como "un conjunto de esfuerzos, temporales, dedicados a crear un servicio o producto único".

En todo caso, independientemente de la definición, se puede concluir que un proyecto presenta las características siguientes:

- Una delimitación temporal.
- Unos objetivos precisos de plazos, costes y resultados que se deben obtener.
- No es una acción repetitiva, sino una acción creativa.
- Está compuesto por un conjunto de tareas complejas que requieren un trabajo de análisis, estudio y planificación.

3.1.1. Gestión de un proyecto

La gestión de un proyecto es la planificación, organización y dirección de las tareas o actividades y de los recursos necesarios para conseguir un objetivo definido, limitado temporalmente y con un presupuesto económico fijado.

Así pues, la finalidad de la gestión de proyectos es conseguir un objetivo específico en una fecha máxima y con un presupuesto determinado.

Ejemplos de proyectos pueden ser:

- La construcción de un local, supermercado, edificio, etc.
- El lanzamiento de un nuevo producto: plan de marketing, plan de ventas.
- El desarrollo de un producto nuevo: plan de I+D, plan de desarrollo.
- La organización de acontecimientos deportivos o culturales.
- El desarrollo de una aplicación informática.
- El desarrollo del proceso de fabricación de un vehículo nuevo.
- La gestión de una campaña electoral.
- La implantación de nuevos procesos organizativos en una empresa.
- La instalación de un sistema informático.

Un proyecto u objetivo puede ser tan "simple" como diseñar el prospecto de publicidad de un producto nuevo o tan "complejo" como la construcción de un centro comercial.

En cada caso, es necesario:

- dividir el proyecto en unas tareas o actividades,
- programarlas en fechas,
- asignar los recursos necesarios
- y, finalmente, llevar a cabo un seguimiento para evaluar el progreso del trabajo y detectar posibles desviaciones y poder corregirlas.

La gestión de proyectos ayuda a responder a preguntas. Para poder evaluar si el proyecto tiene desviaciones o no, una buena gestión de proyectos tiene que poder contestar a:

- ¿Cuánto tiempo exigirá este proyecto?
- Si una tarea se retrasa, ¿cuánto tiempo se retrasará el proyecto?
- ¿Se cuenta con suficientes recursos, humanos y materiales, con el fin de completar el proyecto, tal como hemos definido?
- ¿Cuáles son los costes asociados a los recursos para el proyecto?

- En caso de retraso, ¿qué puedo hacer para recuperar tiempo?

- Si se quisiera adelantar la finalización del proyecto, ¿qué recursos se deberían añadir y qué impacto económico supondría?

La gestión de un proyecto engloba la planificación, la organización y la dirección de las tareas o actividades y de los recursos necesarios para conseguir un objetivo definido, limitado temporalmente y con un presupuesto económico fijado.

3.1.2. Las fases de la gestión de un proyecto

Generalmente, la gestión de proyectos agrupa tres fases: la definición y la planificación, la dirección y el seguimiento, y la generación de informes.

- **Definición y planificación de un proyecto**. Constituye la parte más importante, ya que incluye la definición de las tareas y su duración, el análisis de las relaciones entre las tareas y la asignación de recursos a cada una, principalmente.
  Todas las fases posteriores del proyecto se llevarán a cabo según la información procedente de este diseño.

- **Dirección y seguimiento del proyecto**. Es la fase de seguimiento del proyecto una vez que ha empezado y acaba con el proyecto.
  Incluye el seguimiento de las tareas definidas, la asignación prevista de los recursos y los reajustes necesarios para reflejar los cambios que se producen mientras avanza el proyecto.

- **Generación de informes sobre el proyecto**. Una vez acabado el proyecto, en esta fase se producen los diferentes informes finales.
  Aun así, durante la fase de dirección también se generan informes de seguimiento y durante la fase de creación se elaboran los informes de planificación.

En función del tipo de proyecto u objetivo que se ha de alcanzar, estas fases se subdividirán en más o menos subetapas, pudiendo hablar, incluso, de modelos de proyecto según su tipología: construcción de un edificio, levantamiento de un puente, elaboración de una campaña de publicidad, o la implantación de una herramienta CRM (*customer relationship management*) o el diseño e implantación de una web, en el ámbito de los proyectos informáticos.

3.2. Planificación y seguimiento

La planificación de un proyecto, tal como se ha visto en el apartado "Gestión de un proyecto", constituye la primera fase de todo proyecto y consiste en definir qué tareas serán necesarias para conseguir el hito marcado, qué recursos, materiales y humanos, será necesario gestionar, cuánto tiempo se necesitará para llevarlo a cabo y, finalmente, qué costes asociados tiene.

Una vez se ha iniciado el proyecto, se debe realizar un seguimiento con el fin de evaluar la progresión de las tareas, comprobar la realización de éstas, identificar posibles retrasos y gestionar las incidencias que surjan.

3.2.1. Las tareas de un proyecto

Un proyecto está constituido por un conjunto de tareas (o actividades).

Una tarea es necesaria para el proyecto si su ejecución contribuye a finalizarlo y, por otra parte, si su no ejecución conlleva una dificultad en la consecución de los objetivos.

Una tarea se determina por las características siguientes:

- nombre,
- duración estimada,
- su interrelación con otras tareas del proyecto,
- los recursos que requiere para ser realizada,
- una descripción sobre qué consiste.

Un tipo de tarea particular son los hitos, que corresponden a tareas de duración nula y que sirven para indicar puntos de control del proyecto.

Hitos

Ejemplos de hitos son el inicio del proyecto o la consecución parcial de objetivos.

Los hitos también pueden corresponder a tareas externas al proyecto como, por ejemplo:

- La concesión de una licencia de obras por parte del ayuntamiento.
- La aprobación de un crédito bancario para llevar a cabo la inversión necesaria del proyecto.
- Etc.

En estos casos, dado que la tarea no es controlable por el equipo de trabajo, no se considera ni un tiempo ni un esfuerzo concreto. Aun así, en estos casos, aunque la responsabilidad de la tarea es externa al proyecto, sí que es necesario considerar los riesgos asociados a ésta, tal como se comentará más adelante en el apartado "Evaluación de riesgos y herramientas".

Por ejemplo, las tareas necesarias para planificar y llevar a cabo un programa de formación en la empresa podrían ser las siguientes:

1) Identificar las necesidades de los trabajadores.

2) Planificar el programa de formación.

3) Diseñar y elaborar los cursos o contratar a unos consultores externos para llevarlos a cabo.

4) Convocar a los participantes.

5) Llevar a cabo las actividades de formación.

6) Evaluar a los participantes.

7) Evaluar a los formadores y al programa.

En este caso, sería necesario realizar una división más exhaustiva de estas tareas o fases.

Un proyecto se constituye de un conjunto de tareas (o actividades) determinadas por distintas características como el nombre, la duración estimada, su interrelación con otras tareas del proyecto, los recursos que requiere para ser realizada y una descripción sobre en qué consiste.

3.2.2. Las interrelaciones entre las tareas

Las diferentes tareas de un proyecto no se realizan todas a la vez, ni todas de forma secuencial, una detrás de otra. Hay que analizar qué tareas son dependientes de otras, cuáles son independientes y, entre las que tienen dependencias, cabe priorizar la ejecución en función de la criticidad y de las dependencias entre ellas. Un jefe, es decir, quien tiene la orden de ejecución, ha de ordenarlas en el tiempo en función de los recursos humanos y sus capacidades, de manera que sea posible hacerlas todas respetando el orden y minimizando el tiempo de ejecución y, por lo tanto, también minimizando el coste.

Hacer todas las tareas de manera sucesiva comportaría alargar de modo innecesario el proyecto, y hacerlas todas simultáneamente es imposible, ya que algunas pueden depender de otras (es decir, para empezar una tarea hace falta haber finalizado otras).

Por ello, una vez se han identificado las principales tareas del proyecto, hay que dar un paso más y establecer el encadenamiento más lógico y conveniente entre ellas.

Unas tendrán una prioridad, otras se tienen que hacer en momentos diferentes, unas terceras serán secuenciales y otras se podrán hacer en paralelo.

Por ejemplo, en una construcción de un edificio, hay que finalizar las excavaciones antes de poder iniciar los trabajos de cimentación, y no se suele pintar hasta que los electricistas y los albañiles han finalizado su trabajo.

Estas relaciones se conocen con el nombre de "precedencia y sucesión de las tareas".

Las tareas o actividades de un proyecto no se realizan todas de manera paralela, sino que algunas se harán de manera consecutiva y dependerán de la finalización o el inicio de otras.

3.2.3. La gestión de los recursos

Para muchas personas, la gestión de los proyectos concluye en la planificación de las tareas, tratadas en el apartado anterior. Es decir, en la división del proyecto en tareas y subtareas, en la determinación de la duración de cada una y en la definición de las interrelaciones y vinculaciones existentes entre las distintas partes que componen el proyecto.

Desgraciadamente, sólo con este trabajo no se puede garantizar el éxito y la eficiencia de un proyecto.

Si no se introduce en la planificación del proyecto la gestión y asignación de recursos a cada una de las tareas, la planificación podrá alcanzar uno de los objetivos, que es la consecución de los hitos, pero no se contemplarán aspectos como el control de los costes, la correcta utilización y optimización de los recursos disponibles (personas, maquinaria, materiales, etc.).

Así pues, la realización de las tareas que se han identificado debe estar acompañada de la asignación de los recursos que se han de utilizar en cada una de ellas.

La gestión de los recursos es muy importante en la gerencia de un proyecto por varias razones:

- Los recursos disponibles, humanos, técnicos, financieros, siempre se limitan a una empresa u organización.
- Los tipos de recursos utilizados y su cantidad determinan los costes del proyecto.
- Los recursos y, en especial, los humanos y técnicos son los responsables de llevar a cabo las tareas en el tiempo previsto y con la calidad necesaria.
- Los proyectos exigen el uso de recursos muy variados (máquinas, especialistas en diferentes materias, mano de obra, fungibles, etc.).
- Estos recursos no se concretan de manera estable durante todo el proyecto, sino que para cada tarea se necesitan recursos diferentes en naturaleza y cantidad.

Existen dos aspectos básicos en la gestión de recursos: la previsión y el equilibrio de los recursos.

Por una parte, por *previsión* se entiende que cada actividad, al ser ejecutada, cuente con la garantía de disponer de los recursos necesarios, mientras que, por otra, el equilibrio tiene como requisito la previsión y procura cubrir las necesidades de los recursos utilizados durante el proyecto, de manera que se minimice la existencia de recursos ociosos (o sobrantes) durante la ejecución.

La realización de las tareas que se han identificado en un proyecto sólo se puede conseguir con la asignación de los recursos humanos responsables de su ejecución y los recursos materiales que se han de utilizar en cada una.

Para saber más

En el apartado "El equipo de trabajo" de este mismo módulo se presenta una descripción de los principales roles que pueden participar en un proyecto de sistemas informáticos.

3.2.4. La redistribución de los recursos

Tal como se comentaba en el apartado anterior, uno de los objetivos de la gestión de los recursos consiste en minimizar la existencia de recursos ociosos y, a la vez, garantizar que todas las tareas del proyecto dispondrán de los recursos necesarios en el momento justo.

Así, tal como se ha visto en el apartado "Planificación y seguimiento", una de las principales actividades en la fase de seguimiento es detectar las incidencias y los retrasos de las diferentes actividades del proyecto e intentar corregirlos.

Los cambios de planificación y retrasos provocan también desajustes en la gestión de los recursos, ya que éstos se deben reasignar en función de estos cambios.

Por otra parte, tal como ya hemos mencionado, los recursos suelen ser escasos y limitados, por lo que, a veces, la menor disponibilidad de estos recursos, con respecto a la previsión realizada en la fase de definición, obliga a redistribuir las cargas de estos recursos durante la vida del proyecto.

Una responsabilidad importante del jefe de proyectos y del gerente de recursos es aplicar técnicas y utilizar metodologías adecuadas para la correcta asignación, reasignación y resolución de sobreasignaciones de recursos a las tareas.

3.2.5. Gráficos de tiempo y diagramas

Existen dos diagramas básicos en la gestión de proyectos, el diagrama de Gantt y el diagrama Pert, que tienen la forma siguiente:

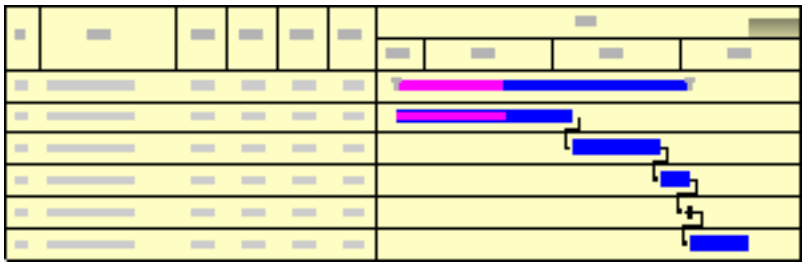


Diagrama de Gantt

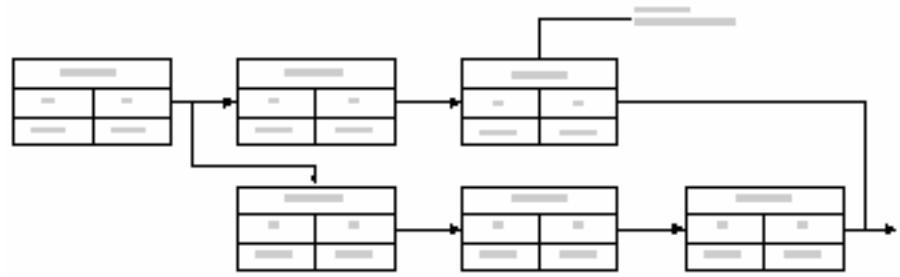


Diagrama Pert

Cada uno presenta una serie de ventajas e inconvenientes, por lo que es el uso de los dos, combinados, la mejor opción para la gestión de proyectos.

Una manera no gráfica de representar las tareas en un proyecto es por medio de tablas.

Un ejemplo podría ser:

Tabla de tareas

Número tarea	Nombre tarea	Duración (en días)
1	Inicio	0

Número tarea	Nombre tarea	Duración (en días)
2	A	4
3	B	3
4	C	2
5	D	2
6	E	2
7	F	4
8	G	5
9	H	4
10	I	2
11	J	2
12	Fin	0

El paso siguiente que se ha de considerar son las interrelaciones entre las tareas, lo que se conoce como precedencia y sucesión de las tareas.

Para saber más

Para ampliar información sobre las interrelaciones entre las tareas, podéis consultar el apartado "Las interrelaciones entre las tareas" de este mismo módulo.

Suponiendo las precedencias siguientes:

- ADH (la tarea D no se puede empezar hasta que se finalice la tarea A, y tras la tarea D está la tarea H).
- BEIJ.
- CFIJ.
- CGH.
- IJ (cuando acabe la tarea I se podrá empezar la tarea J).

Para ejemplificarlo, podríamos encontrar que:

- La tarea A es el hito de inicio del proyecto.
- La tarea D es la recopilación de los datos necesarios para llevar a cabo la definición funcional.
- La tarea H es el diseño funcional de la solución que se ha de implementar.

O, por ejemplo, que:

- La tarea C es la definición de la arquitectura de la infraestructura del sistema (hardware).
- La tarea F es la adquisición del hardware necesario.
- La tarea I es la instalación de este hardware.
- La tarea J es la configuración de las máquinas.

Estas dependencias se reflejarían de la manera siguiente en la tabla:

Tabla con precedencias

Número tarea	Nombre tarea	Duración (en días)	Tarea precedente
1	Inicio	0	
2	A	4	
3	B	3	
4	C	2	
5	D	2	A
6	E	2	B
7	F	4	C
8	G	5	C
9	H	4	D;G
10	I	2	E; F
11	J	2	I
12	Fin	0	

O si se considerara el número de la tarea, en lugar del nombre:

Tabla con precedencias

Número tarea	Nombre tarea	Duración (en días)	Tarea precedente
1	Inicio	0	
2	A	4	
3	B	3	
4	C	2	
5	D	2	2
6	E	2	3
7	F	4	4

Número tarea	Nombre tarea	Duración (en días)	Tarea precedente
8	G	5	4
9	H	4	5; 8
10	I	2	6; 7
11	J	2	10
12	Fin	0	

Siguiendo con el ejemplo, puesto que las tareas A, B y C (2, 3 y 4) no dependen de ninguna otra, empezarán todas a la vez y se podrá considerar que el proyecto ha finalizado cuando se hayan llevado a cabo las tareas H, I y J (9, 10 y 11).

Esta tabla de tareas se podría representar, también gráficamente, con un diagrama de Gantt.

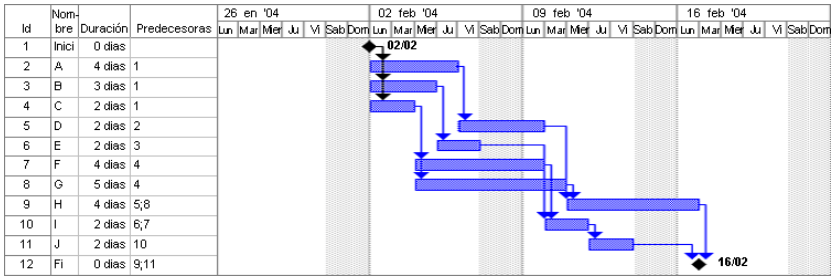


Diagrama de Gantt

Existen diferentes gráficos y diagramas que representan las tareas de un proyecto y sus interrelaciones. Los principales son el diagrama de Gantt, el diagrama de Pert y las tablas de tareas.

3.2.6. Herramientas, métodos y técnicas

Camino crítico

Una de las principales técnicas o métodos de gestión de proyectos es el cálculo del camino crítico.

Un camino es cada uno de los encadenamientos necesarios de tareas, partiendo de las tareas iniciales y siguiendo las sucesiones (precedencias) definidas.

Ejemplo anterior

En el ejemplo del apartado anterior, los diferentes caminos serían ADH, BEIJ, CFIJ, CGH.

Cada uno de estos caminos tiene una duración prevista, que es la suma de los tiempos necesarios para realizar cada tarea.

Al estar encadenadas, la duración total del camino es la suma de sus duraciones.

Siguiendo el ejemplo, las duraciones de los diferentes caminos del proyecto son las siguientes:

- ADH = 4 + 2 + 4 = 10 (el tiempo necesario para llevar a cabo las tareas A, D y H es de 10 días).
- BEIJ = 3 + 2 + 2+ 2 = 9
- CFIJ = 2 + 4 + 2 + 2 = 10
- CGH = 2 + 5 + 4 = 11

Es muy importante, en todo proyecto, detectar el camino crítico, que es el más largo.

Su importancia viene dada por dos factores:

- Todo retraso en una de las tareas del camino crítico provocará que el proyecto, en conjunto, se retrase. Es decir, que finalice más tarde.
- Todo adelanto en la ejecución de una tarea que pertenezca al camino crítico supondrá un adelanto en la ejecución del proyecto.

Ejemplo anterior (continuación)

Según el ejemplo, un retraso de un día en una de las tareas B o E no supondría un retraso en el proyecto:

- ADH = 10
- BEIJ = 10
- CFIJ = 10
- CGH = 11

En cambio, un adelanto (recuperación de tiempo) de dos días en la tarea G provocaría que el proyecto pudiera acabar antes:

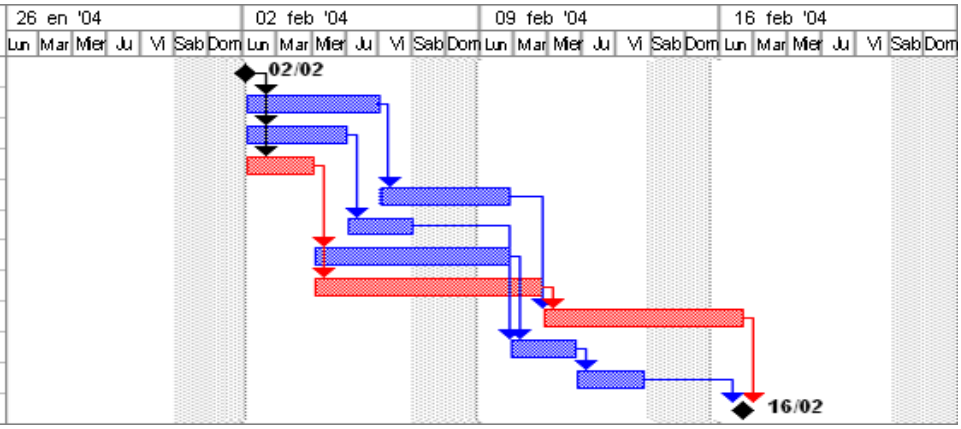
- ADH = 10
- BEIJ = 9
- CFIJ = 9
- CGH= 9

En este caso, además, el cambio de duración de las tareas supone que el camino crítico pase a ser el que forman las tareas ADH.

El camino crítico y las tareas críticas se suelen destacar en los gráficos con el objetivo de llamar la atención de los responsables del proyecto a fin de que maximicen su seguimiento.

Partiendo del ejemplo inicial.

Id	Nombre	Duración	Predecesoras
1	Inici	0 dias	
2	A	4 dias	1
3	B	3 dias	1
4	C	2 dias	1
5	D	2 dias	2
6	E	2 dias	3
7	F	4 dias	4
8	G	5 dias	4
9	H	4 dias	5;8
10	I	2 dias	6;7
11	J	2 dias	10
12	Fi	0 dias	9;11



Los márgenes

Las tareas que no pertenecen al camino crítico se dice que tienen margen, es decir, que su momento de inicio y de fin no se encuentran rígidamente marcados, sino que pueden variar, un margen, en el tiempo, sin que ello afecte a la fecha de finalización del proyecto.

Esta información es muy importante para el jefe de proyectos, ya que cuenta con unos márgenes de tiempo de retraso o adelanto de determinadas tareas con el fin de compensar desviaciones del proyecto, sobreasignaciones de recursos, etc.

A partir de la duración de cada tarea y de la relación con sus predecesoras, estos métodos permiten calcular:

- "tan pronto como" puede empezar una tarea,
- "tan pronto como" puede finalizar una tarea,
- "tan tarde como" puede empezar una tarea,
- "tan tarde como" puede finalizar una tarea.

De manera simplificada se denomina *tan pronto* al tiempo más temprano que una tarea puede empezar o acabar y *tan tarde* al tiempo más tarde admisible para una tarea.

Para ello se utilizan dos métodos de cálculo de las fechas "tan pronto" y "tan tarde" de un proyecto atendiendo a la ordenación "tan pronto" y a la ordenación "tan tarde" respectivamente.

- La ordenación "tan pronto como" determina para cada tarea la fecha de inicio y de fin "tan pronto como", teniendo en cuenta las relaciones con las tareas predecesoras y sus duraciones. En el ejemplo, suponiendo que el proyecto se inicia el día 1, la tarea inicio no tenía duración, por lo que, la tarea A empieza el día 1 y acaba el día 4 y la tarea D empezará el día 5 y acabará el día 6.

- La ordenación "más tarde" calcula para cada tarea la fecha de comienzo "más tarde" y la fecha de finalización "tan tarde" sin cambiar la fecha del proyecto determinada por la planificación "tan pronto". Con el fin de determinar la planificación "tan tarde" se debe empezar por la última tarea del proyecto y "remontar" sucesivamente hasta el inicio.

Una vez calculadas las fechas de inicio y finalización "tan pronto" y "tan tarde" se pueden calcular los márgenes.

El uso correcto de los márgenes es importante para la gestión de los proyectos, ya que las tareas con margen se pueden desplazar en el tiempo, adelantarse o retrasarse, dentro de los márgenes establecidos por la medida del margen sin que esto afecte al plazo del proyecto.

Cálculo de las fechas "tan pronto"

El proyecto se inicia el día 1:

a) Para una tarea con una sola predecesora

- el comienzo "tan pronto" de una tarea es igual a la finalización "tan pronto" de la tarea predecesora + 1. Siguiendo el ejemplo, para la tarea A = 0 + 1 = 1 y para la tarea D = 4 + 1 = 5

- el final "tan pronto" de una tarea es igual al principio "tan pronto" de la tarea + la duración de la tarea – 1. Por ejemplo, el final "más bien" de la tarea D es igual a 5 + 2 – 1 = 6. La tarea acabará el día 6 y por ello se resta una unidad en la fórmula.

b) Para una tarea con diferentes predecesoras

- El comienzo "tan pronto" de una tarea es el valor máximo de las finalizaciones "tan pronto" de las tareas predecesoras + 1. Por ejemplo, para la tarea H = Máx (6, 7) + 1 = 7 + 1 = 8

- El final "tan pronto" de una tarea es igual al principio "tan pronto" de la tarea + duración – 1. Para la tarea H del ejemplo, sería 8 + 4 – 1 = 11.

La tabla completa de las fechas "tan pronto" del ejemplo sería la siguiente:

Número tarea	Nombre tarea	Duración (en días)	Tarea precedente	Inicio "tan pronto"	Fin "tan pronto"
1	Inicio	0		0	0
2	A	4	Inicio	1	4(*)
3	B	3	Inicio	1	3
4	C	2	Inicio	1	2
5	D	2	A	5	6
6	E	2	B	4	4
7	F	4	C	3	6
8	G	5	C	3	7
9	H	4	D;G	8	11
10	I	2	E; F	7	8
11	J	2	I	9(**)	10
12	Fin	0	H; I; J	11	11

(*) La fecha de finalización "tan pronto" es el comienzo "tan pronto" (1) más la duración (4) menos 1 = 4
(**) La fecha de comienzo "tan pronto" es la finalización "tan pronto" de la predecesora (8) más 1 = 9

Cálculo de las fechas "tan tarde"

El proyecto acaba el día 11.

a) Para una tarea con una sola sucesora

- la finalización "tan tarde" de una tarea es la finalización "tan tarde" de la sucesora – la duración de la tarea sucesora. Por ejemplo, para la tarea J sería 11 – 0 = 11 y para la tarea I, 11 – 2 = 9

- el comienzo "tan tarde" de una tarea es la finalización "tan tarde" de la tarea – duración + 1. Como ejemplo, para la tarea I, el comienzo "tan tarde" es 9 – 2 + 1 = 8.

b) Para una tarea con diferentes sucesoras

- La finalización "tan tarde" de una tarea es el mínimo de los comienzos "tan tarde" de las tareas sucesoras – 1. Por ejemplo, para la tarea I = Mín. de los comienzos "tan tarde" (J, Fin) – 1 = Mín. (10, 11) – 1 = 9.

- El comienzo "tan tarde" de una tarea es la finalización "tan tarde" de la tarea – duración + 1. Para la tarea I del ejemplo, sería 9 – 1 + 1 = 8.

La tabla completa de las fechas "más pronto" y "más tarde" será la siguiente:

Número tarea	Nombre tarea	Duración (en días)	Tarea precedente	Inicio "tan pronto"	Fin "tan pronto"	Inicio "tan tarde"	Fin "tan tarde"
1	Inicio	0		0	0	1	1
2	A	4	Inicio	1	4	2	5
3	B	3	Inicio	1	3	3	5
4	C	2	Inicio	1	2	1	2
5	D	2	A	5	6	6	7
6	E	2	B	4	4	6	7
7	F	4	C	3	6	4	7
8	G	5	C	3	7	3	7
9	H	4	D;G	8	11	8	11
10	I	2	E; F	7	8	8	9
11	J	2	I	9	10	10	11
12	Fin	0	H; I; J	11	11	11	11

Se puede comprobar, por ejemplo, que en algunas tareas, los tiempos "tan pronto" y "tan tarde" coinciden, es decir, las fechas "tan pronto" y "tan tarde" de comienzo y final son las mismas, como, por ejemplo, las tareas C, G y H.

En cambio, otras tareas como, por ejemplo, la E, tienen margen, lo que significa que se pueden iniciar en diferentes momentos o se pueden alargar sin alterar el plazo final previsto.

Métodos de redistribución

Los métodos de redistribución de los recursos pretenden conseguir un diagrama de carga tan uniforme como se pueda.

La aplicación de estos métodos de redistribución se puede deber a varias razones:

- Cambios en la planificación de las tareas.

- Incidencias ocurridas durante el proyecto y retrasos.

- Cambios en disponibilidad de los recursos previstos.

- Solapamiento temporal de tareas que requieren el mismo tipo de recurso.

Esta operación de redistribución resulta mucho más complicada de lo que pueda parecer.

Para poder llevarla a cabo, es necesario redibujar el gráfico de Gantt, basándose en la tabla de los tiempos de "Antes y Después" y los márgenes totales que puede soportar cada tarea. De esta manera, se podrá evaluar la posibilidad de realizar las tareas en momentos en los que se disponga de recursos sobrantes (ociosos o sin asignación).

Las tareas que componen el camino crítico no tienen margen y, por lo tanto, si no se dispone de recursos para llevarlas a cabo, es imposible que el proyecto acabe en la fecha prevista.

Por esta misma razón, en la redistribución de recursos se ha de dar prioridad a las tareas críticas.

Una vez redistribuidos los recursos, si continúa habiendo problemas de asignación, es necesario:

- evaluar el posible retraso o adelanto de tareas, lo cual puede implicar un retraso en el proyecto que provocará una desviación en tiempo,

- incorporar nuevos recursos para llevar a cabo estas tareas, lo cual implicará un aumento de los costes previstos que provocará una desviación presupuestaria.

3.3. Proyectos informáticos

El resultado final de un proyecto informático es un sistema informático compuesto de un conjunto de elementos y módulos que procesan y gestionan informaciones y datos, por medio de diferentes métodos, procedimientos y controles.

Un sistema informático, tal como hemos comentado en las aplicaciones del software (podéis ver el apartado "Aplicaciones del software") puede tener unas funcionalidades y estar dirigido a unos entornos muy diversos.

En el apartado anterior hemos presentado los principales conceptos de la gestión de proyectos generales.

Ahora bien, un proyecto informático presenta unas particularidades y características específicas, sobre todo por los elementos que componen un sistema informático:

- Hardware. Dispositivos electrónicos y dispositivos electromecánicos, como la CPU y la memoria (podéis ver el módulo "El hardware"), los circuitos integrados, componentes electrónicos, tarjetas de circuito impreso, etc.

- Software. Son los diferentes programas y estructuras de datos, así como las aplicaciones que se integrarán o utilizarán con los sistemas, como los gestores de bases de datos, los servidores de mensajería, servidores de aplicaciones o Internet, por ejemplo.

- El equipo de trabajo. Compuesto por diferentes profesionales que adoptan un papel o más.

Estas características y particularidades hacen que en la gestión de proyectos informáticos sea necesario tener en cuenta diferentes fases, como el diseño, el desarrollo e integración, la verificación y la implantación, y la documentación necesaria, tanto para explicar el uso y las operaciones del sistema como para dar fe de cómo se ha implementado el sistema para facilitar un posterior mantenimiento.

En los apartados siguientes introduciremos los principales conceptos en la definición inicial del proyecto (evaluación de riesgos, recursos, equipo de trabajo, evaluación de esfuerzos), el diseño de la solución, la gestión de los costes, el desarrollo y la integración, así como la documentación y el mantenimiento del software.

Un proyecto informático presenta unas características particulares, no sólo por las tareas que se han de desarrollar (ingeniería del software), sino también por los elementos que componen los sistemas informáticos: el hardware y el software.

3.3.1. Fases y tareas principales

Todo proyecto informático sigue las mismas fases, aunque en función de su tipología y naturaleza, se llevan a cabo más o menos subfases.

En los apartados siguientes se definirá, con más detalle, cada una de estas fases:

- Diseño de la solución. Tiene como objetivo crear una solución conceptual, identificando la aplicación de las principales funcionalidades, las características necesarias y las adaptaciones que se han de llevar a cabo. También se definen diferentes modelos operacionales para el uso y trabajo del futu-

ro sistema. Asimismo, se diseñan los principales elementos y módulos y se prepara el entorno de desarrollo, la arquitectura del sistema y los procesos de aseguramiento de la calidad del producto. En esta fase se llevan a cabo los procesos siguientes, principalmente:

- inicio del proyecto,
- requerimientos de negocio,
- requerimientos funcionales,
- diseño general,
- diseño detallado,
- preparación del entorno de desarrollo.

- Desarrollo y construcción. El objetivo de la fase de desarrollo y construcción es la creación de los códigos de los componentes, la definición y configuración de las bases de datos, la construcción de los programas de las aplicaciones, la documentación y las pruebas de aseguramiento de calidad. Los dos principales procesos son el desarrollo y la preparación de la implantación.

- Implantación. En esta fase final se configura el entorno de producción, se verifica la solución completa, se ajustan los parámetros finales, se llevan a cabo las pruebas de funcionalidad y se forma a los usuarios de la solución, y se entregan los documentos de uso y trabajo.

Aun así, previamente al diseño se ha de definir el proyecto evaluando los riesgos, los recursos necesarios y definiendo el equipo de trabajo.

Esta planificación definirá los costes necesarios.

Y para obtener esta planificación se ha de realizar una evaluación y estimación siguiendo un conjunto de herramientas y métodos.

Una vez definido el proyecto, es necesario realizar el diseño de la solución, desarrollarla e implantarla.

Evaluación de riesgos y herramientas

Se puede definir el riesgo en la gestión de proyectos como la incertidumbre que implica la ejecución de una estimación. Es decir, la posibilidad de que surjan imprevistos y cambios que no permitan cumplir la planificación tanto en tiempo como en costes, trabajo o disponibilidad de recursos, entre otros.

Todo proyecto supone unos riesgos asociados, más o menos probables y graves, en función de su complejidad.

En todo proyecto informático suele haber diferentes componentes o elementos que pueden suponer un riesgo:

- La misma naturaleza del software. Errores de desarrollo, herramientas poco probadas, incompatibilidades de integración y comunicación entre aplicaciones diferentes, etc.

- Problemas asociados con el hardware. Fallos en las máquinas, errores de integración de sistemas, etc.

- La complejidad del proyecto. Cuanto mayor sea un proyecto y más elementos intervengan en él (número de gente, proveedores, máquinas, aplicaciones, etc.), más probabilidades existen de que se den fallos en el equipo de producción.

- El tamaño del proyecto. Cuanto mayor y más complejo sea el proyecto, más complicado será llevar a cabo su valoración y planificación y, por lo tanto, existirán más probabilidades de error.

Por lo tanto, pues, en todo proyecto informático, en su fase inicial y dentro de la valoración, será necesario evaluar los riesgos y tomar las medidas pertinentes.

Este proceso se puede dividir en tres pasos:

- identificación de los riesgos,

- evaluación de los riesgos,

- actuaciones correspondientes.

Para identificar los riesgos se deben tener en cuenta los aspectos siguientes:

- La descripción final del producto o productos del proyecto.

- La planificación de otras áreas con las cuales interaccione nuestro proyecto.

- La información histórica en el desarrollo de otros proyectos informáticos similares.

En esta primera fase de identificación de los riesgos se deben conseguir los elementos siguientes:

- Relación de posibles fuentes de riesgo. Tienen una probabilidad alta o media de suceder.

- Una lista de riesgos potenciales, que tienen una probabilidad menor de suceder como, por ejemplo, la marcha, de la empresa, de parte del equipo de trabajo o cambios en la tecnología de implementación del software.

- Una lista de síntomas de riesgo.

Una vez recogida e identificada esta información, se ha de averiguar la probabilidad de que ocurra cada uno de estos riesgos y qué consecuencias tendrían para el proyecto.

Para ello se deberán llevar a cabo los pasos siguientes:

- Definir una escala de valores que permita asignar a cada riesgo su probabilidad de que suceda.

- Enumerar las consecuencias de estos riesgos. Es decir, estimar el impacto del riesgo en el proyecto y en el producto o productos finales.

- Identificar en qué momento podría suceder y cuánto tiempo duraría.

Finalmente, el tercer paso, una vez se disponga de estos datos, es decidir qué riesgos se han de prevenir y para qué otros es más rentable asumirlos sin establecer ninguna medida.

En el segundo caso, simplemente, es necesario aceptarlos y desarrollar, si procede, un posible plan de choque por si sucedieran.

Para el primer grupo de riesgos se puede optar por dos alternativas:

- Evitarlos, suprimiendo la causa.
- Si no es posible evitarlos, se deberá reducir la posibilidad de que ocurran.

La gestión de los riesgos en un proyecto consiste en el análisis de las posibles incidencias que puedan surgir, la evaluación de su magnitud, el impacto y las consecuencias que tendrían en el proyecto y los productos que se deben desarrollar y, finalmente, tomar las medidas adecuadas, tanto para evitarlos como para disminuir la probabilidad o, simplemente, aceptarlos.

Recursos necesarios

Durante la primera fase de definición y planificación del proyecto se deberán especificar los recursos necesarios para llevarlo a cabo.

Tal como ya hemos comentado en el apartado anterior, se habrá de definir qué recursos son necesarios, cómo se pueden conseguir (contratación, miembros de la organización, alquiler, etc.), cuándo se necesitarán y durante cuánto tiempo.

La diferencia, sin embargo, tal como ya hemos comentado anteriormente, es que se deberán tener en cuenta dos tipologías especiales de recursos: los recursos de software y de hardware.

Para saber más

Para tener más información sobre el tipo de recursos y su gestión podéis consultar el apartado "La gestión de los recursos" de este mismo módulo.

- Los recursos de hardware, que se necesitan tanto para desarrollar el proyecto (servidores de desarrollo, máquinas de verificación, etc.), como los que llevará o los que formarán parte del producto final (infraestructura de producción, terminales, dispositivos para los usuarios, etc.).

- Recursos de software, que engloban tanto los lenguajes de desarrollo, los compiladores o intérpretes correspondientes, las herramientas de apoyo al diseño y producción, como el software resultante del proyecto (programa y/o aplicaciones que formarán parte del nuevo sistema informático).

Para saber más

Para tener más información sobre los lenguajes de desarrollo podéis consultar el apartado "Principales lenguajes de desarrollo". Para tener más información sobre compiladores o intérpretes podéis consultar el apartado "Traductores". Para tener más información sobre las herramientas de apoyo al diseño y producción, podéis consultar el apartado "Herramientas CASE". Todos estos apartados los encontraréis en este mismo módulo.

En un proyecto informático no sólo se han de considerar los recursos humanos y materiales, sino también los recursos de software y de hardware.

Sólo como introducción, enumeraremos a continuación algunos de los recursos de software que pueden utilizarse y ser necesarios para llevar a cabo el proyecto:

- Herramientas de análisis y diseño que permiten crear los modelos de software con el fin de validar la consistencia y la validez del diseño, lo que hace disminuir la existencia de posibles errores de desarrollo.

- Herramientas de simulación y creación de prototipo. Facilitan que tanto el equipo de trabajo, como, sobre todo, el usuario final del sistema puedan evaluar y entender el funcionamiento del futuro sistema antes de desarrollarlos.

- Herramientas de estructura que permiten la definición y gestión de las bases de datos y fuentes de información (herramientas de gestión documental, de contenido, etc.).

- Herramientas de desarrollo: editores, compiladores, depuradores, lenguajes de desarrollo, lenguajes de acceso y gestión de bases de datos, etc.

- Software ya existente, bien sean productos estándar, del mercado o librería de componentes desarrollados anteriormente, que se pueden reutilizar en el proyecto.

- Herramientas de prueba que facilitan no sólo la tarea de verificación tanto del producto final, como de los pasos intermedios, sino también la integración de los diferentes componentes del sistema informático futuro.

Entre los recursos de software es necesario tener en cuenta no sólo el producto final, sino también los recursos necesarios para desarrollarlos (herramientas de análisis y diseño, simulación, lenguajes de desarrollo, librerías existentes o herramientas de verificación).

La reutilización del software es muy importante en el desarrollo de un proyecto; el hecho de utilizar módulos o partes ya desarrollados por otros proyectos hace que el tiempo de proyecto sea más corto y se agudiza la fiabilidad, ya que son partes ya probadas por el usuario. Un alto nivel de reos indica experiencia, fiabilidad y menor coste.

Evaluación y estimación

Con el fin de poder establecer los recursos necesarios para llevar a cabo el proyecto, se ha de realizar una valoración, es decir, se debe estimar la "magnitud" del proyecto.

Para este proceso, se ha de disponer, por una parte, de información histórica de los anteriores proyectos similares realizados y, por otra, contar con la experiencia de los profesionales y expertos de la compañía.

Asimismo, hay un conjunto de indicadores y métricas de productividad del software que permiten dar apoyo a esta tarea:

- Líneas de código. Es uno de los indicadores más comunes a la hora de planificar el esfuerzo necesario.

- Puntos de función o funcionalidades que se han de desarrollar en el sistema informático.

- Etc.

Para realizar esta tarea, los analistas también disponen de un conjunto de herramientas en el mercado que implementan técnicas de estimación que aplican fórmulas matemáticas con parámetros obtenidos a partir de la información histórica.

Uno de los modelos más utilizados es el COCOMO (*constructive cost model*). Este modelo calcula el número de personas y el tiempo de gestión necesario para finalizar un proyecto a partir de:

- el tamaño del proyecto, es decir, líneas de código,
- el tipo de software que se ha de desarrollar,
- y la organización y experiencia del equipo de trabajo.

Gestión del coste

La estimación de los costes se realiza, por una parte, a partir de los requerimientos del proyecto y la información histórica, pero, sobre todo, a partir de la planificación de los recursos necesarios definidos (equipo de trabajo, recursos materiales, recursos de software y recursos de hardware).

Por ello es importante realizar el cálculo de los costes una vez evaluada la planificación y las necesidades de recursos dentro de la fase de definición inicial del proyecto.

Esta estimación debe ser una valoración cuantitativa del coste necesario con el fin de disponer de los recursos necesarios para llevar a cabo el proyecto en el tiempo previsto.

Durante todo el proyecto se deberá realizar un seguimiento del presupuesto inicial para verificar que se está cumpliendo y, en caso contrario, se deberá actualizarlo y tomar las medidas necesarias.

Este control del coste implica vigilar el gasto que se va produciendo con el fin de detectar variaciones sobre la estimación, registrar los cambios que se produzcan e informar a los implicados de los cambios autorizados.

Es necesario que la planificación del proyecto se evalúe y defina el presupuesto inicial con el fin de poder llevar a cabo un seguimiento durante la ejecución del proyecto, así como tomar las medidas necesarias en caso de desviación.

El coste de un proyecto tiene dos grandes partidas:

- Coste del hardware y coste del software.

- Coste de las horas de desarrollo.

El primero de los dos costes no acostumbra a tener grandes desviaciones; el segundo, en cambio, fluctúa mucho y es el que, si no se ha hecho una planificación esmerada, puede llevar a grandes desviaciones que hagan entrar en pérdidas al proyecto.

3.3.2. El equipo de trabajo

Tal como se ha detallado en el apartado de "Planificación y seguimiento", a continuación detallamos los principales roles que pueden participar en un proyecto de sistemas informáticos.

La pretensión es enumerar y describir la mayoría de los roles de los miembros que pueden participar en proyectos de estas características.

Con el fin de describir estos diferentes roles de los participantes de un equipo de proyectos, se agrupan en diferentes categorías o familias:

- Gestión del proyecto.
- Procesos y negocios.
- Marketing.
- Diseño gráfico y multimedia.
- Ingeniería.
- Control de la calidad.

Dependiendo de la tipología del proyecto, el equipo de trabajo estará formado por unos roles u otros.

En un proyecto una persona puede asumir más de un rol, por ejemplo, puede ser que un miembro del equipo adopte el papel de jefe de proyecto con una dedicación del 50% de su tiempo y la otra parte de su tiempo actúe como analista funcional, en la fase de diseño y como verificador, en la fase final de verificación; o puede ser, por ejemplo, que una misma persona asuma los papeles de analista funcional, en la primera fase, y de analista programador en la siguiente.

Analista programador

Persona que se encarga de analizar el problema que se desea resolver mediante la creación de una aplicación de software específica y que debe determinar la manera exacta como lo solucione este programa.

Ejemplo web

Por ejemplo, en un proyecto de implantación de una solución web, será necesario que en el equipo haya diseñadores gráficos y desarrolladores de webs, mientras que en un proyecto de implantación de una herramienta de gestión financiera se deberá contar con la participación de un experto en la parametrización de aquella herramienta y, seguramente, de un consultor de finanzas y contabilidad, entre otros.

Según la tipología y magnitud del proyecto, el equipo de trabajo estará formado por más o menos papeles, según las necesidades. Aun así, estos roles se pueden agrupar en seis tipos o familias.

Gestión del proyecto

Dentro de esta familia, se identifican entre otros, tres papeles:

- **Gerente del proyecto**. Asigna los recursos, establece las prioridades, coordina la interacción y mantiene al equipo focalizado en el objetivo correcto, y establece las prácticas que aseguran la integridad y calidad de los artefactos del proyecto.

- **Administrador del proyecto**. Lleva el control de las horas y del presupuesto del proyecto, elabora los estados de cuentas y la facturación, programa y controla el pago a proveedores y da apoyo a los miembros del equipo con los gastos de viaje, la entrada de horas, localización, etc.

- **Gestor del conocimiento**. Es el responsable de revisar la documentación que se genera en el proyecto, hacer cumplir los estándares de documentación y revisar que la documentación generada se adecue a los acuerdos definidos en el proyecto.

A menudo, estos tres roles se unifican con una sola persona bajo la denominación de jefe de proyecto o *project management*.

Procesos y negocios

Bajo esta familia de responsabilidad, se agrupan los papeles siguientes:

- **Consultor de negocios**. Realiza el análisis de la industria y de la empresa y la estrategia de negocio del cliente. Asegura la dirección adecuada de la estrategia de posicionamiento. Lidera el análisis de requerimientos, el diseño organizacional del cliente, su posterior formación y su apoyo operacional. Asegura la realización de todos los requerimientos por parte del equipo de desarrollo e implantación del proyecto.

- **Consultor de procesos**. Es el responsable de realizar el análisis de los procesos de negocio de cada una de las áreas operativas del cliente que serán afectadas por el proyecto (por ejemplo, la gestión de los pedidos, la aten-

ción posventa a los clientes, los procesos de facturación, etc.) detallando cada uno de los procesos del cliente en flujos de trabajo y elaborando el caso de negocio correspondiente con las funcionalidades descritas.

- **Consultor de catálogos**. Da apoyo a la selección y clasificación de los productos y servicios del cliente. Define los estándares técnicos y de presentación del contenido de catálogos. Da apoyo al proceso de integración de proveedores. Asegura la instalación y el acceso a las herramientas para la carga de catálogos en el ambiente de pruebas (verificación) y el de producción.

- **Líder de transacciones**. Define las reglas de negocio para las transacciones desarrollando los diagramas de flujo de procesos, generales y detallados por las transacciones de negocio.

Estos roles son adaptados por el analista funcional, con la ayuda de los expertos en la materia que pueda aportar la empresa que pide el proyecto.

Marketing

Dentro de este grupo se engloban todos aquellos profesionales responsables de considerar la gestión de la marca del cliente, el análisis del mercado, el tratamiento de la imagen corporativa, etc.:

- **Consultor de estrategia de marca**. Combina la estrategia del negocio de Internet con la gestión de marca para asegurar la creación de experiencias de marca relevante al público objetivo y desarrolla recomendaciones de estrategia de posicionamiento.

- **Consultor-analista de marketing**. Prepara los presupuestos, las previsiones y los informes de los procesos y actividades de marketing para el cliente. Realiza el análisis detallado de las actividades de mercado y contribuye al desarrollo de la estrategia de marketing del proyecto Internet, en caso de la implantación de un sitio de Internet.

- **Consultor de marketing**. Es el responsable de buscar oportunidades estratégicas en el mercado utilizando la información proporcionada por el cliente y cualquier otra fuente fiable; presenta, posteriormente, sugerencias para la planificación de la estrategia.

Diseño gráfico y multimedia

La parte gráfica de todo proyecto resulta cada vez más importante. Hubo un salto cualitativo de las aplicaciones *host* o servidor a las aplicaciones y programas Windows, pero el gran salto ha sido con el desarrollo de aplicaciones Internet y webs. Existe todo un conjunto de papeles asociados a estas tareas:

- **Experto funcional creativo**. Coordina la imagen del proyecto, ofrece soluciones y coordina al equipo creativo asesorando y guiando al cliente en las decisiones de imagen.

- **Experto funcional de arte**. Responsable de la dirección artística del proyecto, de establecer consistencia en el diseño y de proveer de experiencias e ideas creativas al resto de equipo.

- **Diseñador gráfico**. Responsable de los elementos gráficos

- **Desarrollador de la interfaz gráfica de usuario**. Programa los elementos que compondrán el proyecto, tanto si es una aplicación web como Windows, y optimiza estos elementos.

Ingeniería

El equipo de ingeniería o técnico es el equipo propiamente responsable del desarrollo de todo el software, la instalación y configuración del hardware y la integración con la parte de comunicaciones y redes:

- **Director técnico**. Comprueba que toda la documentación de requerimientos, análisis y diseño esté completa, correcta y firmada por el cliente. Dirige el desarrollo desde el inicio hasta la finalización con éxito. Coordina el equipo de desarrollo, pruebas, QA. Realiza el seguimiento y coordina las actividades relacionadas con proveedores tecnológicos.

- Obtiene requisitos técnicos del cliente para desarrollar la estrategia tecnológica. Asegura la escalabilidad de la solución y sirve como punto de contacto del cliente para cuestiones técnicas.

- **Administrador de contenido**. Desarrolla las estrategias y los procesos para la gestión de contenido. Adapta los servidores de gestión de contenido estándar del mercado a las necesidades del cliente y forma al cliente en el uso de estas herramientas.

- **Líder técnico de contenido**. Define el esquema y diseña el ciclo de trabajo de contenidos. Ayuda a la integración del servidor de gestión de contenidos con el resto del sitio.

- **Arquitecto de información**. Analiza a los usuarios y las tareas que deben llevar a cabo, diseña la arquitectura del programa y la interfaz de usuario, incluyendo la organización de la información y las reglas de navegación. Recoge los objetivos de negocio, los requisitos del usuario, el contenido y la funcionalidad. Organiza el contenido y desarrolla categorías, esquemas de nombres y jerarquías de navegación. Finalmente, crea y documenta casos de uso.

- **Desarrollador del *back-end***. Es el responsable de creación de los accesos a la información (bases de datos, páginas HTML, etc.).

- **Desarrollador de la lógica de negocio**. Construye los componentes de la capa lógica de negocios en una aplicación de tres capas.

- **Desarrollador *front-end***. Es el responsable de programar las páginas y componentes de interfaz de usuario.

- **Ingeniero de software**. Define y desarrolla las clases, los paquetes y los subsistemas que formarán parte del sistema informático.

- **Ingeniero de comunicaciones**. Evalúa los productos existentes y coordina la instalación, la integración, el desarrollo y la configuración de los productos seleccionados.

- **Diseñador de datos**. Participa con el arquitecto de software en la definición de las bases de datos de las aplicaciones y su interacción. Diseña la estructura de datos verificando y garantizando la integridad, la normalización, la optimización, la migración y la integración de datos.

- **Administrador de bases de datos (DBA**). Instala, genera y mantiene la base de datos.

- **Programador de bases de datos**. Crea los objetivos de bases de datos necesarios para la solución.

- **Arquitecto de red**. Diseña la arquitectura de red de los diferentes ambientes del proyecto (producción, calidad, verificación, desarrollo, etc.).

- **Administrador de sistemas**. Administra los sistemas operativos y aplicaciones entre los diferentes ambientes (producción, desarrollo, QA).

Todos estos roles se reparten en tres perfiles:

- Analistas técnicos.
- Programadores.
- DBA.

Aseguramiento de la calidad

- **Responsable de calidad**. Determina la carga de trabajo para el equipo de aseguramiento de la calidad según el tipo de proyecto. Genera el plan de pruebas con los requerimientos del proyecto en desarrollo y dirige la ejecución de las pruebas.

- **Verificador de código**. Revisa los estándares de codificación y el plan detallado de desarrollo del proyecto con el fin de verificar que el código de la aplicación cumpla los estándares antes de entregar el producto al cliente.

- **Verificador de seguridad y acceso**. Verifica la seguridad de la aplicación, lo cual incluye pruebas que midan la fiabilidad del acceso y la seguridad.

- **Verificador de funcionalidad**. Genera, de acuerdo con los escenarios que el responsable de calidad ha definido en el plan de pruebas, las secuencias (*scripts*) y los casos de prueba para identificar errores de funcionalidad y requisitos. Es el responsable de asegurar que la aplicación cumpla con la funcionalidad de acuerdo con los requisitos definidos por el cliente.

- **Verificador de "Look & Feel"**. Comprueba que el diseño gráfico de la solución cumple los requerimientos de Look & Feel antes de su entrega final al cliente.

El trabajo de test se hace entre:

- Jefe de proyecto.
- Analistas.
- Usuarios finales.

3.3.3. Diseño

El diseño es la primera fase de implementación de todo proyecto.

Una vez definidas las necesidades y los requerimientos del sistema informático e identificados los recursos necesarios y formado el equipo de trabajo, es necesario llevar a cabo el diseño del producto que se quiere desarrollar.

Tal como hemos comentado, el diseño parte de las necesidades del sistema y del modelo de interrelaciones de los elementos que deben intervenir.

El diseño es clave en todo proyecto, ya que un mal diseño conlleva errores en el desarrollo.

El primer factor que se ha de tener en cuenta en el diseño de todo sistema informático es la fragmentación del problema, es decir, definir los diferentes componentes que lo formarán, diseñarlos por separado y después diseñar la integración.

Este proceso se conoce como el diseño de la arquitectura.

Diseño de la arquitectura

Nombre que se utiliza para definir un conjunto de componentes hardware y software estandarizados con los que está diseñado un sistema informático: procesador, sistema operativo, conectores, etc.

De esta manera, el diseño se suele dividir en módulos que se comunicarán e integrarán entre ellos para formar el sistema final.

En la fase del diseño también se deben definir la infraestructura necesaria y la arquitectura de software que lo soportará (servidores de Internet y aplicaciones, bases de datos, sistemas de mensajería, sistemas operativos, etc.).

Otra parte importante en la fase de diseño es el diseño de datos. De este diseño dependen la organización, los métodos de acceso o las alternativas de procesamiento de los datos y la información que ha de gestionar el sistema informático.

Como resultado de este diseño, se definen las estructuras de datos.

Finalmente, una parte cada vez más importante en el diseño de soluciones informáticas es la interfaz de usuario, es decir, el diseño del aspecto "visual" de la aplicación. Para ello se han de tener en cuenta aspectos tanto de diseño gráfico, como de usabilidad y navegación.

En la fase de diseño de un proyecto informático se deben definir tanto la arquitectura del sistema y las infraestructuras necesarias, como los módulos y funcionalidades que se han de desarrollar e integrar, pero también las estructuras de datos y el diseño gráfico, usabilidad y navegación de la solución.

Asimismo, es necesario definir, en esta fase, las pruebas de verificación que se llevarán a cabo durante el desarrollo, la integración y la implantación de la solución.

Para llevar a cabo estas definiciones y estos diseños, los analistas informáticos cuentan con diferentes métodos y herramientas.

El objetivo de utilizar un método para el diseño es garantizar la obtención de un software comprensible en el que se puedan detectar los errores con facilidad y que permita el mantenimiento y la extensión posteriores.

Para diseñar el software, conviene disponer de un buen sistema de notación que permita entender el diseño de una manera simple, más fácilmente que leyendo el código de un lenguaje de programación.

Las notaciones ayudan a obtener un detalle de las funcionalidades del sistema que se ha de desarrollar de la manera más clara y comprensible.

Algunas de las principales notaciones que se utilizan son los niveles de abstracción, los diagramas de flujo, las tablas de decisión y los casos de uso.

El objetivo de este material no es profundizar ni detallar estas técnicas, métodos y notaciones, sino sólo destacar la importancia y el uso en el diseño de los sistemas informáticos.

Para poder diseñar, de manera comprensible, los sistemas informáticos y las funcionalidades que se han de desarrollar se utilizan diferentes métodos, herramientas y notaciones.

3.3.4. Desarrollo e integración

La fase de desarrollo empieza después de la aprobación formal de los requisitos del sistema por parte del usuario o cliente y de las fases de diseño de alto nivel (análisis funcional) y diseño detallado (análisis orgánico).

Análisis funcional

Nombre con el que se conoce la primera fase del desarrollo de una aplicación que consiste en reunir todos los datos disponibles del problema que se quiere solucionar y crear un diagrama de flujo o un pseudocódigo con los elementos generales del programa que se creará para solucionarlo.

Análisis orgánico

Fase del diseño de una aplicación en la que se convierte el algoritmo escrito en forma de pseudocódigo o diagrama de flujo en código que pueda entender el compilador, intérprete o ensamblador correspondiente.

A partir de la arquitectura definida del software y del sistema, se han de desarrollar las diferentes funciones necesarias y los componentes de software e implementar los flujos de datos y de control entre estos componentes.

La arquitectura de software se descompondrá en unidades de menor complejidad y a menudo se adopta una metodología de tipo *top-down*.

Para cada elemento o componente del software se habrán indicado, durante la fase de diseño, los datos de entrada, las funciones que realizará y los datos de salida.

Como resultado de la fase de desarrollo, o programación, se obtiene, aparte del código del programa, el documento de diseño detallado y el manual de usuario del software.

Todo módulo desarrollado será objeto, posteriormente, de un test de integración.

Para saber más

Para tener más información sobre los tests de integración, podéis consultar el apartado "Verificación y pruebas" de este mismo módulo.

El software codificado y validado está preparado para entrar en la etapa de transferencia, que se instala sobre la plataforma (hardware) final con el fin de poder llevar a cabo los tests de aceptación especificados.

La fase de desarrollo e integración es de las más complejas, tal como ya se ha visto, por la diversidad de roles del equipo de trabajo que puede participar en ella.

Para saber más

Para tener más información sobre la diversidad de roles del equipo podéis consultar el apartado "El equipo de trabajo" de este mismo módulo.

El objetivo de este material no es describir ni profundizar en el desarrollo de aplicaciones, sino simplemente indicar su importancia dentro del proyecto.

Para saber más

Para tener más información sobre el desarrollo de aplicaciones podéis consultar la unidad "Desarrollo de software" de este mismo módulo.

3.3.5. Verificación y pruebas

En el proceso de verificación y pruebas, además de asegurar que el producto o sistema cumple correctamente las funciones que el cliente ha solicitado, se ha de comprobar que los requerimientos establecidos son los adecuados.

Por este motivo, en esta etapa se debe definir una estrategia de pruebas que ha de considerar:

- la planificación de la prueba,
- el diseño de casos de prueba,
- la ejecución de estas pruebas,
- y la evaluación de los resultados.

La verificación de un sistema informático tiene en cuenta tanto las pruebas de software, como posteriormente las pruebas del sistema.

Existe, en el mercado, además, un conjunto de herramientas y aplicaciones que facilitan la tarea del equipo de pruebas.

Pruebas del software

A la hora de verificar y buscar posibles errores en el software desarrollado hay dos técnicas de verificación:

- **Pruebas de caja blanca**: comprueban que los elementos del software encajan correctamente y que los procedimientos internos cumplen las especificaciones. Se centran en la estructura de control del programa, e intentan buscar errores en el código. Se ponen a prueba los caminos independientes de cada módulo, las iteraciones, las estructuras internas de datos y las decisiones lógicas.
- **Pruebas de caja negra**: comprueban que las funciones que se ejecutan en el software son operativas y sirven para conseguir la función por la que han

sido desarrolladas. Confirman que el software responde bien a las entradas, tanto válidas como no, que las salidas son correctas, y que las funciones son operativas. Intentan encontrar errores, tanto errores funcionales de partes que no cumplen con las especificaciones iniciales, como errores en funciones y estructuras de datos incorrectos, errores en el acceso a bases de datos externas, en la interfaz, de rendimiento, etc.

Con el fin de simplificar y garantizar el proceso de pruebas, se inician en los niveles más básicos, los módulos. Una vez verificado el funcionamiento correcto de los módulos, se comprueba el funcionamiento conjunto de los diferentes módulos que componen una parte del sistema y, finalmente, se verifica el correcto funcionamiento de todo el sistema. De esta manera, las pruebas se van llevando a cabo a partir de la integración de los módulos y partes del sistema.

De esta manera, la localización de los errores es más fácil.

Los pasos que se siguen en el proceso de prueba del software son los siguientes:

- Prueba de las funciones y módulos. Se utilizan pruebas de caja blanca, examinando, entre otros elementos, la interfaz del módulo, las estructuras de datos locales y sus funcionalidades. Las llevan a cabo los miembros del equipo de desarrollo.

- Pruebas de integración. En estas pruebas se utilizan las técnicas de caja blanca y caja negra. Se comprueba que cada módulo funciona bien con los demás garantizando que el conjunto mantiene las funcionalidades previstas. Las llevan a cabo los miembros del equipo de desarrollo, pero pueden intervenir los usuarios y otras personas expertas en las funcionalidades pedidas a las especificaciones.

Una vez realizadas las pruebas de funcionalidad e integración del software, se realizan dos pruebas más: la prueba alfa y la prueba beta.

- La prueba alfa: el usuario final del software lo prueba en el mismo lugar donde se ha desarrollado, en un entorno, por lo tanto, controlado.

- La prueba beta: el usuario prueba el software en el lugar donde será utilizado.

Durante todo el proceso de testeo y pruebas, se van detectando errores que se tienen que corregir y volver a probar; es un proceso con un alto contenido de realimentación y muy delicado, ya que soluciones a errores pueden producir errores en funciones ya probadas con éxito y provocar el desconcierto y desencanto del futuro propietario del software.

Beta tester

Término inglés que se traduciría por "verificador de beta". Se utiliza para referirse a aquellas personas que se encargan de probar el funcionamiento de un programa que todavía no ha salido al mercado (se encuentra en versión beta) con el objetivo de encontrar errores de funcionamiento. De estos errores se informa a la empresa autora del programa para que los corrija con el fin de que la versión definitiva sea tan perfecta como sea posible.

Con el fin de verificar el correcto desarrollo y la funcionalidad del software hay dos técnicas: la caja blanca y la caja negra.

El proceso de verificación también se conoce, en argot informático, depurar.

Depurar

Palabra utilizada por los programadores para referirse a la acción de revisar un programa en desarrollo con el objetivo de encontrar y eliminar posibles errores de funcionamiento que pueda haber (conocidos por *bugs*). La depuración también persigue la optimización del programa para que su funcionalidad y velocidad sean lo mejores posibles. La depuración normalmente se realiza con la ayuda de programas especializados en esta tarea, lo que facilita el trabajo del programador y acorta el tiempo utilizado en la fase de depuración. Para evitar los errores en el funcionamiento de los programas, se suelen lanzar varias versiones beta de cada programa que prueban los especialistas.

Pruebas del sistema

Finalmente, una vez verificado el desarrollo correcto del software, se deben realizar las pruebas del sistema.

En este momento, el software se integra con el resto del sistema (hardware, servidores, etc.).

En estas pruebas participan los responsables de desarrollo de cada elemento y parte del sistema.

Las principales pruebas del sistema que se llevan a cabo son las siguientes:

- **Pruebas de recuperación**. Evalúan si la recuperación de errores es la apropiada. Es decir, se comprueba si la reinicialización, los mecanismos de recuperación de estado del sistema, la recuperación de datos, etc. son los adecuados.

- **Pruebas de seguridad**. Tratan de garantizar que el sistema se encuentra protegido contra robos de clave de acceso, ataques con software, bloqueos del sistema, errores provocados por intentos de ataque al sistema, etc.

- **Pruebas de resistencia**. Intentan garantizar que el sistema tiene unos límites correctos para aguantar en condiciones extremas, por ejemplo, el acceso concurrente de muchos usuarios al sitio web, la disponibilidad de memoria suficiente en el caso de ejecución de muchos procesos o que el ancho de banda de la Red permite garantizar el acceso y el trabajo correctos de los usuarios.

Las pruebas del sistema se dividen en recuperación, seguridad y resistencia.

Herramientas de apoyo

Con el fin de que el proceso de verificación sea lo más rápido posible, hay un conjunto de herramientas en el mercado que permiten disminuir el esfuerzo y el coste de esta fase.

Estos programas se pueden clasificar en función del tipo de ayuda que proporcionan:

- **Generadores de datos de prueba**. Producen datos que hacen que los programas y aplicaciones que se deben verificar se comporten de una manera determinada.

- **Comparador de archivos**. Trabajan comparando diferentes conjuntos de datos de salidas de distintas pruebas.

- **Simuladores**. Permiten imitar el entorno real del software y el sistema informático que se está verificando.

Hay distintas herramientas en el mercado para ayudar al equipo de pruebas y aseguramiento de la calidad del sistema. Estas herramientas se pueden clasificar en generadores de datos de prueba, comparadores de archivo y simuladores.

3.3.6. Operación y mantenimiento

Muchos equipos de trabajo tienden a creer que su misión ha acabado cuando el software, y el sistema informático en general, está instalado y funcionando en las instalaciones del cliente. Pero en la práctica pocas veces es así.

Las peculiaridades de un sistema informático, vista la complejidad de todos los elementos que intervienen en él (podéis ver "Proyectos informáticos"), provocan que se deba supervisar el funcionamiento correcto durante mucho tiempo después de haber sido entregado.

Un requisito de disponibilidad o de fiabilidad del sistema, por ejemplo, es muy difícil garantizar que el programa se comportará adecuadamente en circunstancias no previstas o transcurrido un tiempo determinado, por lo que los tests de aceptación suelen incluir, a petición del cliente, pruebas a largo plazo del software.

Esta fase de supervisión (y corrección de los vicios o defectos del producto) recibe el nombre de *fase de operación*.

Sólo después de que haya transcurrido este intervalo, el cliente acepta definitivamente el software, provisionalmente aceptado al acabar la fase de transferencia.

Al cabo del tiempo, el software definitivamente aceptado se debe poder cambiar, bien como consecuencia de errores no detectados, bien como respuesta ante nuevas necesidades del usuario.

Esta fase siguiente recibe el nombre de *fase de mantenimiento*.

Durante la fase de operación y mantenimiento se genera y actualiza el documento de historia del proyecto, que recoge todos los errores y todas las modificaciones realizadas sobre el software y permite calcular y analizar la fiabilidad del software y el rendimiento del equipo de trabajo (para futuros proyectos).

Una vez implantado el sistema, se ha de iniciar una fase de supervisión, llamada *de operación*, durante las primeras semanas o meses, e iniciar posteriormente la fase de mantenimiento si se deben efectuar modificaciones o adaptaciones a la solución.

3.3.7. Documentación

Durante el ciclo de vida de cualquier proyecto se genera mucha documentación relativa, de tipo técnico, de gestión o documentación complementaria.

La gestión correcta de toda esta documentación es vital, no sólo para el seguimiento del proyecto, sino también para garantizar una información histórica, tanto para llevar a cabo futuros cambios y el mantenimiento, como para contar con información en la que basarse en nuevos proyectos similares.

En proyectos complejos, de larga duración o donde intervengan equipos numerosos de personas (o diferentes empresas), su gestión evita duplicar innecesariamente los documentos generados y permite que los integrantes del equipo de trabajo sepan qué documentación existe, cuál es la última versión, dónde se encuentra localizada, etc.

La gestión de la documentación de un proyecto supone la realización de las tareas siguientes:

- redacción de los procedimientos de gestión de documentación y asignación de referencias a documentos,

- asignación de referencias a los documentos, en la medida en que se generan,

- inclusión de los nuevos documentos en la base de datos (listado) de documentación del proyecto,

- difusión en el equipo de trabajo,

- incorporación de la base de datos del proyecto a la base de datos general de la empresa al finalizar el proyecto.

Algunos de los tipos más frecuentes de documentos de un proyecto son:

- Documentación técnica. Requerimientos técnicos y especificaciones, documentos de diseño, notas técnicas, planos, diagramas, listado de programas, manuales técnicos, de instalación de usuario, etc.

- Documentos de gestión. Presupuestos y ofertas, minutas y acciones, informes de situación, balance de horas y gastos, etc.

- Documentos complementarios. Listas de documentos, listas de productos, presentaciones, planes de calidad, procedimientos, faxes, comunicaciones internas, correos electrónicos, etc.

La gestión de la documentación, que en pequeños proyectos suele ser parte de la tarea del gestor del proyecto, se puede delegar en otras personas en proyectos complejos o en los que el volumen de la documentación generada lo aconseje.

Una vez cerrado el proyecto, la documentación se clasificará en reutilizable y no reutilizable.

La documentación reutilizable se incorporará inmediatamente al archivo de documentación de la empresa, mientras que la no reutilizable se puede destruir una vez transcurrido un período prudencial de tiempo (que incluya garantías, servicios posventa o períodos de posible contratación de una continuación del proyecto).

La gestión de la documentación de un proyecto supone la realización de un conjunto de tareas, como son la redacción de los procedimientos de gestión de documentación y la asignación de referencias, la inclusión de nuevos documentos en la base de datos de documentación del proyecto, su difusión entre el equipo de trabajo y la incorporación de la base de datos del proyecto en la base de datos general de la empresa, cuando éste finalice.

En todo proyecto tienen un interés especial las ayudas automáticas de las que pueda disponer el equipo, entendiendo por esto los programas informáticos destinados a facilitar el trabajo de ayudar a construir software.

Tal como ya hemos comentado, estas ayudas pueden facilitar una actividad de análisis, diseño o implementación propias del procedimiento recomendado por la metodología.

También pueden ayudar a gestionar la organización del proyecto, en particular a controlar el proceso de avance y realización, así como facilitar la tarea de elaborar la documentación exigida por la metodología.

4. Licencias de software

La compra de una licencia de un producto o aplicación representa para el comprador el derecho de instalar y utilizar un programa.

Por cada aplicación de software que se utiliza, el fabricante otorga una licencia de uso que se documenta en el contrato de licencia de usuario.

El software está protegido por la ley de derechos de autor, que establece que un producto no se puede copiar sin la autorización del propietario de los derechos de autor.

Una licencia de software aplica a cualquier tipo de programa (podéis ver el apartado "Aplicaciones del software"): sistemas operativos, procesador de textos, antivirus, servidor de correo, gestor de proyectos, etc.

Estas licencias se pueden clasificar, de manera simplificada, en tres tipos:

- personal,
- servidora,
- de cliente.

Una licencia de software personal implica el derecho de instalar un programa en un ordenador "personal". Por ejemplo, la licencia de uso de un procesador de textos o de un gestor de hojas de cálculo.

En ocasiones, la compra de un software personal incluye, en la licencia, la posibilidad de instalarlo en varios ordenadores personales (2-3). Esta política es muy seguida por los antivirus, de manera que al comprar uno lo puedes instalar en tu ordenador personal, en el portátil y en el ordenador de la habitación del hijo, por ejemplo.

En cambio, una licencia servidora corresponde al derecho de instalar y poder utilizar un programa que atenderá "peticiones" de otros programas "clientes". Ejemplos de licencias servidoras son las correspondientes a los servidores de bases de datos, servidores de correo, servidor de acceso a Internet, etc.

Finalmente, a una licencia "cliente" corresponde el derecho de que un ordenador acceda y trabaje con un programa que se ha instalado como servidor. De este modo, para cada ordenador o usuario que acceda al servidor de bases de datos o al servidor de correo, se debe adquirir una licencia cliente correspondiente.

Los clientes potenciales de las licencias servidoras y clientes son empresas y grandes corporaciones. La obtención de licencias servidoras y clientes van muy de la mano; a menudo, se compra una o dos licencias servidoras y tantas licencias clientes como ordenadores esté previsto que accedan a los datos del servidor. Por ejemplo, si se tratara de un programa gestor de correo electrónico, se tendría que comprar una licencia servidora y tantas licencias cliente como cuentas de correo se quieran tener.

El uso ilegal de un programa supone un delito y, como tal, está penalizado.

Hay varias organizaciones, nacionales e internacionales, encargadas de perseguir la piratería informática en las compañías y los organismos.

Las licencias se pueden clasificar de manera simplificada en personal, de servidor y de cliente.

4.1. Políticas de compra y uso de licencias

Cada fabricante tiene una política de venta de licencias diferente y particular.

Estas políticas, además, evolucionan con el tiempo y se adaptan a las nuevas características de las tecnologías de la información (TI).

Así, la aparición de aplicaciones Internet supuso la aparición no sólo de un canal nuevo de venta y distribución, sino también de nuevas políticas de compra y uso de licencias como, por ejemplo, las aplicaciones Web.

Aplicaciones web

Una aplicación web está instalada en un proveedor de Internet, de manera que el cliente no es propietario de esta aplicación, sino que alquila su uso para llevar a cabo algunos procesos que requieren esta solución, de manera análoga al alquiler de un coche para hacer un viaje.

Algunos ejemplos son los siguientes:

- Microsoft office en línea: se pueden crear, editar y guardar documentos de MS Office (Excel, Word, Access, PowerPoint, etc.) sin tener instalado el MS Office. Para hacerlo, hay que registrarse en la web que MS tiene dedicada a este servicio.
- Programas convertidores de archivos: se pueden encontrar en la Red sitios web dedicados a la conversión de archivos libres de tasas, por ejemplo, para convertir documentos de Word o imágenes en pdf.
- Antivirus en línea: la mayoría de empresas del sector tienen una parte de su web dedicada a la detección de virus en línea. De esta manera, desde allí, y sin necesidad de comprar una licencia de antivirus, hay la posibilidad de escanear y detectar virus en vivo. La mayoría de estos servicios, sin embargo, sólo te informan de la existencia o no del virus en tu PC; para eliminarlo, hay que comprar la licencia.

Una vez decidida la necesidad de adquirir un programa, es necesario seguir tres pasos básicos para comprar las licencias de software:

a) Determinar la licencia necesaria (tipo de software y cantidad).

b) Decidir la mejor opción entre comprar o *leasing*.

c) Determinar dónde realizar la compra.

Para determinar la licencia necesaria debemos tener en cuenta los datos o categorías siguientes:

- **Tipo de producto**. Algunos fabricantes clasifican sus programas según estos tres tipos:
  - **Aplicaciones**. Programas genéricos de productividad personal, utilidades o herramientas de desarrollo.
  - **Sistemas**. Corresponden a los sistemas operativos.
  - **Servidores**. Enmarca todas las aplicaciones corporativas o servidores corporativos, como por ejemplo, los servidores de bases de datos, de Internet o de correo.
- **El producto**. El programa en sí, según las funcionalidades requeridas. Por ejemplo, un procesador de textos, un antivirus, etc.
- **Versión**. Un mismo producto puede contar con diferentes versiones, tanto por la evolución de los sistemas operativos en los que se pueden instalar y ejecutar, como por disponer de más o menos funcionalidades y prestaciones.
- **Edición**. Especifica el conjunto de prestaciones y/o aplicaciones incluidas en el producto.
  Un producto puede disponer, por ejemplo, de las ediciones estándar, profesional y experto.
- **Tipo de licencia.** En función del fabricante pueden existir diferentes tipos como, por ejemplo, la licencia de nuevo usuario, correspondiente a la compra inicial del producto, la actualización de versión, que permite la adquisición de una versión posterior del programa si se dispone de la licencia de la versión anterior del mismo producto, o la actualización competitiva, que corresponde a la compra de una licencia de "coste menor" de un programa, si se dispone de una licencia de un producto relacionado de otro fabricante.

Aun así, estos son ejemplos genéricos de tipos de licencias, ya que cada fabricante posee su política específica de compra y uso de sus programas.

Algunos fabricantes tienen una política de venta relacionada con el uso de la herramienta por parte del cliente, cobrando, por ejemplo, una comisión por cada transacción realizada (plataformas de eCommerce) utilizando la aplicación. Este último tipo de uso de licencias suele corresponder a sistemas de comercio electrónico donde el comprador abona un importe base (coste fijo) para la herramienta y una comisión (coste variable) para cada transacción que se realiza mediante ésta.

Hay opciones muy económicas que permiten a las empresas disfrutar de software de alta calidad sin necesidad de comprar el software ni el hardware. Así sólo se paga por la licencia; tanto el software como el hardware están físicamente en casa del fabricante; lo que se hace es reservar una parte de los recursos del hardware y configurar el software según las necesidades del cliente. Suelen ser aplicaciones web, de manera que se puede acceder con un navegador desde la empresa y hace el mismo uso como si fuera una aplicación instalada en tu propio hardware. Un ejemplo de este caso es el Siebel, una herramienta CRM muy potente y con un coste económico muy grande que, en su versión Siebel *on demand*, funciona como se ha descrito y cualquier pyme puede acceder a Siebel, una herramienta CRM, hasta ahora reservada a corporaciones con un gran volumen de negocio.

Hay diferentes opciones de compra del software:

- **OEM (*original equipment manufacturas*).** Corresponde al software incluido en la compra de un ordenador o un equipo informático nuevos. El caso más común es cuando, al comprar un ordenador personal, el fabricante incluye la licencia del sistema operativo y un conjunto de aplicaciones de productividad personal: procesador de textos, hoja de cálculo, herramienta de dibujo, etc.
  El software OEM es una versión especial del software que se debe distribuir de manera preinstalada en el disco duro del PC.
  Nunca se puede distribuir software por OEM sin el PC o el hardware correspondiente.

- **Compra del paquete.** Corresponde a la compra física de la caja con el producto en cualquier tienda o almacén.
  Incluye el producto en formato magnético, por ejemplo, CD o DVD junto a las licencias correspondientes.

- **Licencia por volumen para organizaciones.** Este tipo de compra corresponde a las realizadas por las compañías que necesitan una gran cantidad de licencias del mismo software.

Ejemplos

Por ejemplo, una organización de 200 trabajadores puede necesitar, por ejemplo, adquirir 110 licencias de un procesador de textos. En estos casos, en lugar de adquirir los 110 CD o DVD del producto, se compran dos o tres CD y las 110 licencias (el derecho de instalar el producto en 110 máquinas). Esto reduce el coste del producto y los costes de

envío. La mayoría de los fabricantes aplican diferentes políticas de precio dependiendo del número de licencias adquiridas o del tipo de organización (administración pública, instituciones académicas, etc.).

Por ejemplo, Microsoft distingue entre la venta a pequeñas y medianas empresas (pymes) con la Open Multilicencia (adquisición de licencia perpetua) o la Open Suscripción (licencias no perpetuas) y la venta a grandes empresas, con el Contrato Select o Enterprise Agreement Suscription.

Una vez decidida la necesidad de adquirir un programa, se debe determinar la licencia necesaria (tipo de software y cantidad), decidir la mejor opción entre comprar o *leasing* y determinar dónde se debe realizar la compra.

4.2. Licencias de software gratuito (*freeware*) y software de prueba (*shareware*)

En los apartados anteriores se ha introducido el concepto de software y sus aplicaciones.

Una solución informática es un "programa" que se instala en un ordenador o hardware para llevar a cabo ciertas tareas, ya sea para elaborar y editar documentos, gestionar bases de datos, ofrecer conectividad entre ordenadores o para gestionar proyectos.

Utilizar estos programas implica tener una o más licencias.

Se han comentado diferentes políticas de compra y uso de licencias de software.

Ahora bien, hay dos "tipos de licencias" no consideradas en los apartados anteriores y que cada vez tienen más importancia y resultan más usuales: las licencias de software gratuito (*freeware*) y las licencias de software de prueba (*shareware*).

Según la disponibilidad de los archivos fuente, el software se puede clasificar en abierto (libre, de dominio público y semilibre) y cerrado (gratuito, de prueba, de demostración y propietario).

Según el coste que representa para el usuario, el software se puede clasificar en gratuito (libre, de dominio público, semilibre y gratuito) y en software no gratuito (de prueba, de demostración y propietario).

- El software **gratuito** se puede utilizar, copiar y distribuir libremente, pero no modificar, ya que no incluye el código fuente. Para la FSF (Free Software Foundation), entidad que promueve el uso y desarrollo de software de este tipo, el software gratuito no es libre, aunque tampoco lo califica como

semilibre ni como propietario. Es un software que no es necesario comprar para utilizarlo.

- El software **de prueba** se caracteriza por que es de libre distribución o copia, de manera que se puede utilizar, si se cuenta con el permiso del autor, durante un período limitado de tiempo, y se debe pagar, transcurrido éste, para continuar utilizándolo, aunque la obligación es sólo de tipo moral, ya que los autores entregan los programas confiando en la honestidad de los usuarios. Este software se suele distribuir sin el código fuente y no se permite modificarlo ni redistribuirlo. Muchas veces, por ignorancia, los programas de esta clase se suelen utilizar de manera ilegal. A menudo se confunde el software de prueba con el software de demostración, que son programas no funcionales al cien por cien o que dejan de funcionar después de un tiempo determinado.

Según la disponibilidad de los archivos fuente, el software se puede clasificar en abierto (libre, de dominio público y semilibre) y cerrado (gratuito, de prueba, de demostración y propietario); y según el coste que representa para el usuario, se puede clasificar en gratuito (libre, de dominio público, semilibre y gratuito) y en software no gratuito (de prueba, de demostración y propietario).

Hay casos cuyo régimen está a caballo entre los mencionados, en los que se permite el uso del software hasta el momento en que se usa por tareas lucrativas; entonces, se tiene que pagar la licencia. Es decir, son *freeware* hasta el momento en que se usa para sacar un rendimiento económico, y luego hay que hacer el pago de la licencia. Por ejemplo, la base de datos Oracle.

No debemos confundir, sin embargo, las licencias de software gratuito con el código fuente abierto u *open source*.

Ejemplos de aplicación gratuita

Ejemplos de aplicaciones gratuitas son los siguientes:

- Paquetes ofimáticos, con procesador de textos, hojas de cálculo, etc.: OpenOffice.org, Star Office y los gestores de bases de datos DBMaker 4.03 y TreeDBNotes Free.
- Sistemas operativos: OPTIX IV, BeOs y Linux MandrakeSoft.
- Gestores de correo electrónico: FoxMail 4, Desktop Sidebar y Cactus Mail.
- Programas de retoque fotográfico: PhotoPlus y Photo Librarian Image Editor.
- Programas para realizar copias de seguridad: WinDriversBackup y InFoAL BackUp.

Open source

Nombre que se da al software que se puede distribuir libremente con la única condición de que siempre se dé el código fuente junto al ejecutable. En caso de que el código contenga errores de programación, el usuario los puede corregir. Según cuál sea la licencia de código fuente abierto, el usuario también tiene el permiso de distribuir la versión del programa que haya modificado. Los sistemas operativos Linux o la *suite* ofimática OpenOffice son el paradigma del software *open source*.

Ejemplos de aplicación de prueba

Ejemplos de aplicaciones de prueba son los siguientes:

- Gestores de bases de datos: SQLConnect 1.6.2.
- Programas de diseño gráfico: 3D Flash Animator, Paint Shop Pro e IntelliCAD 2001 Standard.
- Editores de páginas web: Ace Expert, CoffeeCup HTML Editor y Web Design.
- Antivirus: Anti-Trojan y MC Afee VirusScan 14.4.0.
- Gestor de correo electrónico: Calypso.
- Programas de gestión de copias de seguridad: mCopias, Magellan y Backup BaK Again II Workstation.

4.3. La piratería de software

El término *piratería de software* cubre diferentes actividades, desde copiar programas ilegalmente, hasta falsificar y distribuir software sin el consentimiento o acuerdo con el fabricante correspondiente.

4.3.1. Consecuencias de la piratería de software

La mayoría de las personas creen que la piratería de software sólo afecta a la industria informática.

En cambio, se ha de tener en cuenta que, actualmente (12/05/2009), el 42% del software que se utiliza en España está copiado ilegalmente. A escala mundial, el porcentaje es del 41%. Con estos datos, se puede dimensionar el alcance real del problema.

Según la BSA (Bussiness Software Alliance), la piratería informática causa estragos no sólo en la industria del sector, sino también en la sociedad; a los millones de euros en pérdidas del sector hay que añadir los millares de puestos de trabajo que se pierden o se dejan de generar debido a la piratería. Se pueden encontrar muchos informes de actualidad referidos a la piratería informática, desde el punto de vista de de la industria del software en la web de la BSA (indicada arriba).

Un estudio sobre piratería de software realizado por International Planning & Research Corp., también durante el año 2000, señala que estas acciones ilegales representaron la pérdida de 118.026 puestos de trabajo en Estados Unidos.

La utilización ilegal del programa también supone otros gastos y costes a las organizaciones que lo utilizan.

Ejemplo de costes derivados

Ejemplos de estos costes derivados son los siguientes:

- la falta de apoyo técnico, por parte del fabricante,
- el acceso a nuevas versiones, actualizaciones, adaptaciones y resolución de errores (garantía posventa),
- el acceso a programas completos con todas las funcionalidades,
- disponer de programas libres de virus.

Asimismo, la violación del Código Penal expone a las compañías a varias acciones judiciales y al desprestigio y la pérdida de su imagen.

Resumiendo, se podrían clasificar estas consecuencias según tres tipos de destinatarios: los consumidores, los desarrolladores y los vendedores.

Cuando un usuario decide hacer una copia no autorizada de un programa, está renunciando al derecho a la asistencia, documentación, garantías y actualizaciones periódicas. Si la copia ilegal del software se obtiene por medio de programas P2P, el riesgo de que el fichero esté infectado por algún virus es muy alto.

P2P o *peer 2 peer*

Son programas que ponen en contacto dos ordenadores remotos y se pueden compartir ficheros. Son programas P2P el Emule o el BitTorrent, por citar dos de los más populares.

Al piratear un producto protegido por las leyes de propiedad intelectual, el individuo se expone al riesgo legal que esto conlleva.

La pérdida de ingresos que implica la piratería se habría podido invertir en el producto y conseguir, así, reducir el precio final para el consumidor. Con estas acciones ilegales se perjudican muchas y variadas empresas que tienen su medio de subsistencia en el diseño y desarrollo de software y en su distribución y venta.

La piratería de software no sólo afecta a la industria informática, sino también a los consumidores, desarrolladores y vendedores.

4.3.2. Formas de piratería

Habitualmente, cuando se piensa y se habla de piratería de software se identifica con las copias. Aunque ésta es una de las formas más extendidas de piratería, hay muchas otras:

- **Copias realizadas por el usuario final**. Son simples copias sin licencia realizadas por usuarios individuales o empresas. También se pueden considerar, en el caso de la compra de licencias por volumen, una información incorrecta sobre el número de ordenadores en los cuales se ha instalado el software.

- **Carga en el disco duro**. Tal como hemos comentado en el apartado anterior, a menudo los fabricantes de ordenadores tienen contratas con los fabricantes de software que les permiten distribuir software por OEM (soft-

ware que se debe distribuir de manera preinstalada en el disco duro del PC). Un tipo muy habitual de piratería lo llevan a cabo fabricantes de sistemas de PC que comercializan los ordenadores con software preinstalado sin que este software sea legal. Es decir, sin haber firmado un contrato con el fabricante del software para distribuirlo.

- **Falsificaciones**. Es la piratería de software a gran escala, en la que se duplica ilegalmente el software y sus cajas, llevado a cabo por redes organizadas que distribuyen los productos como supuestamente legales.

- **Distribución por canales fraudulentos**. Muchos fabricantes de software distribuyen licencias a colectivos determinados como, por ejemplo, el colectivo educativo, con unos descuentos especiales. Otras veces, por compras de gran volumen, los fabricantes ofrecen unos descuentos al comprador (grandes empresas, administraciones públicas). Una casuística de piratería corresponde al software distribuido como licencias con un descuento especial, a clientes con un gran volumen, a fabricantes de ordenadores o a entidades académicas, que se redistribuye, posteriormente, a otros usuarios que no cumplen los requisitos para disponer de estas licencias.

- **Piratería en Internet**. En este caso, corresponde más al medio que al tipo. Se utiliza Internet para la distribución ilegal del software, de manera que se puede acceder a copias de los diferentes programas sin adquirirlos al fabricante o a los distribuidores autorizados. Algunos informes de la BSA hablan de que dos de los grandes canales de distribución de falsificaciones y copias ilegales de software son los programas P2P y el portal de subastas Ebay. El tráfico de datos en Internet debido a los programas P2P es tan alto que, según fuentes de la BSA, en algunos países llega a situarse entre el 49%-89% del tráfico total diurno en Internet, y la cifra se eleva hasta el 95% de noche.

Otros ejemplos

Otras modos de piratería pueden ser, por ejemplo, instalar un producto original en varios ordenadores sin haber adquirido el número de licencias necesario o instalar en el ordenador personal un software legal pero adquirido para utilizarlo en el entorno laboral.

La piratería de software va más allá del simple hecho de copiarlo ilegalmente. Hay otras maneras de piratear como la carga ilegal en el disco duro, las falsificaciones o el uso de más licencias que las adquiridas legalmente.

Ejercicios de autoevaluación

1. Clasificad los programas y las aplicaciones siguientes siguiendo la taxonomía presentada en el apartado "Aplicaciones del software".

Aplicación para el control de los pedidos de los clientes de una empresa.

- Simulador de una constelación y/o del movimiento de los planetas del sistema solar.
- Sistema de gestión de proyectos.
- Programa de reconocimiento de la voz.
- Sistema operativo.
- Programa de interfaz de comunicaciones entre un ordenador y un dispositivo móvil.
- Herramienta de apoyo del control aéreo.
- Sistema de visión artificial.
- Aplicación de agenda.
- Herramienta de apoyo al diseño de piezas.
- Geográfico y seguimiento de un paquete.
- Analizador de los componentes de una solución química.
- Sistema de control de los semáforos de una ciudad.
- Herramienta de posicionamiento.

2. Ordenad, cronológicamente, los lenguajes presentados en el apartado "Principales lenguajes de desarrollo": Pascal, Eiffel, Cobol, Simula, C, Fortran, Basic, C++, Lisp, Algol, RPG, Modula2, Ada, Prolog.

3. Clasificad los lenguajes presentados en las cuatro categorías siguientes:

Lenguajes de aplicación en inteligencia artificial	Lenguajes de aplicaciones científicas	Lenguajes orientados a objetos	Lenguajes de propósito general	Generadores de programas

(Si creéis que un lenguaje puede estar en más de una categoría, indicadlo).

4. Clasificad los elementos de ingeniería del software siguientes según si son un método, una herramienta o un procedimiento:

- Redacción y generación de informes.
- Etapas del ciclo de vida.
- Proceso de verificación de una solución.
- Generador del modelo de bases de datos.
- Sistema de gestión de proyectos.
- Programación modular y programación estructurada.
- Diseño del modelo conceptual de una base de datos y generación del modelo físico.
- Generador de formularios de entrada de datos.

5. Vistas las tareas siguientes, indicad cuáles creéis que podrían ser hitos:

- Presentación informe detallado del análisis de requerimientos.
- Inicio del proyecto.
- Redacción y generación de informes.
- Diseño del modelo conceptual de la base de datos.
- Fichero generado.
- Generación del modelo físico de la base de datos.
- Entrega del modelo conceptual de la base de datos.
- Módulo validado por el cliente.
- Inicio fase mantenimiento.
- Copias de seguridad validadas.

6. Indicad para cada una de las interrelaciones entre tareas si creéis que son del tipo fin-inicio, fin-fin, inicio-inicio o inicio-fin:

- Lectura secuencial de un fichero de datos-generación de la media de los valores leídos.
- Validación diseño detallado-desarrollo de los módulos de software.
- Traslado material del almacén del proveedor a casa del cliente-vigilancia del material del almacén del proveedor.

- Traducción del texto de un idioma a otro-revisión ortográfica y gramatical de un texto.

7. A partir de la siguiente descripción del trabajo pendiente, identificad las posibles tareas, sus interrelaciones y dibujad el diagrama de Gantt.

"El objetivo del proyecto es la producción y distribución de una nueva línea de perfumes de señor para que se presente en la Feria Internacional del Perfume de París del próximo año".

Para poder llevar a cabo este proyecto, se deben realizar una serie de tareas:

- Realizar un estudio de mercado (4 semanas).
- Desarrollar el diseño del producto y las especificaciones de coste (2 semanas + 1 semana).
- Producir un prototipo (3 semanas).
- Probarlo (1 semana).
- Coordinar la producción con el equipo de fabricación (8 semanas).
- Diseñar y producir el envase (2 semanas + 4 semanas).
- Envasado del producto (1 semana).
- Coordinar las campañas de publicidad para su lanzamiento al mercado (4 semanas).

Y con el fin de controlar el avance del proyecto, se ha decidido establecer los puntos de control siguientes:

- Análisis de mercado completo.
- Diseño y precio aprobados.
- Prototipo finalizado para las pruebas.
- Prototipo aprobado para la producción.
- Producto empaquetado y enviado al almacén.
- Producto a punto para salir al mercado.

8. A partir del ejercicio anterior, identificad el camino crítico del proyecto, indicando qué tareas forman parte de este camino crítico y, de manera intuitiva, indicad cuál sería el margen de cada una de las tareas.

9. Asociad para cada una de las tareas indicadas qué rol del equipo de trabajo de los mencionados las asumiría.

Tareas:

- Instalación de los sistemas operativos de las máquinas y su configuración.
- Diseño de los iconos del programa.
- Coordinación del equipo de ingeniería.
- Definición de los productos y la tipología para una solución de ofertas comerciales.
- Diseño del plan de comunicación del lanzamiento de una nueva web en el mercado.
- Gestión y control de las horas y del presupuesto del proyecto.
- Análisis del proceso de compraventa en un mercado virtual.
- Programación de las rutinas de código del programa.
- Instalación y configuración del servidor de bases de datos.
- Comprobación del cumplimiento de los requerimientos del diseño gráfico de la solución.
- Coordinación del equipo de aseguramiento de la calidad.

Roles:

- Consultor de procesos.
- Consultor de catálogos.
- Consultor de marketing.
- Diseñador gráfico.
- Director técnico.
- Ingeniero de software.
- Administrador de bases de datos.
- Administrador de sistemas.
- Verificador de "Look & Feel".
- Responsable de calidad.
- Administrador del proyecto.

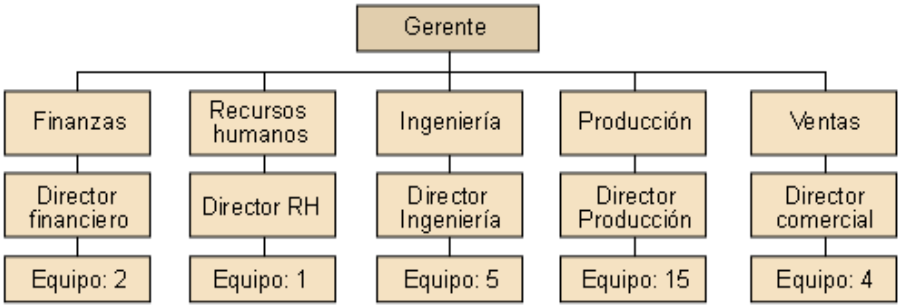
10. A continuación indicamos la estructura organizativa de una empresa y el personal y los cargos de cada departamento.

Se ha decidido utilizar el sistema operativo SOP1, que requiere licencia servidor y una licencia cliente para cada usuario que acceda a él, y el procesador de textos PTX1, que requiere una licencia cliente para cada usuario que lo utilice. Lo mismo sucede con el programa de gestión

de hoja de cálculo FCAL1, el sistema de gestión de proyectos SGP1 y el programa de antivirus AVS1.

Así, se adquirirá el sistema de gestión de mensajería electrónica, SME1, que implica los mismos requerimientos de licencias que el sistema operativo SOP1.

Se pide identificar las licencias de servidor y cliente de cada aplicación, teniendo en cuenta la estructura organizativa indicada y las necesidades funcionales de la compañía siguientes:



- Se dispondrá de un único servidor de ficheros e impresoras (sistema operativo) en el que también se instalará el servidor de mensajería electrónica.
- Todos los trabajadores tienen acceso a este servidor y al correo electrónico.
- Sólo el equipo de ingeniería, el director de producción y el gerente han de gestionar proyectos.
- Los responsables de cada departamento, el gerente y los miembros de los departamentos de Finanzas y RRHH deben trabajar con el procesador de textos y la hoja de cálculo.
- Todo el personal debe tener instalado en la máquina el programa antivirus.

Solucionario

Ejercicios de autoevaluación

1. Aplicación para el control de los pedidos de los clientes de una empresa.

- Software de sistemas
  Sistema operativo. Programa de interfaz de comunicaciones entre un ordenador y un dispositivo móvil.
- Software de tiempo real
  Herramienta de apoyo al control aéreo. Sistema de control de los semáforos de una ciudad. Herramienta de posicionamiento geográfico y seguimiento de un paquete.
- Herramientas de uso personal
  Aplicación de agenda. Herramienta de apoyo al diseño de piezas.
- Software de ingeniería y científico
  Analizador de los componentes de una solución química. Simulador de una constelación y/o del movimiento de los planetas del sistema solar.
- Herramientas de gestión y rediseño de procesos organizativos
  Sistema de gestión de proyectos. Aplicación para el control de los pedidos de los clientes de una empresa.
- Aplicaciones de inteligencia artificial
  Programa de reconocimiento de la voz. Sistema de visión artificial.

2.

Fortran (1957)
Lisp (1959)
Cobol (1960)
Simula (1962)
RPG (década los sesenta)
Algol (década de los sesenta)
Basic (1965)
Pascal (1970)
C (1972)
Prolog (década de los setenta)
Modula2 (finales de los setenta)
C++ (década de los ochenta)
Ada (1983)
Eiffel (1986)

3.

Lenguajes de aplicación en inteligencia artificial	Lenguajes de aplicaciones científicas	Lenguajes orientados a objetos	Lenguajes de propósito general	Generadores de programas
Prolog, Lisp, "Simula"	Fortran, "Algol", PL1	C++, Simula, Smalltalk, Eiffel	Cobol, Algol, PL1, Basic, C, Pascal, Modula2, Ada	RPG

4.

Método	Etapas del ciclo de vida, programación modular y programación estructurada, diseño del modelo conceptual de una base de datos y generación del modelo físico.
Herramienta	Generador del modelo de bases de datos, sistema de gestión de proyectos, generador de formularios de entrada de datos.
Procedimiento	Redacción y generación de informes, proceso de verificación de una solución.

5. Aunque dependerá de cada proyecto, del significado de la tarea y del esfuerzo necesario para llevarla a cabo, en aquel proyecto se podrían considerar hitos, de manera general, las tareas siguientes:

- Entrega del modelo conceptual de la base de datos.
- Módulo validado por el cliente.
- Presentación informe detallado del análisis de requerimientos.
- Inicio del proyecto.
- Inicio de la fase mantenimiento.
- Fichero generado.
- Copias de seguridad validadas.

Todas indican acciones completadas o "puntos de control" dentro del proyecto.

En cambio, las tareas:

- redacción y generación de informes,
- diseño del modelo conceptual de la base de datos,
- y generación del modelo físico de la base de datos

representan una "tarea" que se debe realizar con una duración para completarla.

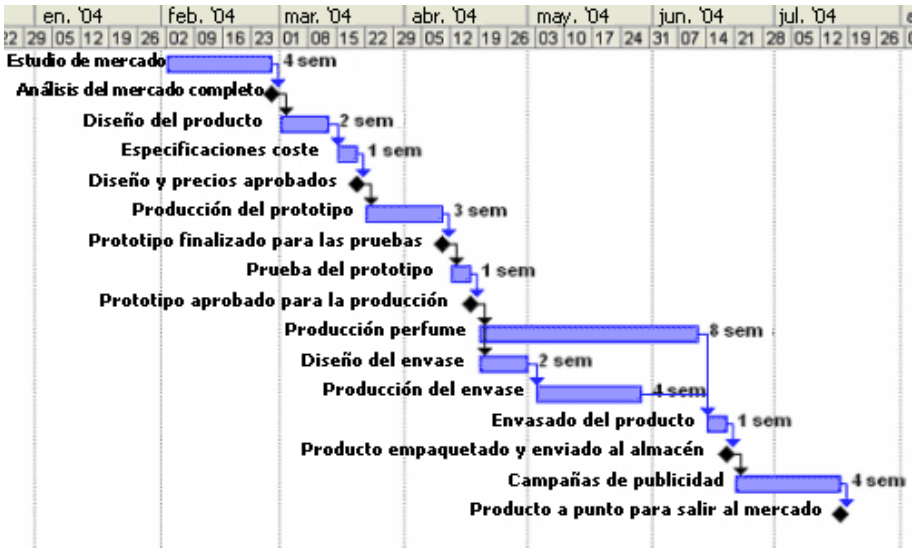
6.

- Validación diseño detallado-desarrollo de los módulos de software (fin-inicio).
  Dado que hasta que no se ha validado el diseño detallado de las funcionalidades, no se pueden empezar a desarrollar los módulos según este diseño.
- Traducción del texto de un idioma a otro-revisión ortográfica y gramatical de un texto (inicio-inicio).
  Se pueden realizar las dos tareas en paralelo, de manera que cuando empiece la traducción, también pueda empezar la revisión.
  En este caso, también se podrían secuenciar las tareas, de manera que cuando acabe la traducción, se inicie la corrección.
- Lectura secuencial de un fichero de datos-generación de la media de los valores leídos (fin-fin).
  Considerando que a medida que se va leyendo el fichero de datos, se puede ir calculando la media de los valores leídos hasta el momento, habría una relación fin-fin entre las dos tareas. También se podría considerar una relación fin-inicio, si la media no se calculara hasta el final de la lectura de todo el fichero, en lugar de calcularse después de leer cada valor.
- Traslado material del almacén del proveedor a casa del cliente-vigilancia del material del almacén del proveedor (inicio-fin).
  En el momento en el que se inicia el traslado del material del almacén del proveedor al cliente, se detiene la tarea de vigilancia del material que hay en el almacén (inicio-fin).

7. Se descompone el proyecto en tareas, se intenta poder efectuar tareas en paralelo, con el fin de adelantar el trabajo (producción del perfume y diseño y producción del envase):

NÚM: TAREA	TAREA	DURACIÓN	PREDECESORA
1	Estudio de mercado	4 sem.	
2	Análisis del mercado completo	0 días	1
3	Diseño del producto	2 sem.	2
4	Especificaciones coste	1 sem.	3
5	Diseño y precios aprobados	0 días	4
6	Producción del prototipo	3 sem.	5
7	Prototipo finalizado para las pruebas	0 días	6
8	Prueba del prototipo	1 sem.	7
9	Prototipo aprobado para la producción	0 días	8
10	Producción perfume	8 sem.	9
11	Diseño del envàse	2 días	9
12	Producción del ehvase	4 sem.	11
13	Envasado del producto	1 sem.	12;10
14	Producto empaquetado y enviado al almacén	0 días	13
15	Campañas de publicidad	4 sem.	14
16	Producto a punto para salir al mercado	0 días	15

El diagrama de Gantt tendría el aspecto siguiente:



8. El camino crítico del proyecto está compuesto por las tareas que, si se retrasan, provocarán, seguro, el atraso de la fecha de fin del proyecto.

En el ejemplo, son todas las tareas excepto las tareas de diseño del envase y de producción de éste.

Para evaluar los márgenes de las diferentes tareas, es necesario centrarse en las tareas que no forman parte del camino crítico.

Las tareas críticas no tienen margen, ya que si alguna se retrasara, el proyecto también se retrasa.

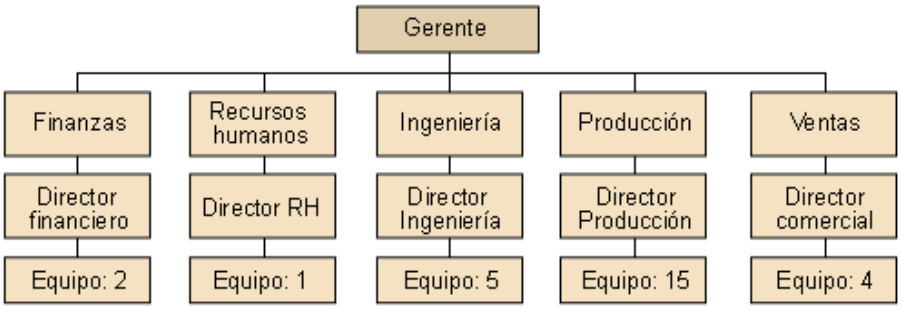
En cambio, las tareas de diseño del envase y de producción del envase tienen un margen de atraso posible.

De manera intuitiva, estas dos tareas (duración prevista 2 + 4 semanas) se realizan en paralelo con la producción del perfume (con una duración prevista de ocho semanas), por lo que tienen un margen posible de dos semanas de atraso. Si se retrasan más de dos semanas, pueden afectar al fin del proyecto.

9.

Roles	Tareas
Consultor de procesos	Análisis del proceso de compraventa en un mercado virtual
Consultor de catálogos	Definición de los productos y tipología para una solución de ofertas comerciales
Consultor de marketing	Diseño del plan de comunicación del lanzamiento de una web nueva al mercado
Diseñador gráfico	Diseño de los iconos del programa
Director técnico	Coordinación del equipo de ingeniería
Ingeniero de software	Programación de las rutinas de código del programa
Administrador de bases de datos	Instalación y configuración del servidor de bases de datos
Administrador de sistemas	Instalación de los sistemas operativos de las máquinas y su configuración
Verificador de "Look & Feel"	Comprobación del cumplimiento de los requerimientos del diseño gráfico de la solución
Responsable de calidad	Coordinación del equipo de aseguramiento de la calidad
Administrador del proyecto	Gestión y control de las horas y del presupuesto del proyecto

10.



Total trabajadores compañía y por departamento:

- Gerencia: 1
- Finanzas: 1 + 2
- Recursos Humanos: 1 + 1
- Ingeniería: 1 + 5
- Producción: 1 + 15
- Ventas: 1 + 4
- Total empresa: 33

Licencias requeridas:

- Sistema operativo (SOP1): 1 licencia servidor + 33 licencias cliente
- Sistema mensajería electrónica (SME1): 1 licencia servidor + 33 licencias cliente
- Sistema de gestión de proyecto (SGP1): 8 licencias
- Procesador de textos (PTX1): 9 licencias
- Hoja de cálculo (FCAL1): 9 licencias
- Antivirus (AVS1): 33 licencias

Bibliografía

Barceló, M.; Costa, M.; Quer, C. (1993). *Anàlisi d'aplicacions informàtiques*. Barcelona: Edicions UPC.

Pressman, Roger S. (1997). *Ingeniería de Software. Un enfoque práctico*. Madrid: McGraw-Hill.

Drudis, A. (1999). *Gestión de Proyectos. Cómo planificarlos, organizarlos y dirigirlos*. Barcelona: Ediciones Gestión 2000.

Villarreal, S. (1999). *Introducción a la computación*. México DF: McGraw-Hill.

Domingo Ajenjo, A. (2000). *Dirección y gestión de proyectos*. Madrid: Editorial Ra-Ma.