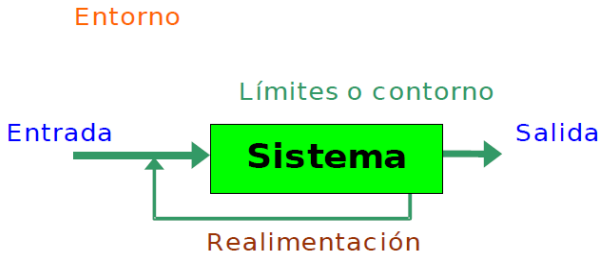
1.1. Sistema Informático

Podemos definir un sistema, como un conjunto de estrategias, herramientas, conceptos u objetos, que de forma conjunta pueden describir un comportamiento o modelarlo.

Según el Diccionario de la Real Academia de La Lengua Española, textualmente "un sistema es un conjunto de cosas que ordenadamente entre si contribuyen a un determinado objetivo".



Veamos ahora cuáles son los principales elementos que podemos identificar en cualquier sistema:

· El objetivo del sistema.
· Los componentes del sistema.
· Las relaciones entre componentes, que determinan su estructura
· El entorno.
· Los límites del sistema.

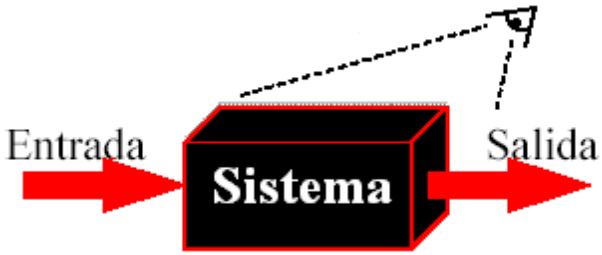
En muchos sistemas la salida influye en el estado del sistema, eso se denomina realimentación.

Para poder identificar más claramente estos elementos veámoslo con un sistema conocido: el Sistema de Formación Profesional.

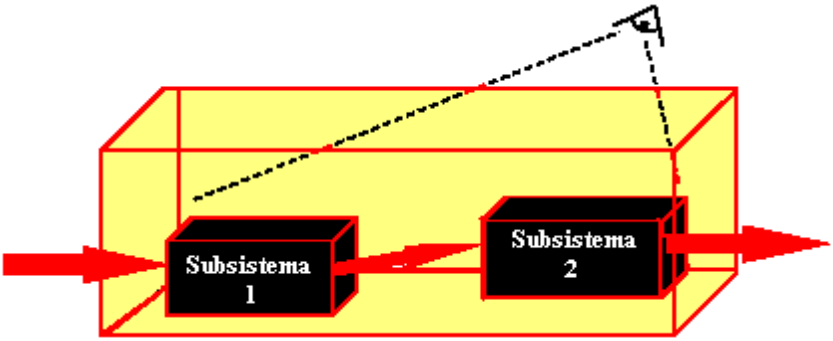
- **El objetivo** de este sistema, es llegar a conseguir los objetivos de la formación profesional, que no son otros que formar profesionales competentes para desempeñar los puestos de trabajo propios de cada familia profesional y cada ciclo formativo.
- **Los componentes del sistema** son los tutores, los alumnos, la plataforma educativa, los materiales multimedia, las herramientas de comunicación, etc.
- **Las relaciones**:
  - Los alumnos y profesores se relacionan en el aula.
  - Los alumnos y profesores están en contacto a través de las herramientas de comunicación.
  - Los materiales multimedia son colgados en la plataforma educativa.
- **El entorno del sistema** es en este caso el aula de informática, dentro del centro de enseñanza.
- **Los límites del sistema** están establecidos en la programación del módulo.

Lo que nosotros necesitamos, es conocer la manera en que las empresas manejan la información necesaria para su funcionamiento, constituyendo sistemas. Para ello analizaremos también las técnicas creadas para el estudio de los sistemas, lo que se llama teoría general de sistemas o enfoque sistémico.

Este enfoque lo que hace es analizar los sistemas desde un punto de vista global. Se inicia como si el sistema fuera una caja negra en la que sólo podemos distinguir la información de entrada y la de salida, para ir progresivamente descomponiendo el sistema.



Una vez estén identificados los límites del sistema y sus relaciones, abrimos la caja negra y vemos el interior, donde nos encontramos otras grandes cajas negras, que forman los subsistemas de nuestro sistema, además de las relaciones entre ellos. Cada una de estas cajas negras a su vez se irá descomponiendo hasta que nos encontremos con problemas de fácil resolución. La técnica utilizada en este enfoque sistémico u holístico, puedes encontrarla en otros contextos también con el nombre de técnica de “divide y vencerás”.



1.2. Información, datos.

Hemos visto que en el sistema hay una serie de entradas y una serie de salidas, por lo tanto entendemos que existe un flujo de “cosas” que entran, salen de nuestro sistema, y en la mayoría de las veces se transforman dentro de él. Pues bien de lo que estamos hablando es de la información, o de datos solamente.

Normalmente se tiende a confundir estos dos términos, por lo que es importante tener claro que:

Un dato es un registro de un hecho, acontecimiento, situación, transacción o estado. Por tanto un dato sería por ejemplo una serie de números y caracteres: como “200211”

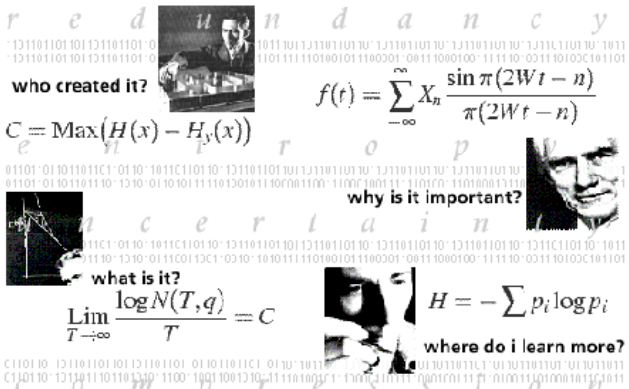
Por el contrario:

Información es uno o varios datos, que están procesados de una manera concreta para poder darles un significado dentro de un contexto.

Siguiendo con nuestros ejemplos anteriores, “200211” para ti podría significar “20-02-11”, la fecha del examen de febrero”; sin embargo para mi, “200211” podría significar “200211 €, la cantidad de dinero que tengo pendiente de pago en mi hipoteca”.

Así podríamos decir que el dato “200211”, se convierte en Información, en el momento en el que tenga un significado válido para nuestro sistema.

Matemáticamente, Claude Shannon, estableció una fórmula que calculaba la cantidad de información que se comunica, en función del número total de mensajes que se pueden enviar, teniendo en cuenta que cualquier mensaje podría tener la misma posibilidad de ser enviado.



También tiene en cuenta que la comunicación más elemental que puede producirse es el envío de una de estas dos alternativas posibles “si / no”, o puedes encontrarla en otros textos como “verdadero / falso”. Informativamente esta información se traduce como:

Si 1
No 0

y recibe el nombre de “bit”, y es la menor cantidad de información que puede considerarse.

Pues bien, Shannon establece que cuando existe la posibilidad de enviar no dos, sino una cantidad n de diferentes mensajes equiprobables (con la misma probabilidad p=1/n), la cantidad de información que se comunica, medida en bits cuando se envía uno de los n mensajes es:

I = log2 n = log2 (1/p)

La cantidad de información de un mensaje, es equivalente al número mínimo de dígitos binarios (ceros o unos) necesarios para codificar todos los posibles mensajes a enviar.

1.3. Sistemas de información.

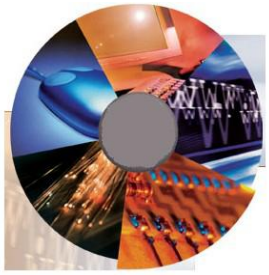
Un Sistema de Información es el dedicado a coordinar los flujos y los registros de información necesarios para desarrollar las actividades de una empresa de acuerdo a una estrategia de negocio.

No hay que confundir los sistemas de información, con los sistemas de información automatizados. Y es que cuando uno piensa en un sistema de información lo primero que le viene a la mente es un conjunto de ordenadores con una serie de programas (software), lo que hoy en día se les llama tecnologías de la información, pero esto no debe de hacernos olvidar el concepto inicial. Más adelante veremos en detalle que

esta aplicación de las tecnologías de la información en los sistemas de información es lo que se conoce como sistema de información automatizado.

Antes de conocer los elementos que componen un sistema de información, es necesario dar una definición de éstos, un poco más formal, ya que hay que tener en cuenta, que el objetivo final de los sistemas de información es propiamente la correcta gestión de la información.

Quizás la definición más completa sea la ofrecida por R. Andreu en 1991, que decía:



Un Sistema de información es un conjunto formal de procesos que, operando sobre una colección de datos estructurada según las necesidades de la empresa, recopilan, elaboran y distribuyen la información (o parte de ella) necesaria para las operaciones de dicha empresa y para las actividades de dirección y control correspondientes (decisiones) para desempeñar su actividad de acuerdo a su estrategia de negocio.

1.4. Elementos de un sistema de información.

A estas alturas ya tendrás una idea clara del concepto de sistema de información, así que es el momento de conocer cuales son los elementos que lo componen:

· La información
· Los usuarios
· El equipo de soporte.
· Los procedimientos de trabajo.

La información evidentemente es el objetivo de nuestro sistema, por lo tanto es la pieza clave del mismo, adaptándose a los usuarios que la manejan y al equipo disponible siguiendo los procedimientos de trabajo que la empresa haya dispuesto para que sea rentable para sus actividades.

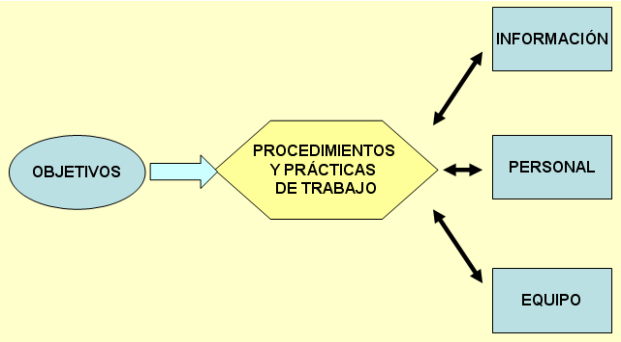
Los usuarios son los individuos de la estructura organizativa del sistema encargados de recabar, introducir, manejar o usar la información, para realizar los objetivos de la empresa siguiendo los procedimientos de trabajo convenientes.

El equipo de soporte es el encargado de la comunicación de la información, así como de su procesamiento, y en su caso almacenaje, y esto es importante tenerlo claro, ya que el equipo de soporte podría ser por ejemplo el bolígrafo que usa un administrativo para recoger los datos de los clientes, de la misma forma que podía ser

también un ordenador para realizar la misma tarea.

Los procedimientos o prácticas habituales de trabajo, por último, son una serie de reglamentos o guías necesarios para coordinar los distintos aspectos de una empresa.

De una manera gráfica, la relación existente entre los distintos elementos que forman el sistema de información sería la siguiente:



1.5. Aplicaciones de las T.I. a los sistemas de información.

Por fin llegó el momento de ver cuál sería la aplicación de las nuevas tecnologías llamadas también “tecnologías de la información o TI” a los sistemas de información, ya que si lo recuerdas, hasta ahora hemos insistido mucho en que no había que confundir los términos de **sistemas de información (SI)** con los de **sistemas de información automatizados (SIA)**.

Hasta este momento hemos trabajado con los sistemas de información independientemente de la tecnología empleada.

En la mayoría de los casos, los equipos informáticos aportan una serie de valores añadidos a las funcionalidades de un sistema de información, reduciendo el tiempo de cálculos, estudios, procesamiento y análisis de la información, mejorando la gestión de los datos con la ayuda de los sistemas gestores de bases de datos, etc. Hoy en día la cantidad de datos a manejar es enorme. El manejo de todos estos datos sin sistemas informáticos es imposible. Mediante ordenadores las empresas pueden recoger y generar grandes cantidades de datos.

Hay diferentes tipos de sistemas de información en función de las diferentes tareas a realizar dentro de la empresa.

Según el grado de complejidad podemos encontrar los siguientes tipos de sistemas de información:

- Sistemas de información operativos.
- Sistemas de información de dirección.
- Sistemas de soporte a la toma de decisiones.
- Sistemas de soporte ejecutivo.
- Sistemas expertos.

- Sistemas de información ofimáticos.

1.5.1. Sistemas de información operativos (T.P.S.“Transaction Procesing Systems”).

Piensa en las siguientes tareas que son ya algo cotidiano en la mayoría de las empresas:

- Pago de cuentas
- Entrada de albaranes
- Gestión de pedidos
- Gestión de inventario

Esta información la emplean los responsables de la operativa diaria para supervisar el día a día del sistema.

Así nos permite analizar los errores detectados por nosotros o por nuestros clientes o proveedores.

Los sistemas de información operativos son los que almacenan y ayudan a la realización de transacciones.

Podríamos incluirlos en la categoría de gestión administrativa. Además de realizar la transacción aportan información sobre las transacciones pasadas.

Los **TPS** realizan las siguientes funciones con los datos:

- Entrada de datos.
- Validación de datos de entrada.
- Procesamiento de la información.
- Actualización de registros.
- Generación de salidas (papel, pantalla, modem)

1.5.2. Sistemas de información a la dirección (M.I.S.“Management Information Systems”)

Al principio los sistemas informáticos únicamente almacenaban información de tipo contable, de una manera rutinaria y a posteriori. Uno de los grandes cambios fue el paso dado al realizar entradas interactivas y el de obtener información "online".

El objetivo de un sistema de información a la dirección es el de proveer de información a los directivos que les permita resolver problemas, controlar la empresa y tomar decisiones.

Habitualmente se emplean en situaciones estructuradas y repetitivas y normalmente aportan datos sobre:

· Ventas de artículos en promoción
· Evolución de facturación de clientes
· Diversos coeficientes y ratios

Muy a menudo los **TPS** (Sistemas de Información Operativos) alimentan los datos que utilizan estos sistemas.

Así un **MIS** puede usar la información de ventas recogida durante varios años por un **TPS** para predecir los ventas del año siguiente. Con la previsión de ventas, el director comercial puede tomar decisiones respecto a acciones de publicidad futuras.

Se utilizan en todas las áreas de actividad, incluyendo:

· Planificación
· Marketing
· Finanzas
· Fabricación
· Recursos humanos
· Gestión de proyectos

Se emplean básicamente por directivos de rango medio. Las decisiones de planificación que se toman con la ayuda de estos sistemas son para un plazo corto y sirven para cumplir objetivos estratégicos definidos por los niveles de gestión más altos.
La información se produce en forma de informes "online" a la dirección y es presentada de una forma fija al usuario.

1.5.3. Sistemas de soporte a la toma de decisiones. (D.S.S.“Decision Support Systems”)

Piensa en una hoja de cálculo. Es un ejemplo básico de sistema de información **interactivo**, de soporte a la toma de decisiones, que ayuda a resolver problemas de gestión. De ahí se puede llegar a sistemas con un grado de complejidad enorme, que permitan la elaboración de **hipótesis** complejas y su validación. Son utilizados por directivos de empresas de alto grado y se utilizan pare crear modelos que ayuden a analizar el problema y a la toma de decisiones.

Un sistema de soporte a la toma de decisiones puede presentar diferentes soluciones posibles para el problema y permite al usuario introducir, obtener y analizar datos a su gusto. Puede realizar modelos que le permitan conocer mejor el sistema. No necesita de informáticos para cambiar el programa, meter datos o crear relaciones entre los datos. Por tanto la información está disponible al momento**. El sistema nunca tomará ninguna decisión por el usuario, pero le ayudará a decidir exponiendo las ventajas e inconvenientes de cada caso.**

Hay muchas herramientas comerciales en el mercado, sobre todo en las siguientes áreas :

Planificación financiera: Los Bancos analizan el impacto de las variaciones en los tipos de interés, regulaciones fiscales y variaciones en el mercado de divisas.
Fabricación: Analizan el Impacto de diferentes sistemas de producción, salarios, adquisición de nuevas máquinas.
Fusiones y adquisiciones.
Desarrollo de nuevos productos.
Ampliaciones de plantas.

Previsión de ventas.

1.5.4. Sistemas de soporte ejecutivo. (ESS "Executive Support System")

Ejecutivo es sinónimo de dirección de alto nivel y de decisiones estratégicas. Como vimos antes en
la definición de los niveles de las empresas, la responsabilidad fundamental de un ejecutivo es el
establecimiento de objetivos a largo plazo y de las estrategias de alto nivel necesarias para llevarlas
a cabo.

Así **los sistemas de soporte ejecutivo están diseñados específicamente para las necesidades de información de los ejecutivos, planificación directiva, análisis y seguimiento temporal de las decisiones.**

Los sistemas de soporte ejecutivo contienen gran cantidad de datos e información externa a la
empresa que se enlazan y comparan con información interna a la misma, normalmente obtenida del sistema de información a la dirección, y adaptadas a las necesidades de los ejecutivos. Su principal
función es sintetizar y controlar el volumen de información que un ejecutivo debe leer.

1.5.5. Sistemas expertos

También se denominan **sistemas basados en el conocimiento**. Es un tipo de programa preparado para proponer decisiones o resolver problemas en un campo muy concreto. Utiliza el conocimiento de un experto humano que se transforma en un conjunto de reglas. Las reglas se obtienen de la experiencia de los expertos humanos en ese campo. Funciona como un experto en una disciplina concreta, resolviendo problemas que requieran conocimiento, inteligencia y experiencia.

En muchos casos están integrados en sistemas de soporte a la toma de decisiones y en sistemas de soporte ejecutivo. Hay programas comerciales que ayudan a crear sistemas expertos.

Se usan en Hospitales, laboratorios de investigación, plantas industriales, talleres de reparación, pozos petrolíferos... etc.

1.5.6. Sistemas de información ofimáticos (OIS "Office Information System")

En los últimos 20 años los ordenadores han cambiado radicalmente el entorno de trabajo en las oficinas y la manera de trabajar de las empresas.

Los sistemas de información ofimáticos ayudan en la preparación, almacenamiento, obtención, reproducción y comunicación de información en todos los puntos de la organización ya estén en la misma situación o alejados en el espacio.

Utiliza muchos elementos de todos conocidos:

- Procesadores de textos.

· Hojas de cálculo.
· Sistemas de gestión de bases de datos.
· Correo electrónico.
· Fax.
· Sistemas de gestión documental.
· Agenda electrónica.
· Libretas electrónicas.

Como cuestión final, esta pregunta:

¿En un futuro próximo llegaremos a la oficina sin papel? Parece probable.

1.6. Programas y aplicaciones

Según la RAE Programa es: c*onjunto unitario de instrucciones que permite a un ordenador realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos*, etc. Pero normalmente se entiende por programa un conjunto de instrucciones ejecutables por un ordenador.

Aplicación es el software formado por uno o más programas, la documentación de los mismos y los archivos necesarios para su funcionamiento, de modo que el conjunto completo de archivos forman una herramienta de trabajo en un ordenador.

Normalmente en el lenguaje cotidiano no se distingue entre aplicación y programa; en nuestro caso entenderemos que la aplicación es un software completo que cumple la función completa para la que fue diseñado, mientras que un programa es el resultado de ejecutar un cierto código entendible por el ordenador.

1.6.1. Definición de programa

Una computadora es una **máquina programable**, es decir, capaz de ejecutar programas desarrollados por programadores.

Un programa es un conjunto de instrucciones u órdenes que indican a la máquina las operaciones que ésta debe realizar con unos datos determinados. En general, todo programa indica a la computadora cómo obtener unos **datos de salida**, a partir de unos **datos de entrada**.

En la siguiente figura se muestra, gráficamente, el funcionamiento básico de un programa.

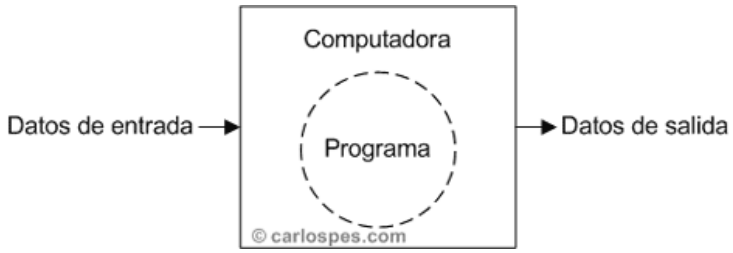


Figura - Funcionamiento básico de un programa en una computadora digital.

Por ejemplo, un programa que sirva para realizar la suma de dos números enteros cualesquiera (por ejemplo, del 3 y el 5), podría representarse de la siguiente manera:

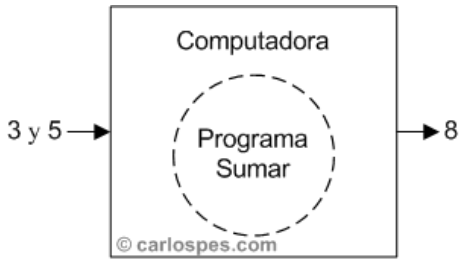


Figura - Programa sumar.

En un programa, los **datos de entrada** son los que la computadora va a procesar. Los **datos de salida** son datos derivados, es decir, obtenidos a partir de los datos de entrada. Por esta razón, a los datos de salida se les considera más significativos que a los datos de entrada. Ambos tipos de datos son información (textos, imágenes, sonidos, vídeos,...) que maneja la computadora. Sin embargo, en un sentido más filosófico, a los datos de entrada se les considera la materia prima de los datos de salida, considerados estos como la verdadera información.

1.6.2. La codificación

En el ciclo de vida de un programa, una vez que los algoritmos de una aplicación han sido diseñados, ya se puede iniciar la fase de codificación. En esta etapa se tienen que traducir dichos algoritmos a un lenguaje de programación específico; es decir, las acciones definidas en los algoritmos hay que convertirlas a instrucciones.

Una vez que los algoritmos de una aplicación han sido diseñados, ya se puede iniciar la fase de codificación. En esta etapa se tienen que traducir dichos algoritmos a un lenguaje de programación específico; es decir, las acciones definidas en los algoritmos hay que convertirlas a instrucciones.

Para codificar un algoritmo hay que conocer la sintaxis del lenguaje al que se va a traducir. Sin embargo, independientemente del lenguaje de programación en que esté escrito un programa, será su algoritmo el que determine su lógica.

La lógica de un programa establece cuales son sus acciones y en qué orden se deben ejecutar. Por tanto, es conveniente que todo programador aprenda a diseñar algoritmos antes de pasar a la fase de codificación.

En programación, un algoritmo establece, de manera genérica e informal, la secuencia de pasos o acciones que resuelve un determinado problema.

Los algoritmos constituyen la documentación principal que se necesita para poder iniciar la fase de codificación y, para representarlos, se utiliza, fundamentalmente, dos tipos de notación: pseudocódigo y diagramas de flujo.

El diseño de un algoritmo es independiente del lenguaje que después se vaya a utilizar para codificarlo.

1.6.3. Lenguaje de programación

Un lenguaje de programación se puede definir como un lenguaje artificial que permite escribir las instrucciones de un programa informático.

En programación, una instrucción indica a la computadora la o las operaciones que ésta debe realizar con unos datos terminados.

1.6.4. Definición de aplicación

La mayoría de los usuarios de programas no tienen conocimientos de programación, pero, sí saben cómo utilizar los programas que ejecutan, tales como: procesadores de texto, gestores de bases de datos, hojas de cálculo, juegos, etc. Estos programas entran dentro de la categoría de **software de aplicación**, también llamados aplicaciones de usuario o, simplemente, **aplicaciones**.

1.6.5. Tipos de Programas

A la persona que desarrolla un programa se le llama **programador**, y no tiene que ser, necesariamente, la misma que la que hace uso de dicho programa. A esta segunda persona se le denomina **usuario**. De hecho, la mayoría de los usuarios no necesitan tener conocimientos avanzados de informática para utilizar los programas.

Por ejemplo, un usuario común no necesita saber que casi todos los programas se pueden clasificar dentro de tres categorías principales:

* software de aplicación
* software del sistema
* software de red

1.6.5.1. Software de Aplicación

La mayoría de los usuarios de programas no tienen conocimientos de programación, pero, sí saben cómo utilizar los programas que ejecutan, tales como: procesadores de texto, gestores de bases de datos, hojas de cálculo, juegos, etc. Estos programas entran dentro de la categoría de software de aplicación, también llamados aplicaciones de usuario o, simplemente, aplicaciones.

Un programa, por sí solo, puede constituir una aplicación informática. Sin embargo, lo normal es que el término aplicación haga referencia a un software compuesto por un conjunto de programas relacionados entre sí.

A las aplicaciones que "ayudan en el trabajo a los humanos" se les considera herramientas de software.

1.6.5.2. Software del Sistema

Además de los programas de aplicación, también existen otros muchos que pueden pasar desapercibidos al usuario común. De ellos, los más importantes pertenecen al sistema operativo, el cual hace posible que la máquina funcione, ya que, gestiona los recursos hardware solicitados por las aplicaciones. El sistema operativo sirve de intermediario (interfaz) entre los programas y la computadora. De forma que, cuando un usuario ejecuta un programa, éste solicitará al sistema operativo las acciones que quiere realizar en la máquina para satisfacer al usuario.

En cierto sentido, el sistema operativo es el encargado de coordinar al usuario, a las aplicaciones y al hardware. Algunos de los sistemas operativos más usados en la actualidad son: Linux, MS-DOS, UNIX, Windows XP,... Estos programas entran dentro de la categoría de software del sistema.

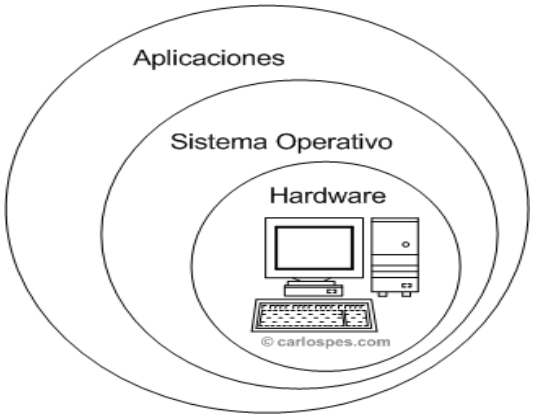


Figura - Relación entre las aplicaciones, el sistema operativo y el hardware.

1.6.5.3. Software de Red

Además de los dos tipos de programas ya citados (de plicación y del sistema), también es importante conocer la existencia de una tercera categoría, llamada software de red.

En el software de red se incluyen programas relacionados con la interconexión de equipos informáticos, es decir, programas necesarios para que las redes de computadoras funcionen.

Entre otras cosas, los programas de red hacen posible la comunicación entre las computadoras, permiten compartir recursos (software y hardware) y ayudan a controlar la seguridad de dichos recursos.

1.7. Tipos de lenguajes de programación

Existen dos tipos de lenguajes claramente diferenciados; **los lenguajes de bajo nivel** y **los de alto nivel.**

El ordenador sólo entiende un lenguaje conocido como código binario o código máquina, consistente en ceros y unos. Es decir, sólo utiliza 0 y 1 para codificar cualquier acción.

Los lenguajes más próximos a la arquitectura hardware se denominan lenguajes de bajo nivel y los que se encuentran más cercanos a los programadores y usuarios se denominan lenguajes de alto nivel.

1.7.1. Lenguajes de bajo nivel

Son lenguajes totalmente dependientes de la máquina, es decir que el programa que se realiza con este tipo de lenguajes no se pueden migrar o utilizar en otras maquinas.

Al estar prácticamente diseñados a medida del hardware, aprovechan al máximo las características del mismo.

Dentro de este grupo se encuentran:

El lenguaje máquina: este lenguaje ordena a la máquina las operaciones fundamentales para su funcionamiento. Consiste en la combinación de 0's y 1's para formar las ordenes entendibles por el hardware de la máquina.

Este lenguaje es mucho más rápido que los lenguajes de alto nivel.

La desventaja es que son bastantes difíciles de manejar y usar, además de tener códigos fuente enormes donde encontrar un fallo es casi imposible.

El lenguaje ensamblador es un derivado del lenguaje máquina y esta formado por abreviaturas de letras y números llamadas nemotécnicos. Con la aparición de este lenguaje se crearon los programas traductores para poder pasar los programas escritos en lenguaje ensamblador a lenguaje máquina.

Como ventaja con respecto al código máquina es que los códigos fuentes eran más cortos y los programas creados ocupaban menos memoria. Las desventajas de este lenguaje siguen siendo prácticamente las mismas que las del lenguaje ensamblador, añadiendo la dificultad de tener que aprender un nuevo lenguaje difícil de probar y mantener.

1.7.2. Lenguajes de alto nivel

Son aquellos que se encuentran más cercanos al lenguaje natural que al lenguaje máquina.
Están dirigidos a solucionar problemas mediante el uso de EDD's.

> Nota: EDD's son las abreviaturas de Estructuras Dinámicas de Datos, algo muy utilizado en todos los lenguajes de programación. Son estructuras que pueden cambiar de tamaño durante la ejecución del programa. Nos permiten crear estructuras de datos que se adapten a las necesidades reales de un programa.

Se tratan de lenguajes independientes de la arquitectura del ordenador, por lo que, en principio, un programa escrito en un lenguaje de alto nivel, lo puedes migrar de una máquina a otra sin ningún tipo de problema.

Estos lenguajes permiten al programador olvidarse por completo del funcionamiento interno de la máquina/s para la que están diseñando el programa. Tan solo necesitan un traductor que entienda el código fuente con las características de la máquina.

Suelen usar tipos de datos para la programación y hay lenguajes de propósito general (cualquier tipo de aplicación) y de propósito específico (como FORTRAN para trabajos científicos).

1.7.3. Lenguajes de Medio nivel

Se trata de un término no aceptado por todos, pero que seguramente habrás oído. Estos lenguajes se encuentran en un punto medio entre los dos anteriores. Dentro de estos lenguajes podría situarse C ya que puede acceder a los registros del sistema, trabajar con direcciones de memoria, todas ellas características de lenguajes de bajo nivel y a la vez realizar operaciones de alto nivel.

PARTE 2: Ingeniería del Software.

2.1. Mitos del Software

Muchas de las causas de la crisis del software se pueden encontrar en una mitología que surge durante los primeros años del desarrollo del software. Los mitos del software propagaron información errónea y confusión.

Mitos de gestión.

Mito. Tenemos ya un libro que está lleno de estándares y procedimientos para construir software. ¿No le proporciona ya a mi gente todo lo que necesita saber?
Realidad. Está muy bien que el libro exista, pero ¿se usa? ¿Conocen los trabajadores su existencia? ¿Refleja las prácticas modernas de desarrollo de software? ¿Es completo?
Mito. Mi gente dispone de las herramientas de desarrollo de software más avanzadas; después de todo, les compramos las computadora más modernas.
Realidad. Las herramientas de ingeniería del software asistida por computador (CASE), son más importantes que el hardware para conseguir buena calidad y productividad.
Mito. Si fallamos en la planificación, podemos añadir más programadores y adelantar el tiempo perdido.
Realidad. El desarrollo de software no es un proceso mecánico como la fabricación. « ... añadir gente a un proyecto de software retrasado lo retrasa aún más».

Mitos del cliente.

Mito. Una declaración general de los objetivos es suficiente para comenzar a escribir los programas; podemos dar los detalles más adelante.
Realidad. Una mala definición inicial es la principal causa del trabajo baldío en software.
Mito. Los requisitos del proyecto cambian continuamente, pero los cambios pueden acomodarse fácilmente, ya que el software es flexible.
Realidad. Es verdad que los requisitos del software cambian, pero el impacto del cambio varía según el momento en que se introduzca.

- Si se pone cuidado al dar la definición inicial, los cambios solicitados al principio pueden acomodarse fácilmente.
- Cuando los cambios se solicitan durante el diseño del software, el impacto en el coste crece rápidamente.
- Los cambios en la función, rendimiento, interfaces u otras características durante la implementación (codificación y prueba) pueden tener un impacto importante sobre el coste.
- Cuando se solicitan al final de un proyecto, los cambios pueden producir un orden de magnitud más caro que el mismo cambio pedido al principio.

Mitos de los desarrolladores.

Mito. Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.
Realidad. Alguien dijo una vez: «cuanto más pronto se comience a escribir código, más se tardará en terminarlo». Los datos industriales indican que entre el 50% y el 70% de todo el esfuerzo dedicado a un programa se realizará después de que se le haya entregado al cliente por primera vez.
Mito. Hasta que no tengo el programa «ejecutándose», realmente no tengo forma de comprobar su calidad.
Realidad. Desde el principio del proyecto se puede aplicar uno de los mecanismos más efectivos para garantizar la calidad del software: la revisión técnica formal.
Mito. Lo único que se entrega al terminar el proyecto es el programa funcionando.
Realidad. Un programa que funciona es sólo una parte de una configuración del software que incluye programas, documentos y datos.

El reconocimiento de las realidades del software es el primer paso hacia la formulación de soluciones prácticas para su desarrollo. El intento de la ingeniería del software es proporcionar un marco de trabajo para construir software con mayor calidad.

2.2. Una visión general de la Ingeniería del Software.

La ingeniería del software es el establecimiento y uso de principios robustos de la ingeniería a fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales.
Históricamente hay cuatro etapas o generaciones en la ingeniería del software.

En la **etapa de desarrollo** convencional no existe como tal la ingeniería del software. En esta etapa, la mayor parte del tiempo está dedicado a la programación. Al no existir una verdadera metodología, las fases de diseño, codificación, implantación, mantenimiento y pruebas se confundían unas con otras, no acabándose casi nunca las especificaciones de las aplicaciones y por ello su diseño.

En la **segunda etapa o de desarrollo estructurado** se diseñan los programas siguiendo determinadas metodologías de programación: Jackson, Warnier, etc. Posteriormente aparecen las metodologías orientadas al ciclo de vida de las aplicaciones.
Estas metodologías buscan disminuir el tiempo empleado en la programación y aumentarlo en el análisis.

En la **tercera etapa** aparece el desarrollo orientado a objetos. Los elementos del mundo real se representan como objetos sobre los que se pueden realizar determinadas acciones.

En la **cuarta generación** se pueden encontrar, entre otros, los siguientes sistemas que permiten desarrollar el ciclo de vida del software: el modelo remolino, modelo Pinball.
La integración de herramientas CASE en sistemas de cuarta generación permite la generación de interfaces gráficas, de documentación automática, de código fuente y, sobre todo, la transferencia de información de unas fases a otras durante el desarrollo, del software, así como mantener los controles de calidad sobre el producto en desarrollo.

El trabajo que se asocia a la ingeniería del software se puede dividir en tres fases genéricas.

La **fase de definición** se centra sobre el qué.

Es decir, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué comportamiento del sistema, qué interfaces van a ser establecidas, qué restricciones de diseño existen, y qué criterios de validación se necesitan para definir un sistema correcto.

Por tanto, han de identificarse los requisitos clave del sistema y del software.

Tendrán lugar tres tareas principales:
- ingeniería de sistemas o de información,
- planificación del proyecto del software, y
- análisis de los requisitos.

La **fase de desarrollo** se centra en el cómo.
Es decir, durante el desarrollo un ingeniero del software intenta definir cómo han de diseñarse las estructuras de datos, cómo ha de implementarse la función como una arquitectura del software, cómo han de implementarse detalles procedimentales, cómo han de caracterizarse las interfaces, cómo ha de traducirse el diseño en un lenguaje de programación (o lenguaje no procedimental) y cómo ha de realizarse la prueba.

Las tres tareas específicas técnicas deberían ocurrir siempre:
- diseño del software
- generación de código y
- prueba del software

La **fase de mantenimiento** se centra en el cambio.
Durante la fase de mantenimiento se encuentran cuatro tipos de cambios:
- Corrección. El mantenimiento correctivo modifica el software para corregir los defectos.
- Adaptación. El mantenimiento adaptativo produce modificación en el software para acomodarlo a los cambios de su entorno externo.
- Mejora. El mantenimiento perfectivo lleva al software más allá de sus requisitos funcionales originales.
- Prevención. En esencia, el mantenimiento preventivo (también llamado reingeniería del software) hace cambios en programas de computadora a fin de que se puedan corregir, adaptar y mejorar más fácilmente.

2.3. Modelos de desarrollo de sistemas

Se estudian a continuación 5 modelos de desarrollo de sistemas:

A. Ciclo de vida clásico (Modelo en cascada, Waterfall)
B. Modelo de Prototipos
C. Modelo Incremental
D. El modelo de Espiral
E. Modelo genérico para desarrollo de Sistemas Orientados a Objetos

A. Ciclo de vida clásico (Modelo en cascada, Waterfall)

El ciclo de vida del diseño de aplicaciones consta de seis fases en cascada, una al continuación de otra. Estas seis fases son: análisis del sistema, análisis de requisitos del software, diseño, programación, prueba y mantenimiento del sistema.

- **Definición del problema**

  1. Análisis del sistema. (Análisis de requisitos de sistema)

  Cuando se ha de desarrollar software no se hace de modo aislado, sino que se han de estudiar los requisitos y funcionalidad de cada uno de los elementos que componen el sistema sobre el que ese software va ser utilizado.

  2. Análisis de requisitos del software.

  En esta fase, el analista ha de recoger, en colaboración con el cliente, toda la información relativa a los requisitos del sistema a desarrollar, entre otros a su función, rendimiento e interfaces.

  Hay que analizar costes, recursos y riesgos y definir tiempos para las diferentes tareas a desarrollar.

- **Desarrollo**

  3. Diseño. (Diseño Preliminar y Diseño Detallado)

  Se encuentran cuatro fases diferenciadas:

  - definición de las estructuras de datos
  - de la arquitectura del software
  - de los algoritmos que desarrollan procedimientos y
  - de las interfaces.

  El objetivo del diseño es la descomposición del sistema en elementos más pequeños (módulos) que a su vez puedan ser descompuestos sucesivamente en otros más sencillos fácilmente programables.

  4. Programación.

  Es el paso del diseño a un lenguaje capaz de ser interpretado por el ordenador. La codificación y generación de interfaces puede automatizarse mediante el empleo de lenguajes de cuarta generación y herramientas CASE.

  5. Prueba.

  Permite comprobar la lógica interna de los programas desarrollados. Se prueba cada uno de ellos comprobando que se producen las respuestas esperadas a los datos introducidos.

- **Mantenimiento del sistema**

  Una vez que el software está en funcionamiento, surgirán cambios y modificaciones debidos a errores en la programación, la adaptación del sistema a nuevas circunstancias y las mejoras que se deseen introducir para obtener utilidades no especificadas o deseadas en un primer momento en el desarrollo del sistema.

Algunas características de este ciclo son:

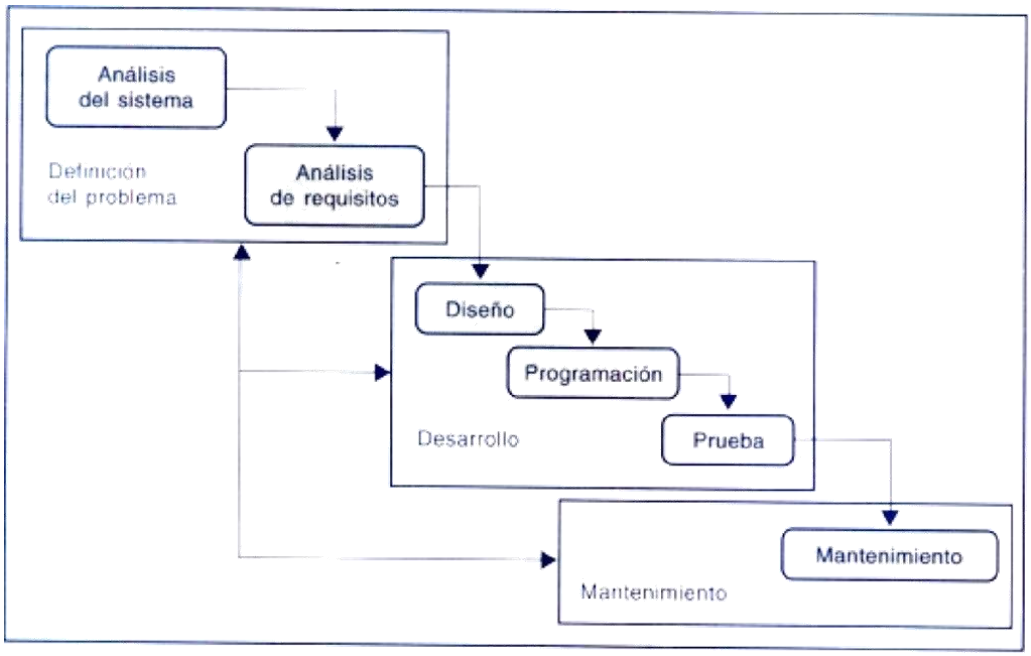
1. Cada fase empieza cuando se ha terminado la fase anterior

2. Para pasar de una fase a otra es necesario conseguir todos los objetivos de la etapa previa
3. Ayuda a prevenir que se sobrepasen las fechas de entrega y los costes esperados.
4. Al final de cada fase el personal técnico y los usuarios tienen la oportunidad de revisar el progreso del proyecto.

Algunas de las críticas del modelo en cascada son:

- No refleja el proceso «real» de desarrollo de software. Los proyectos reales raramente siguen este flujo secuencial, puesto que siempre hay iteraciones que vuelven a fases anteriores y las modifican, cambiando las salidas que éstas producían.
- Se tarda mucho tiempo en pasar por todo el ciclo, dado que hasta que no se finalice una fase no se pasa a la siguiente.
- No da buena impresión al cliente utilizar software con problemas, por lo que mientras que no esté completamente probado, no se entregarán copias del software en desarrollo, ya que no puede ser utilizado en actividades reales.

En la figura se aprecian las diferentes fases del ciclo de vida de una aplicación, y las iteraciones producidas por fases posteriores, que obligarán a modificaciones en las salidas de las fases anteriores.



Modelo en Cascada

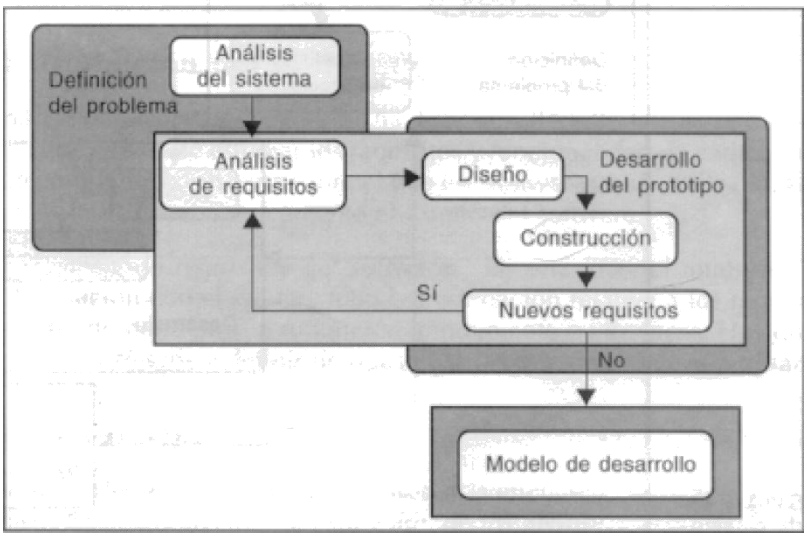
B. Construcción de prototipos

El modelo de prototipos se utiliza **cuando se tienen los objetivos generales del sistema pero falta definición de los requisitos de entrada, proceso y/o salida.**
Ha de estar claro por el cliente que el empleo del modelo de prototipos conlleva un alargamiento en el tiempo de desarrollo del software, ya que no se entregará el prototipo sino el sistema desarrollado, a partir de los requerimientos obtenidos.

Una vez desarrollado el prototipo es necesaria la interacción del equipo de desarrollo con el cliente. La utilización del prototipo producirá los datos necesarios para refinar el diseño a la vista de los requerimientos aportados, modificándose el prototipo sucesivamente.

Las fases del modelo de prototipos son similares a las del ciclo clásico: análisis del sistema y de requisitos, diseño y construcción del prototipo, aportación de nuevos requisitos al prototipo, nuevo diseño y reconstrucción del mismo, hasta que se defina como producto ya terminado.

El modelo de prototipos permite establecer los requisitos necesarios para el desarrollo del software, pero, por su modo de construcción, no es el producto definitivo que se ha de entregar al cliente. Una vez se han conseguido los requisitos, utilizando el ciclo de vida u otro modelo de desarrollo, se genera el software esperado por el cliente. En la figura siguiente pueden verse las diferentes fases del modelo de construcción de prototipos; la incorporación de nuevos requisitos permite realizar diversas iteraciones hasta conseguir un prototipo que cumpla todas las especificaciones deseadas por el cliente. En ese momento se comenzará el diseño del sistema utilizando otro modelo de desarrollo.

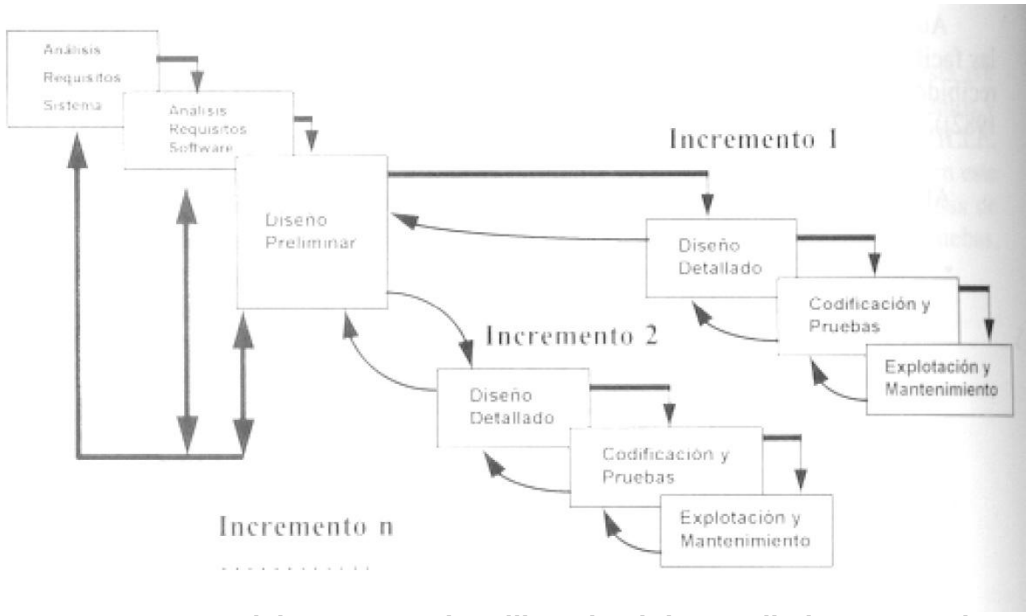


Modelo de prototipos

C. Modelo Incremental

En el modelo incremental se va creando el sistema software añadiendo componentes funcionales al sistema.
El modelo incremental se ajusta a **entornos de alta incertidumbre**, por no tener la necesidad de poseer un conjunto exhaustivo de requisitos, especificaciones, diseños, etc, al comenzar el sistema, ya que cada refinamiento amplia los requisitos y las especificaciones derivadas de la fase anterior.

Aunque permite el cambio continuo de requisitos, aún existe el problema de determinar si los requisitos propuestos son válidos. Los errores en los requisitos se detectan tarde y su corrección resulta tan costosa como en el modelo en cascada.

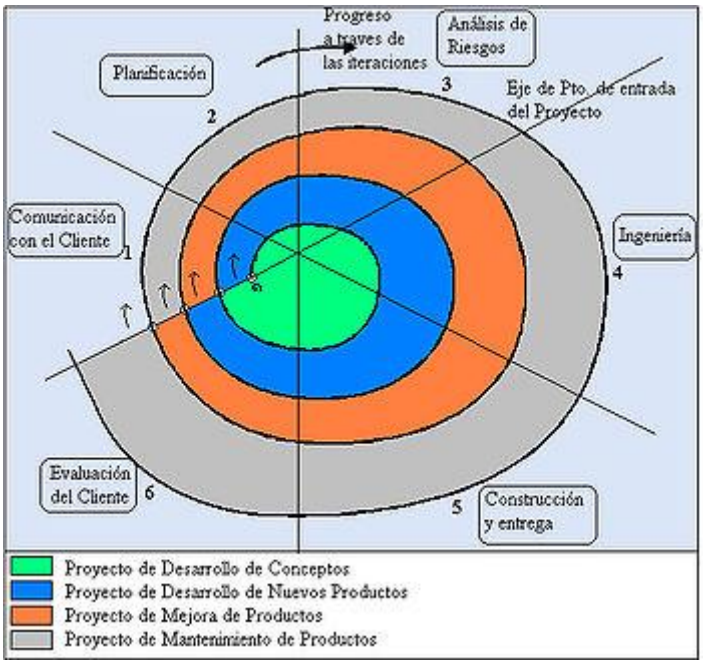


Modelo en cascada utilizando el desarrollo incremental

D. El modelo en espiral

Es un modelo evolutivo que conjuga la naturaleza iterativa del modelo MCP con los aspectos controlados y sistemáticos del Modelo Cascada. En el modelo Espiral el software se construye en una serie de versiones incrementales. En las primeras iteraciones la versión incremental podría ser un modelo en papel o bien un prototipo. En las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado.

El modelo se divide en un número de Actividades de marco de trabajo, llamadas «**regiones de tareas**». En la figura 6 se muestra el esquema de un Modelo Espiral con 6 regiones.



Modelo espiral para el ciclo de vida del software.

Las regiones definidas en el modelo de la figura son:

- Región 1 - Tareas requeridas para establecer la comunicación entre el cliente y el desarrollador.
- Región 2 - Tareas inherentes a la definición de los recursos, tiempo y otra información relacionada con el proyecto.
- Región 3 - Tareas necesarias para evaluar los riesgos técnicos y de gestión del proyecto.
- Región 4 - Tareas para construir una o más *representaciones* de la aplicación software.
- Región 5 - Tareas para construir la aplicación, instalarla, probarla y proporcionar soporte al usuario o cliente (Ej. documentación y práctica).
- Región 6 - Tareas para obtener la reacción del cliente, según la evaluación de lo creado e instalado en los ciclos anteriores.

Al inicio del ciclo, o proceso evolutivo, el equipo de ingeniería gira alrededor del espiral (metafóricamente hablando) comenzando por el centro (marcado con ⊙) y en el sentido indicado; el primer circuito de la espiral puede producir el desarrollo de una especificación del producto; los pasos siguientes podrían generar un prototipo y progresivamente versiones más sofisticadas del software.

Cada paso por la región de planificación provoca ajustes en el plan del proyecto; el coste y planificación se realimentan en función de la evaluación del cliente. El gestor de proyectos debe ajustar el número de iteraciones requeridas para completar el desarrollo.

D. Modelo genérico para desarrollo de Sistemas Orientados a Objetos

Sin entrar en detalles, se puede decir que los modelos orientados a objetos se caracterizan por:

- ✓ La eliminación de fronteras entre fases, ya que debido a la naturaleza iterativa del desarrollo orientado al objeto, estas fronteras se difuminan cada vez más.
- ✓ Una nueva forma de concebir los lenguajes de programación y su uso, ya que se incorporan bibliotecas de clases y otros componentes reutilizables.
- ✓ Un alto grado de iteración y solapamiento, lo que lleva a una forma de trabajo muy dinámica.

En tecnología de objetos se propone seguir un desarrollo iterativo e incremental. Es iterativo porque las tareas de cada fase se llevan a cabo de forma iterativa, a la vez que existe un ciclo de desarrollo análisis-diseño-implementación-análisis que permite hacer evolucionar el sistema.Por lo que respecta al desarrollo incremental, el sistema se divide en un conjunto de particiones, cada una de las cuales se desarrolla de manera completa, hasta que se finaliza el sistema.

De esta forma las actividades de validación, verificación y calidad se pueden realizar, para cada iteración de cada fase de cada incremento en el desarrollo del sistema, es decir, de forma continuada.

En realidad se pueden combinar los modelos tradicionales de ciclo de vida con los más modernos, "*reconciliando así la necesidad de creatividad e innovación con el requisito de prácticas de gestión más controladas"*[BOOCH, 1994]. Booch propone distinguir el "*microproceso"* del *"macroproceso".* El microproceso puede seguir cualquiera de los ciclos de vida más modernos, ya que sirve de marco para el desarrollo interactivo y es más informal. Mientras que el macroproceso está más cercano a un modelo en cascada con el fin de controlar el microproceso de una manera formal.

La ventaja principal de estos modelos es que permiten realizar entregas de sistemas que son operativos cada dos o tres meses, para recibir retroalimentación del cliente lo antes posible e ir adaptando la aplicación según cambian las necesidades y se refinan los requisitos.

PARTE 3: Lenguajes de Programación

3.1. El papel de los lenguajes de programación

Inicialmente los lenguajes se proyectaban para ejecutar programas con eficiencia. A mediados de los años sesenta la programación cambia:

- Las máquinas son menos costosas y aumentan los costos de programación.
- Surge la necesidad de trasladar programas de unos sistemas a otros.
- El mantenimiento del producto consume mayores recursos de cómputo.
- La tarea del lenguaje de alto nivel es la de facilitar el desarrollo de programas correctos para resolver problemas en alguna área de aplicación dada.

Los lenguajes de programación evolucionan o dejan de usarse. Influencias que obligan a la revisión del los lenguajes:

- Capacidad de las computadoras.
- Aplicaciones: Los requerimientos de nuevas áreas de aplicación afectan los diseños de nuevos lenguajes y las revisiones y ampliaciones de los más antiguos.
- Métodos de programación.
- Métodos de implementación.
- Estudios teóricos.
- Estandarización.

Atributos de un buen lenguaje

- **Claridad, sencillez y unidad** (legibilidad): La sintaxis del lenguaje afecta la facilidad con la que un programa se puede escribir, por a prueba, y más tarde entender y modificar.
- **Ortogonalidad**: Capacidad para combinar varias características de un lenguaje en todas las combinaciones posibles, de manera que todas ellas tengan significado.
- **Naturalidad para la aplicació**n: La sintaxis del programa debe permitir que la estructura del programa refleje la estructura lógica subyacente.
- **Apoyo para la abstracción**: Una parte importante de la tarea del programador es proyectar las abstracciones adecuadas para la solución del problema y luego implementar esas abstracciones empleando las capacidades más primitivas que provee el lenguaje de programación mismo.
- **Facilidad para verificar programas**: La sencillez de la estructura semántica y sintáctica ayuda a simplificar la verificación de programas.
- **Entorno de programación**: Facilita el trabajo con un lenguaje técnicamente débil en comparación con un lenguaje más fuerte con poco apoyo externo.
- **Portabilidad de programas**
- **Costo de uso**:
    1. Costo de ejecución del programa.
    2. Costo de traducción de programas.
    3. Costo de creación, prueba y uso de programas.
    4. Costo de mantenimiento de los programas: costo total del ciclo de vida.

3.2. Efectos de los entornos sobre los lenguajes

Cuatro clases generales de entornos objetivos cubren casi todas las aplicaciones de programación:

- de procesamiento por lotes,

- interactivo,
- de sistema empotrado, y
- de programación (entorno interactivo).

Cada uno plantea distintos requerimientos sobre los lenguajes adaptados a esos entornos.

3.2.1. Entornos de procesamiento por lotes

El más simple entorno operativo se compone sólo de archivos externos de datos. Un programa toma un cierto conjunto de archivos de datos como entrada, procesa los datos y produce un conjunto de archivos de datos de salida. El nombre de procesamiento por lotes viene porque los datos de entrada se reúnen en *"lote*s" de archivos y son procesados en lotes por programas.

- Los archivos constituyen la base para casi toda la estructura de E/S.
- Un error que termine la ejecución del programa es aceptable aunque costoso. No es posible la ayuda externa por parte del usuario para manejar o corregir errores de inmediato.
- Carencia de restricciones de regulación de tiempo. No hay recursos para monitorear o afectar directamente la velocidad de ejecución del programa.

3.2.2. Entornos interactivos

El programa interactúa durante su ejecución directa con un usuario en una consola de visualización, enviando alternativamente salidas hacia ésta y recibiendo entradas desde el teclado o ratón (procesadores de texto, hojas de cálculo, juegos, etc.).

- Las características de E/S interactivas son diferentes de las operaciones ordinarias con archivos.
- El programa debe ser capaz de gestionar el manejo de errores. La terminación del programa como respuesta a un error no es ordinariamente aceptable (a diferencia del procesamiento por lotes).
- Los programas interactivos deben utilizar con frecuencia algún concepto de restricciones de tiempo.
- El concepto de programa principal suele estar ausente. En su lugar, el programa se compone de un conjunto de subprogramas y el usuario introduce el "programa principal como una serie de comandos en el terminal.

3.2.3. Entornos de sistemas incrustados (empotrados)

Un sistema de computadora que se usa para controlar parte de un sistema más grande como una planta industrial, una aeronave, etc., se conoce con el nombre de *sistema de computadora incrustad*o. El fallo de una aplicación empotrada puede poner en peligro la vida. La seguridad de funcionamiento y corrección son atributos principales.

- Suelen operar sin un sistema operativo subyacente y sin archivos de entorno y dispositivos de E/S usuales. El programa debe interactuar directamente con la máquina.
- El manejo de errores tiene gran importancia. Cada programa debe estar preparado para manejar todos los errores en forma interna, adoptando acciones apropiadas para recuperarse y continuar. La interrupción del programa no es aceptable y no hay un usuario en el entorno que pueda proporcionar la corrección interactiva del error.
- Operan en tiempo real, donde la respuesta las entradas debe producirse en intervalos de tiempo restringidos.
- Suele ser un sistema distribuido, compuesto por más de una computadora.
- Una vez iniciadas las tareas, se ejecutan por lo común de forma simultánea e indefinida.

3.2.4. Entornos de programación

Es el entorno en el cual los programas se crean y se ponen a prueba. Consiste en un conjunto de herramientas (editor, depurador, verificador, generadores de datos de prueba, etc.) de apoyo y un lenguaje para invocarlas.

Al compilar por separado cada subprograma el compilador necesita información de:

- La especificación del número, orden y tipo de parámetros.
- La declaración de tipo de datos.
- La definición de un tipo de datos (para la declaración local de variables).

Un problema común, es encontrar, durante el ensamblado del programa final completo, que varios subprogramas y otras unidades de programa tienen nombres (de variables) iguales. Métodos para evitar este problema:

- Todo nombre compartido debe ser único. Se deben usar convenciones para la asignación de nombres desde un principio.
- Definir, en el lenguaje, reglas de ámbito, para ocultar nombre.
- Los nombres se pueden conocer agregando explícitamente sus definiciones desde una biblioteca externa (herencia en POO).
- Características que ayudan a poner a prueba y depurar programas.
- Características para rastreo de ejecución.
- Puntos de interrupción. Cuando se alcanza un punto de interrupción durante la ejecución del programa, la misma se interrumpe y el control se traslada al programador en un terminal.
- Asertos: expresan relaciones que deben cumplirse entre los valores de las variables en ese punto del programa.

3.3. Código fuente, código objeto o ejecutable

El **código fuente** de un programa informático (o software) es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

El código fuente de un programa está escrito por un programador en algún lenguaje de programación, pero en este primer estado no es directamente ejecutable por la computadora, sino que debe ser traducido a otro lenguaje (el lenguaje máquina o código objeto) que sí pueda ser ejecutado por el hardware de la computadora. Para esta traducción se usan los llamados compiladores, ensambladores, intérpretes y otros sistemas de traducción.

El término código fuente también se usa para hacer referencia al código fuente de otros elementos del software, como por ejemplo el código fuente de una página web que está escrito en el lenguaje de marcado HTML o en Javascript u otros lenguajes de programación web y que es posteriormente ejecutado por el navegador web para visualizar dicha página cuando es visitada.

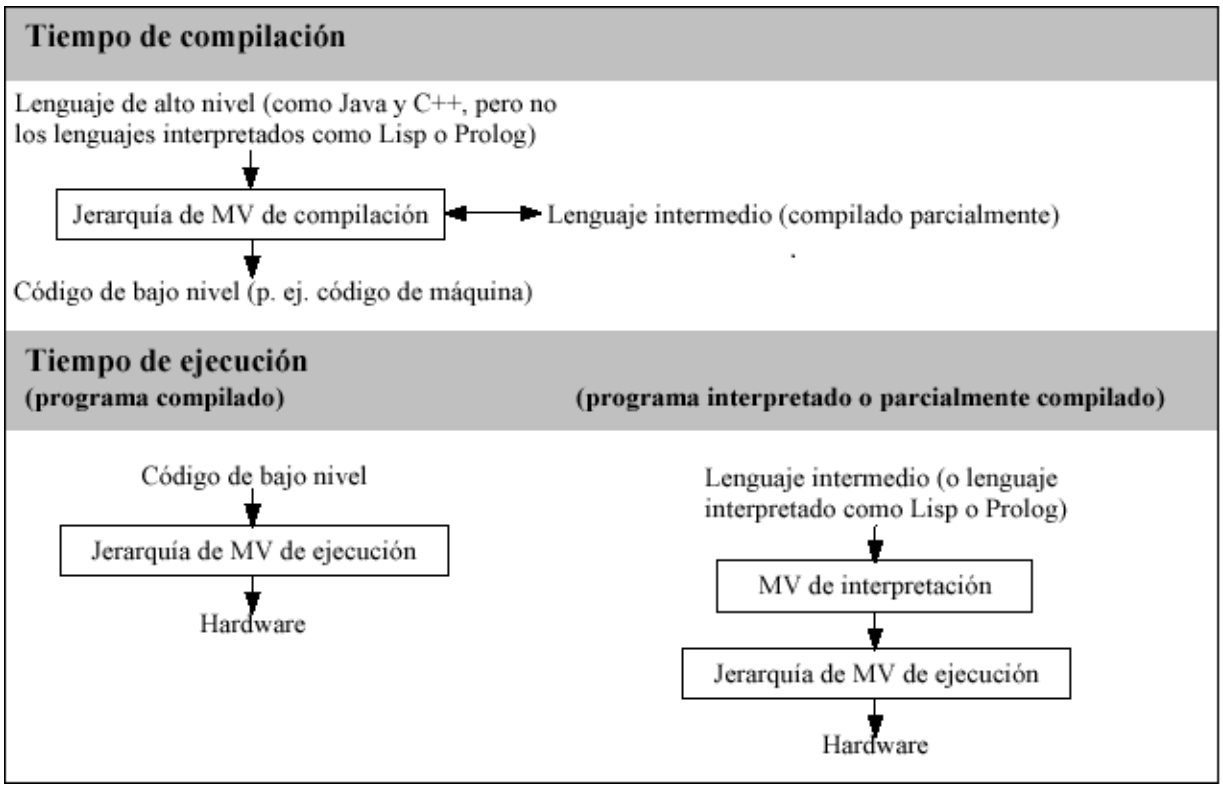
El área de la informática que se dedica a la creación de programas y, por tanto a la creación de su código fuente, es la programación.

3.4. Máquinas virtuales

Aunque hay muchos rasgos que se podrían destacar como candidatos para formar parte de una definición del concepto de máquina virtual (a partir de ahora MV), una MV se puede definir sencillamente como una capa de abstracción que separa el funcionamiento de un ordenador de su hardware. Además, las MV se dividen en *abstractas* o teóricas, como sería la máquina de Turing (el primer ejemplo de una MV), y *concretas* o prácticas (a las que se quiere normalmente se hace referencia al hablar de MV). En esta sección se tratarán las MV concretas que, como se verá, son capas de software que juegan un papel relevante tanto en el funcionamiento de los lenguajes compilados como interpretados.

Se pueden identificar dos tipos de MV concretas:

- las que juegan un papel en la preparación de un programa para su ejecución (tiempo de compilación) y
- las que permiten la ejecución de dicho programa. La figura muestra la diferencia entre los dos tipos:



El papel de las máquinas virtuales en la compilación y la ejecución de un programa

Las MV suelen aparecer en una jerarquía. Usamos la jerarquía de MV de compilación, por ejemplo, cada vez que invocamos el compilador de C++. Y dos ejemplos muy comunes del papel de una MV en la jerarquía de MV de ejecución son PostScript (que define una MV para una impresora) y MSDOS bajo MS Windows (que define una MV que permite que antiguos programas de DOS funcionen bajo Windows).

PostScript es una MV que incluye el estado gráfico, la ruta actual, el estado del diccionario y, aunque sea bastante complejo, presenta una visión de alto nivel de muchos tipos distintos de impresoras, aislando las características de las impresoras de las aplicaciones que las usan.

A la hora de implementar un lenguaje de programación, las estructuras de datos y algoritmos utilizados en la ejecución de un programa (es decir, la sintaxis y semántica de cada estructura) definen las MV (implícitamente) para este lenguaje. Y las decisiones que toma el desarrollador afectan tanto a la traducción como a la ejecución de los programas escritos en este lenguaje.

La realidad es que el desarrollador de un lenguaje suele implementar las MV (de compilación y/o ejecución) de su lenguaje en términos de otras MV ya existentes. Y en el futuro, un programador de aplicaciones utilizará las MV implementadas por el desarrollador del lenguaje para producir programas que a su vez puedan servir como MV para otras aplicaciones, etc. La conclusión, como puede verse, es que una MV no suele existir aislada, sino como parte de una jerarquía de MV. A continuación se va a ver el papel de las jerarquías de MV en el tiempo de compilación y de ejecución.

Además del proceso de compilación, la traducción completa de los programas de alto nivel (previa a su ejecución) en una forma que corre sobre la máquina, también existe otro proceso, que se llama **interpretación**, que tiene más que ver con el papel de las MV en el tiempo de ejecución que en el tiempo de compilación.

Además de los lenguajes compilados “completamente”, hay otros que son compilados “parcialmente” en el sentido de que terminan el proceso de compilación en un código (o lenguaje) intermedio, en vez de en código máquina (lenguajes intermedios).

3.5. Lenguajes intermedios

Un lenguaje intermedio se puede definir como una manera de representar procedimientos y estructuras de datos que sirva como entrada para una MV en alguna parte de su jerarquía, entre el lenguaje de entrada (el nivel más alto) y el código ejecutado en la máquina (el nivel más bajo) tanto en el tiempo de compilación como en el de ejecución.

Para considerar el papel de los lenguajes intermedios y sus ventajas y desventajas, conviene destacar la diferencia entre la traducción de un lenguaje de alto nivel a código máquina anteriormente a su ejecución (su compilación) y su interpretación, es decir, la conversión de cada instrucción del lenguaje a código máquina y su ejecución, una por una, al ejecutar el programa.

Este proceso se realiza a través de una MV de interpretación que simula un ordenador cuyo código máquina es el lenguaje de alto nivel que está siendo interpretado. Y típicamente, esta MV se construye a través de un conjunto de programas de código máquina que representa los algoritmos y estructuras de datos necesarios para la ejecución de las instrucciones del lenguaje de alto nivel.

Hay ventajas y desventajas en cada manera de convertir los lenguajes de alto nivel a código máquina, que se pueden resumir así:

Compilación		
Ventajas	1.	No hay que repetir la conversión de la misma instrucción a código máquina cada vez que aparece.
	2.	Los programas corren muy rápido.
Desventajas	1.	Pérdida de claridad e información sobre el programa.
	2.	Dificultad en localizar la fuente exacta de error.
Ejemplos → Ada, C, C++, FORTRAN, Pascal		
Interpretación		
Ventajas	1.	No hay pérdida de claridad ni de información sobre un programa ni sobre donde están los errores.
	2.	No hay que decodificar código que no se va a ejecutar.
	3.	El código es típicamente más compacto.
Desventajas	1.	Los programas corren mucho más lentamente – se paga el coste de decodificar cada instrucción.
Ejemplos → HTML, Lisp, ML, Perl, Postscript, Prolog, Smalltalk		

Estos dos casos representan los dos extremos porque existe también lo que se llama la compilación parcial, que es una mezcla de los dos enfoques, donde se compila el lenguaje de alto nivel a un lenguaje intermedio (más cerca de las estructuras presentes en el código máquina que las del código fuente) y luego se interpreta este lenguaje al ejecutar el programa.

Como puede imaginarse, esta técnica combina las ventajas y desventajas de los dos enfoques anteriores. Un ejemplo de esta combinación existe en el lenguaje de programación Java y su entorno.

Entre otras cosas, Java empezó con la idea de liberar al programador de las dificultades de portar su aplicación a nuevas plataformas lo cual, si el programa está muy vinculado a algún aspecto del sistema operativo donde fue escrito, podría ser muy difícil.

Se compilará el código fuente de Java a un código byte *(bytecod*e) antes de ejecutarlo. Y a la hora de correr el programa, este código, como lenguaje intermedio, sería el lenguaje de entrada para una MV, que con un conjunto de librerías (el entorno de ejecución de Java, *Java Runtime* o JRE), la interpretaría para su ejecución.

Por lo tanto, este bytecode podría correr en cualquier hardware donde haya una versión del JRE disponible. Como este bytecode está más cerca del nivel de máquina que de un lenguaje de alto nivel, los programas correrán más rápidamente que los programas completamente interpretados, aunque más despacio que los programas previamente compilados al código máquina.

Como se puede ver en la figura anterior, tanto los programas compilados parcialmente a un lenguaje intermedio (como Java) como los programas escritos en lenguajes de alto nivel que se interpretan (como Lisp) requieren una MV para interpretar el programa. La principal ventaja del lenguaje intermedio en este caso es su proximidad al nivel del código máquina, en el sentido de que supone menos trabajo a la hora de ejecutarlo y, por lo tanto, los programas corren más rápidamente que los puramente interpretados.

3.5. La máquina virtual de Java como ejemplo de una MV

La MV de Java es una **máquina de pila**. Las instrucciones interpretadas por ella manipulan datos almacenados como elementos en una pila. El contenido ejecutable de un archivo de bytecodes contiene un vector de instrucciones bytecode para cada método. Los bytecodes son instrucciones para la MV, que tiene algunos registros de variables locales y una pila para la evaluación de expresiones.

Las primeras variables locales son inicializadas con los parámetros actuales. Cada variable local o elemento de la pila es una palabra que corresponde a un entero de 32 bits, a un punto flotante o a una referencia a objeto (puntero). Para puntos flotantes dobles y enteros largos se utilizan dos huecos de la pila.

Los huecos de la pila no están relacionados con un tipo de datos, es decir, en algún punto un hueco podría contener un valor entero y en otro, el mismo hueco podría contener una referencia a un objeto. Sin embargo, no se puede almacenar un entero en un hueco y luego recuperarlo reinterpretándolo como si fuera una referencia a un objeto. Aún más, en cualquier punto del programa, el contenido de cada hueco está asociado con un único tipo de datos que puede ser determinado usando un flujo estático de datos.

El tipo de datos podría ser “no asignado”, con lo cual no se permite leer el valor del hueco. Estas restricciones son parte del modelo de seguridad de Java y se ven reforzadas por el verificador de bytecodes.

El código interpretado es generalmente más lento que un programa escrito en un lenguaje compilado, y Java no es distinto en este aspecto. Se han señalado muchas posibilidades para mejorar el rendimiento de los intérpretes.

Una muy común hoy en día es incluir un compilador relativamente simple en el tiempo de ejecución de la MV. Es decir, en vez de interpretar los bytecodes del programa una y otra vez, se compilan una sola vez “al instante” en el interior de la MV, y la representación compilada de los métodos que corresponden al programa es ejecutada al efectuar una llamada. Esto es conocido como un compilador al instante (o JIT, *Just In Tim*e).