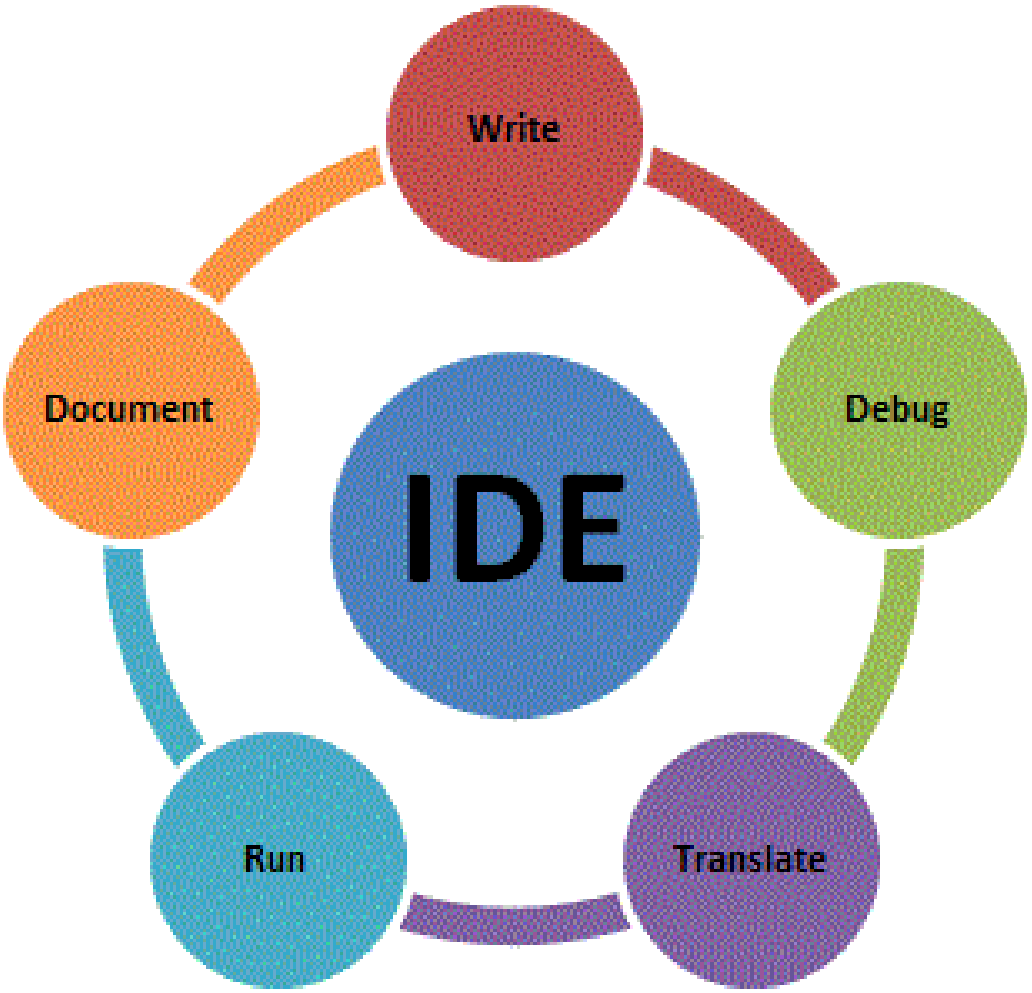


UNIDAD 1. DESARROLLO SOFTWARE

PROFESORA: M.ª José Chaves Delgado
DEPARTAMENTO DE INFORMÁTICA Y COMUNICACIÓN
IES VELÁZQUEZ (SEVILLA)

1. INTRODUCCIÓN

La economía de todo el mundo depende cada vez más de sistemas software y cada vez hay más tareas que se controlan por software. Actualmente los gastos en software son una parte importante del PIB de todos los países. Por lo tanto no se puede desarrollar software como se hacía en los comienzos de la programación de forma artesanal.

La ingeniería del software son todas las teorías, métodos y herramientas que utilizamos en el desarrollo del software.

Actualmente los costes del software son superiores a los de hardware y se mueve más negocio en el mantenimiento de productos software que en su desarrollo, especialmente en sistemas ”long-life”. La ingeniería del software trata de optimizar estos costes.

En este tema vamos a estudiar no sólo en qué consiste la ingeniería de software sino una serie de conceptos importantes para su desarrollo, como el concepto de programa informático, código fuente, código objeto y código ejecutable, máquinas virtuales, los distintos tipos de lenguajes de programación, las características de los lenguajes más difundidos, las fases del desarrollo de una aplicación, entre otras.

Veremos también el proceso de obtención de código ejecutable a partir del código fuente y las herramientas implicadas.

2. INTRODUCCIÓN A LA INGENERÍA DE SOFTWARE

2.1 PRODUCTOS SOFTWARE

Es de sobra conocido que el ordenador se compone de dos partes diferenciadas: hardware y software. El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.

Existen multitud de productos software. Una primera clasificación podría ser atendiendo a la finalidad:

- Software genérico: Sistemas stand-alone vendidos en el mercado a cualquier cliente que lo quiera. Ej: Paquetes ofimáticos, software gráfico, herramientas de gestión de proyectos, CAD, facturación,…

- Software a medida: Productos adaptados a las necesidades específicas de un cliente. Ej: Sistema de control de tráfico aéreo, sistema de gestión hidroeléctrico,…

-

2.2 ESPECIFICACIONES DE SOFTWARE

Por especificaciones de software nos referiremos a qué deber hacer el software, para qué lo desarrollamos. Dependerá del tipo de software según la clasificación anterior:

- Software genérico: Las características las decide el desarrollador del producto, teniendo en cuenta que cuanto más configurable sea su producto, más clientes potenciales.

Software a medida: Las características y su importancia las decide el cliente, así como los cambios o nuevas necesidades a lo largo de su vida útil.

2.3 EFICIENCIA VERSUS EFICACIA

Un producto (software) es **eficaz** cuando satisface las necesidades para las que ha sido desarrollado.

Un producto (software) es **eficiente** cuando además de ser eficaz, consume los mínimos recursos posibles (ej: potencia de cálculo, memoria, tiempo de usuario,…)

La **meta** de todo desarrollo de un producto es que este sea eficiente. La I.S. trata de lograr esto con un producto software aplicando un método.

2.4 IMPORTANCIA DE LA INGENIERÍA DE SOFTWARE

La sociedad en general requiere cada vez de más y mejores sistemas software. Nosotros necesitamos producirlos de manera rentable y rápida.

Un proyecto grande es más barato si desde el inicio se utilizan las técnicas y métodos de la I.S.; en vez de iniciarlo como si se tratase de un proyecto personal. Aunque a veces la dificultad radica en preveer el alcance del proyecto con todas sus necesidades, requisitos y especificaciones.

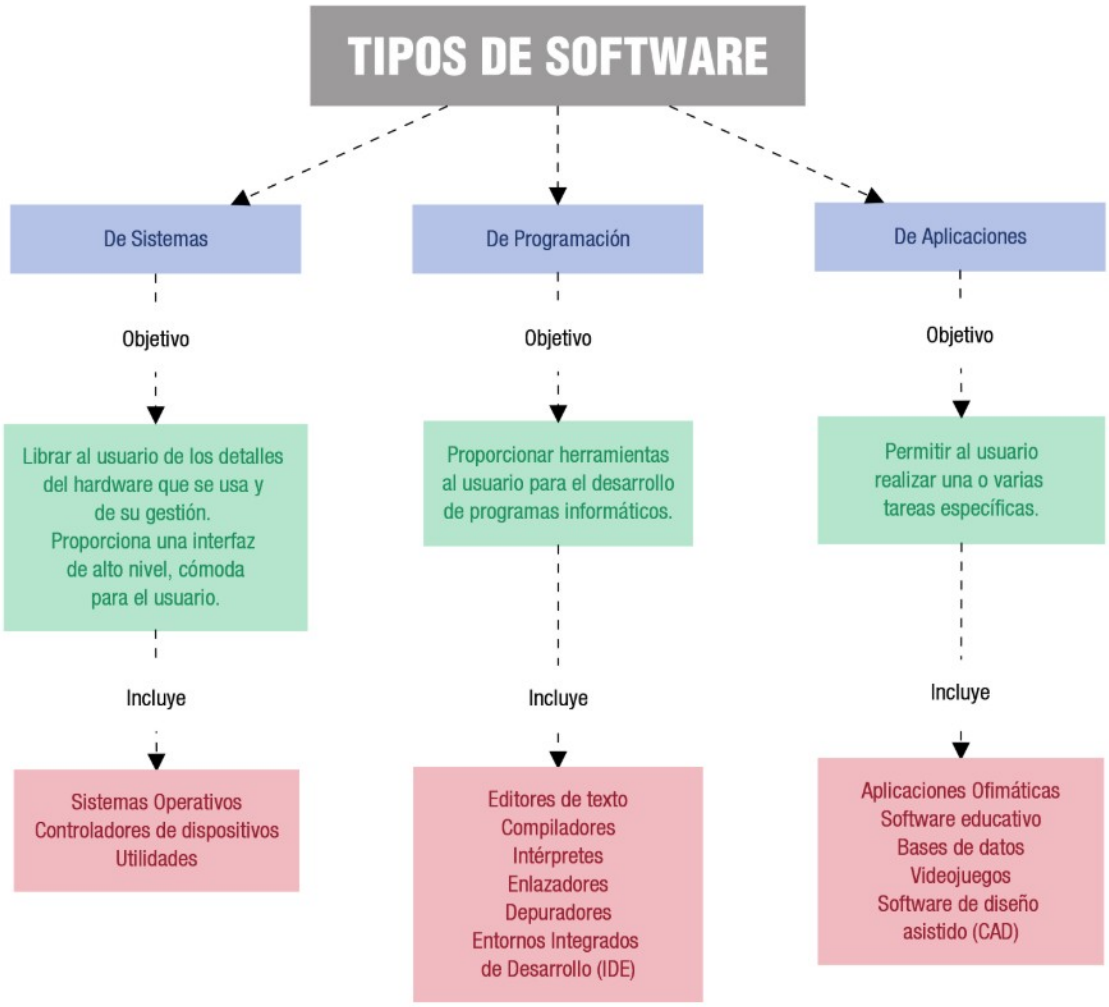
3. DESARROLLO DE SOFTWARE

3.1 SOFTWARE Y PROGRAMA. TIPOS DE SOFTWARE

Es de sobra conocido que le ordenador se compone de dos partes diferenciadas:hardware y software. El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.

Un programa informático es un conjunto de instrucciones que una vez ejecutadas realizará una o varias tareas en un ordenador. Al conjunto general de programas, se le denomina software, que más genéricamente se refiere al equipamento lógico o osporte lógico de una computadora digital.

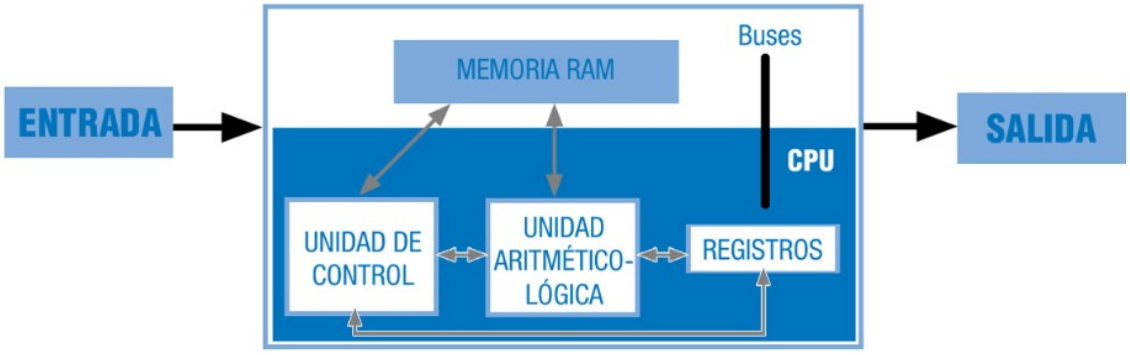
En informática, una aplicación es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo. Esto lo diferencia principalmente de otros tipos de programas como los sistemas operativos (que hacen funcionar al ordenador), las utilidades (que realizan tareas de mantenimiento o de uso general) y los lenguajes de programación (con el cual se crean los programas informáticos).



3.2 INTERACCIÓN CON EL SISTEMA

El conjunto de dispositivos físicos que conforman un ordenador se denomina hardware y existe una relación indisoluble entre hardware y software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione.

La primera arquitectura hardware con programa almacendado se estableció en 1946 por John Von Neumann.



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

a) Desde el punto de vista del sistema operativo

El sistema operativo es el encargado de coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están en ejecución en un momento dado.

Todas las aplicaciones necesitan recursos hardware durante la ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de entrada/salida, etc) Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera “oculta” para las aplicaciones (y para el usuario).

b) Desde el punto de vista de las aplicaciones

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

Hay multitud de lenguajes de programación diferentes. Sin embargo, todos tiene algo en común: estar escritos con secuencias de un idioma que el ser humano puede aprender fácilmente. Por otra parte, el hardware de un ordenador solo es capaz de interpretar señales eléctricas que, en informática, se traducen en secuencias de 0 y 1 (código binario). ¿Cómo será capaz el ordenador de “entender” algo escrito en un lenguaje que no es el suyo? Como veremos a lo largo de la unidad, tendrá que pasar algo para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación

3.3 LENGUAJES DE PROGRAMACIÓN

Los programas informáticos están escritos usando algún lenguaje de programación, es decir, un conjunto de instrucciones, operadores y reglas de sintaxis y semánticas, que se ponen a disposición del programador para que éste pueda comunicarse con los dispositivos hardware y software existentes.

En un principio, todos los programas eran creados por el único códiog que el ordenador entendía, el código máquina, es decir, un conjunto de ceros y unos (código binario). Este método de programación convertía la tarea de programación en una labor sumamente tediosa,hasta que se tomá la solución de establecer un nombre a las secunencias de programación más frecuentes, estableciéndolas en posiciones de memoria concretas, a cada una de estas secuencias nominadas se las llamó instrucciones, y al conjunto de dichas instrucciones, lenguaje ensamblador.

Más adelante, y con el comienzo del uso de los ordenadores en ámbito científico, principalmente en física y química, donde el conocimiento de informática era muy bajo se pudo comprobar que el uso de lenguaje ensamblador era sumamente complicado. Como un modo de facilitar la tarea de programar nace el concepto de lenguaje de alto nivel con FORTRAN (FORmula TRANslation).

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

3.4 CLASIFICACIÓN Y CARACTERÍSTICAS

La cantidad de lenguajes de programación es abrumadora, cada uno ocn unas características y objetivos determinados. Los criterios para clasificar los lenguajes de programación se corresponden con sus características principales.

Se pueden clasificar mediante una gran cantidad de criterios, pero para el curso, tomaremos con mayor relevancia los tres siguientes: nivel de abstracción, forma de ejecución y paradigma de uso.

3.4.1 Nivel de abstracción

Llamamos nivel de abstracción al modo en que los lenguajes se alejan del código máquina y se acercan al lenguaje humano. Cuanto más alejado esté del código máquina, mayor será su nivel de abstracción como lenguaje.

- **Lenguajes de bajo nivel**
  - Primera generación: sólo hay un lenguaje de primera generación, el lenguaje máquina. Cadenas interminables de código binario.
- **Lenguaje de nivel medio**
  - Segunda generación: en esta generación entrarían el lenguaje ensamblador como máximo representante. Se trata de lenguajes con instrucciones para realizar operaciones sencillas con datos simples o posiciones de memoria.
- **Lenguajes de alto nivel**
  - Tercera generación: la gran mayoría de los lenguajes de programación que se utilizan hoy en día pertenecen a este nivel de abstracción. En este nivel se encuadran tanto los lenguajes procedimentales como los lenguajes orientados a objetos, ya que en ambos casos permiten un alto nivel de abstracción y una forma de programar mucho más entendible e intuitiva, donde algunas instrucciones parecen ser una instrucción directa del lenguaje humano.
  - Cuarta generación: lenguajes creados con un propósito específico, lo cual permite reducir la cantidad de líneas de código que se tendrían que usar con un lenguaje de tercera generación. Si por ejemplo tratáramos de resolver un problema matemático complejo con un lenguaje de tercera generación serían necesarias muchas líneas de código, sin embargo, con un lenguaje de cuarta generación tendríamos rutinas para resolverlo de manera más rápida.
  - Quinta generación: son los también llamados lenguajes naturales, y se usan principalmente en el ámbito de la inteligencia artificial y la lógica. Pretenden abstraerse más aún, usando el lenguaje natural con una base de conocimientos que

produce un sistema basado en el conocimiento. Pueden establecer el problema que hay que resolver y las premisas y condiciones que hay que reunir para que la máquina lo resuelva.

3.4.2 Forma de ejecución

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

- **Lenguajes compilados:** un programa traductor (compilador) convierte el código fuente en código objeto y otro programa (enlazador) unirá el código objeto del programa con el código objeto de las librerías necesarias para producir el programa ejecutable. Tanto el compilador como el enlazador dependerán de la arquitectura de la máquina, por lo que el código generado no será multisistema.

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

- **Lenguajes virtuales:** funcionan de manera similar a los lenguajes compilados, pero no es código objeto lo que genera el compilador, sino un conjunto de instrucciones o código que podrán ser interpretadas por cualquier arquitectura que tenga la máquina virtual del lenguaje instalada. Esta máquina virtual ofrece la ventaja de poder ejecutar el código como si fuera multisistema, a cambio de perder velocidad de ejecución el tener que interpretar el código en cada ejecución.

3.4.3 Paradigma de programación

El paradigma de programación es un enfoque particular para la construcción de software. Todos los paradigmas pertenecen a lenguajes de alto nivel, y es común que un lenguaje pueda usar más de un paradigma de programación.

- Paradigma imperativo: describe la programación como una secuencia de instrucciones que cambian el estado del programa, indicando cómo realizar una tarea.
- Paradigma declarativo: especifica o declara un conjunto de premisas y condiciones para indicar qué es lo que hay que hacer y no necesariamente cómo hay que hacerlo. Usando principalmente la lógica.
- Paradigma procedimental: evolución del paradigma imperativo donde el programa se divide en partes más pequeñas llamadas funciones y procedimientos, que pueden comunicarse entre sí. Aplica el concepto “divide y vencerás” para la resolución de problemas complejos.
- Paradigma orientado a objetos: encapsula el estado y las operaciones enobjetos, creando una estructura de clases y objetos que emulan un modelo del mundo real, donde los objetos realizan acciones e interactúan con otros objetos.
- Paradigma funcional: evalúa el problema realizando funciones de manera recursiva, evita declarar datos haciendo hincapié en la composición de funciones y en las interacciones entre ellas.
- Paradigma lógico: define un conjunto de reglas lógicas para ser interpretadas mediante inferencias lógicas. Permite responder preguntas planteadas al sistema para resolver problemas.

En el siguiente cuadro se muestra una clasificación de los principales lenguajes de programación:

Lenguaje	Características
Lenguaje C	Medio nivel, compilado, procedural
Ensamblador	Bajo nivel, compilado, procedural
HTML	Lenguaje de marcas, no es de programación
PHP	Alto nivel, interpretado, procedural
Javascript	Alto nivel, interpretado, procedural
JAVA	Alto nivel, compilado, POO
JSP	Alto nivel, compilado, POO
Prolog	Muy alto nivel, compilado-interpretado, declarativo

4. TIPOS DE CÓDIGO Y PROCESO DE OBTENCIÓN DE EJECUTABLES

Cualquier programa que se quiera ejecutar en una arquitectura determinada necesita ser traducido para poder ser ejecutado (con la excepción del lenguaje máquina). Por lo que, aunque tengamos el código de nuestro programa escrito enel lenguaje de programación escogido, no podrá ser ejecutado a menos que lo traduzcamos a un idioma que nuestra máquina entienda.

4.1 TIPOS DE CÓDIGO (FUENTE, OBJETO Y EJECUTABLE)

El **código fuente** (source code) de un programa informático 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. Cuando programamos, escribimos líneas de código fuente.

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, interpretes y otros sistemas de traducción.

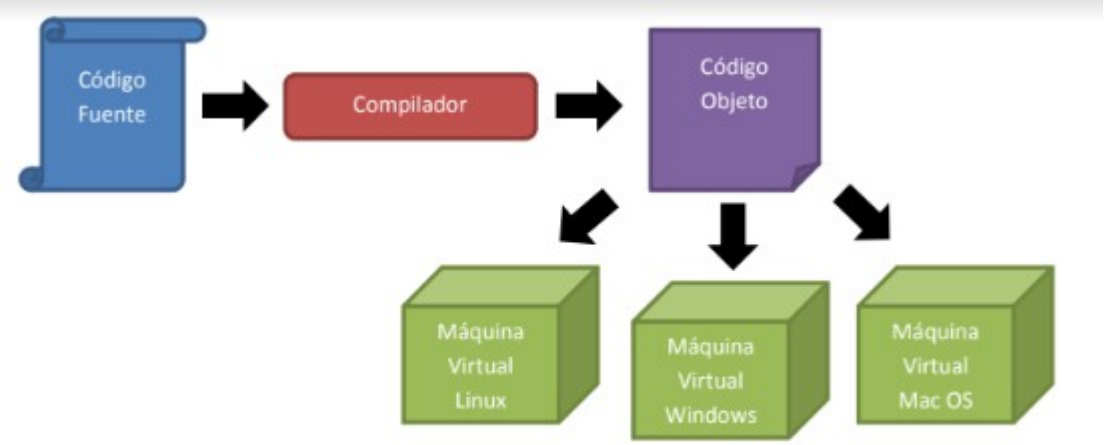
Por último, el código objeto es convertido en **ejecutable** para que pueda ser "entendido" por la máquina para la que fue diseñado.

En el paso de código fuente a código objeto solo se tiene en cuenta relaciones y librerías externas a nuestro código que necesitamos para un correcto funcionamiento de nuestro programa. Una vez las tengamos identificadas, las asimilamos a nuestro código. En cuanto al paso de código objeto a código ejecutable, lo que se asimila a nuestro código son las dependencias de las arquitecturas y sistemas operativos en los que se está trabajando.

Por ello, si hacemos un progrma en C++ en Windows 7, el código objeto será igual que si lo hemos hecho en linux, pero tendrá diferentes ejecutables al tratarse de sistemas operativos distintos.

Buscando una independencia de los lenguajes de programación, del entorno donde sean compilados y del sistema operativo donde van a ser usados, aparecieron las **máquinas virtuales.** Este proceso consiste en crear un programa en un lenguaje que al compilar, creará un código objeto independiente de la máquina. Cuando vayamos a hacer uso del programa, dependiendo de en qué sistema lo esté usando tendrá una interpretación u otra mediante un software específico de casa sistema operativo llamado Maquina Virtual.

El código máquina es el mismo siempre, y al usarlo es cuando hacemos diferenciación gracias a este software especial o máquina virtual.



En el caso de lenguajes interpretados no son necesarios ni los compiladores ni los enlazadores, ya que el código fuente pasará directamente por el intérprete que realizará la ejecución en tiempo real de nuestro código.

4.2 PROCESO DE COMPILACIÓN

El proceso de compilación es una secuencia de varias fases. Cada fase toma su información de entrada de la fase anterior, y cada fase tiene su propia interpretación del código fuente.

1. **Análisis léxico:** se leen los caracteres del programa fuente y se agrupan en cadenas que representan los componentes léxicos. Cada componente léxico es una secuencia lógicamente coherente de caracteres relativa a un identificador, una palabra reservada, un operador o un carácter de puntuación. A la secuencia de caracteres que representa un componente léxico se le llama lexema (o con su nombre en inglés token). En el caso de los identificadores creados por el programador no solo se genera un componente léxico, sino que se genera otro lexema en la tabla de símbolos.

2. **Análisis sintáctica:** los componentes léxicos se agrupan en frases gramaticales que el compilador utiliza para sintetizar la salida.

3. **Análisis semántico:** intenta detectar instrucciones que tengan la estructura sintáctica correcta, pero que no tengan significado para la operación implicada.

4. **Generación de código intermedio:** algunos compiladores generan una representación intermedia explícita del programa fuente, una vez que se han realizado las fases de análisis. Se puede considerar esta operación intermedia como un subprograma para una máquina abstracta. Esta representación intermedia debe tener dos propiedades importantes: debe ser fácil de producir y fácil de traducir al programa objeto.

5. **Optimización de código:** se trata de mejorar el código intermedio, de modo que resulte un código de máquina más rápido de ejecutar.

6. **Generación de código:** constituye la fase final de un compilador. En ella se genera el código objeto que por lo general consiste en código en lenguaje máquina (código relocalizable) o código en lenguaje ensamblador.

7. **Administración de la tabla de símbolos:** una tabla de símbolos es una estructura de datos que contiene un registro por cada identificador. El registro incluye los campos para los atributos del identificador. El administrador de la tabla de símbolos se encarga de manejar los accesos a la tabla de símbolos, en cada una de las etapas de compilación de un programa.

8. **Manejador de errores:** en cada fase del proceso de compilación es posibles encontrar errores. Es conveniente que el tratamiento de los errores se haga de manera centralizada a través de un manejador de errores. De esta forma podrán controlarse más eficientemente los errores encontrados en cada una de las fases de la compilación de un programa.

5. CICLO DE DESARROLLO SOFTWARE

El desarrollo de un software o de un conjunto de aplicaciones pasa por diferentes etapas desde que se produce la necesidad de crear un software hasta que se finaliza y está listo para ser usado por un usuario. Ese conjunto de etapas en el desarrollo del software

responde al concepto de ciclo de vida del software. Las etapas o tareas no son fijas y dependerán del tipo de software, del equipo de personas usados y de las necesidades del cliente el que se usen un conjunto u otro. Es decir, que se use un modelo de ciclo de vida u otro, aunque si se puede decir que existen una serie de tareas recomendadas cuando se realiza el proceso de desarrollo de software.

1. Análisis: define los requisitos de software que hay que desarrollar. Inicialmente esta etapa comienza obteniendo información del cliente por muy diversas vías, entrevistas, reuniones, documentos, etc. Y finaliza con una documentación oficial de requisitos software validada con el propio cliente. En resumen, esta etapa debe contestar a la pregunta ¿Qué debe hacer el sistema?

2. Diseño: en esta fase se define el funcionamiento del sistema a alto nivel, para profundizar y llegar al nivel del diseño de los programas. Dependiendo del modelo se usarán unas técnicas u otras, siendo los diagramas UML una técnica muy extendida en el desarrollo orientado a objetos. Las preguntas que se deben contestar en esta fase son el ¿Cómo debe funcionar el sistema?

3. Codificación: fase en la que se traducirán los programas y algoritmos definidos en la fase anterior a código fuente haciendo uso del lenguaje de programación escogido en la fase de diseño.

4. Pruebas: los elementos programados en la fase anterior se ensamblarán y se comprobará su correcto funcionamiento. Existen distintos tipos de pruebas en función de la persona que las ejecuta y de la fase en que se realizan. La salida de esta fase puede ser una vuelta a cualquiera de las fases anteriores, dado que pueden detectarse errores de análisis, diseño o codificación.

5. Documentación: todo software tiene dos tipos de documentación asociada, la documentación de usuario y la documentación del proyecto, de carácter técnico y funcional. No debe tomarse como una tarea de menor importancia, ya que el éxito de un sistema radica en el buen uso del mismo, algo que será difícil cumplir si no existe una correcta documentación.

6. Explotación: una vez realizado el software se prepara para su distribución, o se implanta en el hardware indicado en la fase de diseño. Esta tarea suele ser llevada a cabo por parte del propio equipo de desarrollo, y una vez realizada el software queda a disposición del usuario para su uso.

7. Mantenimiento: todo software empresarial pasa a una fase de mantenimiento una vez entra en explotación. Esta fase irá en paralelo a la vida útil del sistema, y servirá para realizar correcciones y evoluciones del propio sistema cuando sea necesario.

5.1 DESARROLLO EN CASCADA

Es el modelo más antiguo para desarrollar software, y no por ello poco usado. También es conocido como modelo secuencial, y consiste en la ejecución de las etapas definidas anteriormente de manera secuencial.

Se suele usar en grandes proyectos, cuando el volumen de personas involucradas en los equipos es elevado y el contacto con el cliente está limitado tanto en tiempo como en personas. Existe el riesgo de no mostrar nada al cliente hasta la fase de pruebas.

Ventajas

- Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.
- Es un modelo fácil de implementar y entender.
- Está orientado a documentos.
- Es un modelo conocido y utilizado con frecuencia.

- Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar.

Desventajas

- En la vida real, un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.
- El proceso de creación del software tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el software no esté completo no se opera. Esto es la base para que funcione bien.
- Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.
- Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.

5.2 DESARROLLO EN ESPIRAL

Sigue un modelo de desarrollo con forma de espiral que ejecuta una serie de etapas por ciclo de manera recurrente, como si fuera un buble. En cada iteración se tiene en cuenta cuatro actividades:

1. Determinar objetivos
2. Análisis del riesgo
3. Desarrollar y probar
4. Planificación

Determinar o fijar objetivos

- Fijar también los productos definidos a obtener: requisitos, especificación, manual de usuario.
- Fijar las restricciones.
- Identificación de riesgos del proyecto y estrategias alternativas para evitarlos.
- Hay una cosa que solo se hace una vez: planificación inicial.

Desarrollar, verificar y validar (probar)

- Tareas de la actividad propia y de prueba.

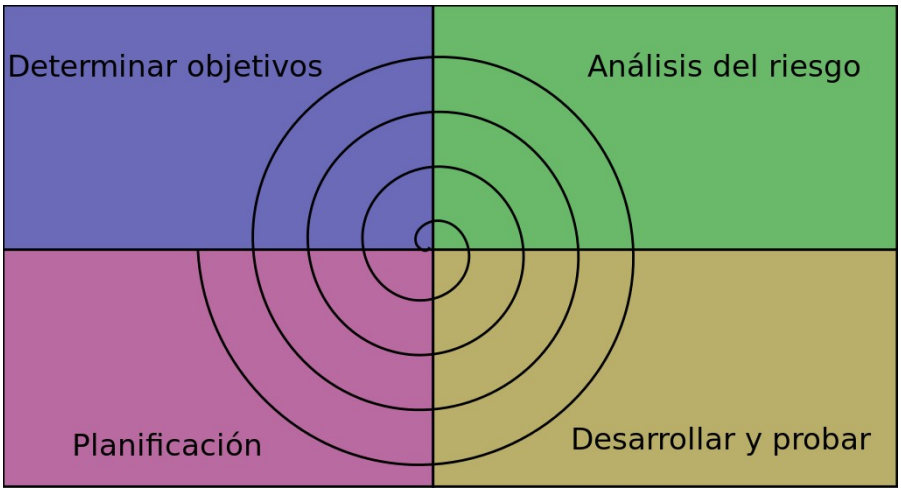
- Análisis de alternativas e identificación resolución de riesgos.
- Dependiendo del resultado de la evaluación de los riesgos, se elige un modelo para el desarrollo, el que puede ser cualquiera de los otros existentes, como formal, evolutivo, cascada, etc. Así si por ejemplo si los riesgos en la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos. Si lo riesgos de protección son la principal consideración, un desarrollo basado en transformaciones formales podría ser el más apropiado.

Análisis del riesgo

- Se lleva a cabo el estudio de las causas de las posibles amenazas y probables eventos no deseados y los daños y consecuencias que éstas puedan producir. Se evalúan alternativas. Se debe tener un prototipo antes de comenzar a desarrollar y probar.

  En resumen, es para tener en cuenta los riesgos de cada uno de los ámbitos.

  Si se pensara gráficamente en el modelo se obtendría el siguiente imagen:



Este modelo busca minimizar el riesgo de no mostrar nada al cliente hasta una fase muy avanzada del desarrollo mediante el desarrollo incremental y el contacto continuo con el cliente.

Ventajas

- El análisis del riesgo se hace de forma explícita y clara.
- Reduce riesgos del proyecto.
- Incorpora objetivos de calidad.
- Integra el desarrollo con el mantenimiento, etc.

- Añade la posibilidad de tener en cuenta mejoras y nuevos requerimientos sin romper con la metodología, ya que este ciclo de vida no es rígido ni estático.

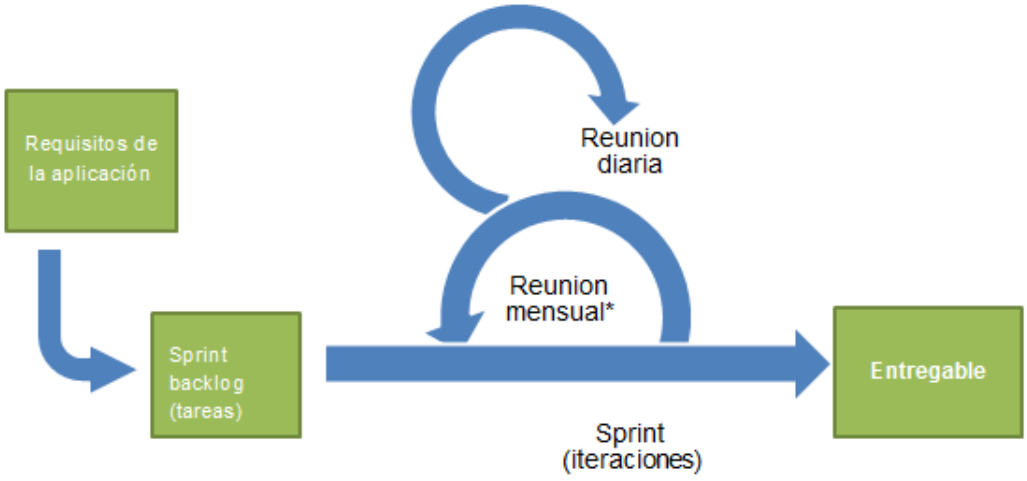
Desventajas

- Genera mucho tiempo en el desarrollo del sistema.
- Modelo costoso.
- Requiere experiencia en la identificación de riesgos.

5.3 DESARROLLO ÁGIL DE SOFTWARE

En los últimos años se ha avanzado en la implantación del desarrollo ágil de software, el cual lleva a un nivel más avanzado el enfoque de desarrollo incremental del desarrollo en espiral, organizando ciclos o iteraciones más cortos en tiempo y dedicación.

Los métodos ágiles enfatizan en las comunicaciones cara a cara en lugar de la documentación escrita, y se basan en el compromiso de los integrantes de los equipos, repartiendo la responsabilidad del proyecto por igual.



SCRUM es uno de los modelos de desarrollo ágil más usado actualmente y se basa en el desarrollo de pequeños incrementos en iteraciones de 2 semanas. Al inicio de cada iteración el equipo decide que va a desarrollar en función de las prioridades del cliente, y durante 2 semanas trabaja en su análisis, diseño y codificación. Al final de las dos semanas se prueba y valida con el usuario todos los requisitos finalizados para dar por finalizada la iteración y continuar con la siguiente.

Este modelo requiere un equipo con cierta experiencia y conocimiento del sistema, ya que los ciclos son muy cortos y una baja eficiencia o dispersión en los trabajos puede ocasionar malestar en el cliente.

Ventajas

- Flexibilidad y adaptabilidad a los cambios.
- Mayor productividad, basada en la motivación de los equipos.
- Reducción de riesgos.
- Alineamiento entre cliente y equipo de desarrollo.
- Cumplimiento de expectativas.

Desventajas

- Dificultad de aplicación en proyectos grandes.
- Supone que el equipo está formado y motivado.
- Problemas en proyectos cerrados en fecha y precio.
- Necesita una elevada interacción con el cliente.