ASIGNATURA

Fundamentos de Programación I

GRADO EN INGENIERÍA INFORMÁTICA

TEMA 1

Introducción a la programación de sistemas de computación

Autor: Francisco José Palacios Burgos

Unidad didáctica.

Tema 1: Introducción a la programación de sistemas de computación

INDICE

Introducción a la programación de sistemas de computación

I. PRESENTACION

La Informática, tal y como se entiende hoy esa rama moderna de la ingeniería, nace en la primera mitad del siglo XX. Entre los años 30 y los años 60 se producen una serie de avances en la producción de componentes electrónicos (transistores simples, transistores MOS) que permitió la construcción de máquinas (computadoras) que eran capaces de procesar información y realizar cálculos a una velocidad mayor que la de cualquier ser humano.

Las computadoras por sí solas no eran tremendamente útiles. Era necesario que los seres humanos registrasen en ellas las instrucciones necesarias para esos procesamientos automatizados de la información.

En este tema se van a ver las principales características de la disciplina de programación de computadoras, así como un recorrido por las herramientas y técnicas más relevantes de este campo.

Este tema servirá como hilo conductor y resumen de todos los demás contenidos de la asignatura.

II. OBJETIVOS.

Mediante el estudio de este tema se adquirirá la capacidad de:

- Comprender en qué consiste la ciencia de la programación de computadoras
- Asimilar y expresar los principales conceptos de esta ciencia (programa informático, instrucciones, lenguaje de programación, ciclo de vida de un programa)

III. ESQUEMA

IV. CONTENIDO

1. Análisis y resolución de problemas mediante la programación de computadoras.

En muchos de los ámbitos humanos de actuación, se requiere el procesamiento de importantes cantidades de información, lo cual constituye una parte fundamental de las organizaciones. Un ejemplo de esto podría ser una entidad financiera, que realiza al día un número muy grande de transacciones monetarias. Todas esas operaciones suponen el tener que manejar y procesar enormes cantidades de información relativas a sus clientes. Otro ejemplo que se podría poner sería el de los centros de salud, que mantienen historiales sobre los pacientes. Estos historiales deben estar accesibles de forma rápida y segura para que el personal sanitario pueda consultarlos en cualquier momento que se necesiten. Una necesidad que es también muy frecuente, es la de la velocidad de procesamiento. Por ejemplo, en el ámbito científico, en sitios como los centros de cálculo de alto rendimiento se necesita procesar información de manera muy rápida y masiva.

A medida que las organizaciones crecen en tamaño y aumentan las interdependencias con otras organizaciones el volumen de información que se genera, se transmite y se almacena es cada vez mayor, siendo este crecimiento en muchos casos no lineal. También se incrementa la necesidad de procesar esa información a una velocidad mayor. Inicialmente todo ese procesamiento de información podía hacerse mediante el trabajo manual de personas, pero llega un momento en que se hace necesario automatizar los procesos si se quiere mantener un rendimiento aceptable y no perder eficacia. Ese es precisamente el campo de actuación de la informática. Hoy en día, ésta constituye una poderosa herramienta de apoyo para cualquier actividad del ser humano que involucre información.

Introducción a la programación de sistemas de computación

Un **problema en informática** consiste en el procesamiento, mediante un sistema computacional, de un cierto volumen de información. Este procesamiento es automatizado, sin apenas intervención del ser humano

Para encontrar una solución a un problema en esta área del conocimiento, se siguen de forma general tres pasos:

- En primer lugar, hay que estudiar el problema en cuestión. Esta etapa recibe el nombre de análisis y comprende tareas como la recogida y especificación de requisitos, la identificación de conceptos clave y de tareas, etc. También forma parte de esta etapa el acotar el alcance de la solución y estudiar la viabilidad de la misma.

- Después de haberlo comprendido se procede a diseñar una solución al mismo. Esta es la etapa del diseño. La especificación de forma concisa de los pasos necesarios para resolver el problema es lo que se conoce como algoritmo.

- Tras el diseño de la solución, se implementa la misma, traduciendo los pasos obtenidos en el algoritmo a un lenguaje de programación concreto, obteniéndose finalmente un programa que puede ser ejecutado en una computadora.

Todas estas etapas pueden ponerse de forma gráfica en una secuencia como la siguiente:

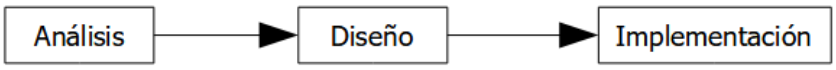

Figura 1. Etapas de la resolución de problemas mediante computadora

El resultado final de esta secuencia es un producto, el programa. Podemos definir este concepto como sigue:

Programa: Conjunto finito de instrucciones de computadora más o menos elementales que resuelven un determinado problema mediante el procesamiento de información y/o la interacción con el usuario.

Es importante que el conjunto de instrucciones sea finito. Conjunto finito significa tiempos de ejecución finitos. Las instrucciones serán más o menos elementales dependiendo del lenguaje de programación escogido. En muchas ocasiones se necesitará una cierta interacción con el usuario, que suele ser de dos clases: introducción de datos y toma de decisiones.

A partir del programa, y como veremos en la sección 1.4 se pueden obtener ficheros que se pueden ejecutar en la computadora para realizar la tarea encomendada. En ocasiones se habla de también de **código fuente** para referirse a la especificación del algoritmo en un lenguaje de programación concreto.

Un aspecto que se debe tener en cuenta es que las aplicaciones informáticas pueden ser de muy diversos tipos. Asimismo, su complejidad varía también en gran medida. No es lo mismo desarrollar una base de datos, que un videojuego o un programa para convertir cantidades de euros a dólares. Para aquellos programas de mayor tamaño y complejidad es necesario contar con una serie de etapas previas de análisis y estudio del problema ya que comenzar directamente por la implementación suele ser fuente de errores y de pérdidas de tiempo y dinero.

2. El ciclo de vida del software

Existen muchos tipos de aplicaciones software. Desde la pequeña aplicación que se construye para resolver un problema muy concreto hasta grandes aplicaciones como puedan ser una base de datos o un sistema operativo en el que el número de líneas de código puede llegar ser del orden de varios cientos de miles o incluso del orden de millones. En general, este segundo tipo de aplicaciones, reciben el nombre de proyectos software, y requieren de grandes recursos humanos y técnicos además de una considerable cantidad de tiempo para completarse.

Proyecto software: Conjunto de recursos tales como códigos fuente, manuales, y documentación técnica necesarios para la construcción de uno o varios programas.

Para el desarrollo de estos proyectos software se requiere un grado muy alto de planificación previa ya que, de no hacerse, el riesgo de fracaso en el proyecto o de disparos de los costes se incrementa notablemente.

Todo proyecto software tiene asociado lo que se denomina un **ciclo de vida** que es el tiempo que transcurre desde que se concibe inicialmente el sistema hasta el momento en que dicho software se retira de la comercialización o de su uso. Este ciclo de vida se divide en varias etapas, como son las de recogida y análisis de requisitos, análisis, diseño, implementación, codificación, pruebas, verificación, documentación, explotación y mantenimiento.

La forma en la que ocurren estas fases es lo que se conoce como modelo de desarrollo. Existen varios modelos distintos, pudiéndose usar uno u otro en función de las necesidades de los desarrolladores. Dos ejemplos muy utilizados son el modelo lineal-secuencial o modelo en cascada en el que las etapas se suceden una tras otra y cada fase empieza donde termina la anterior. Las fases estándar que se suelen emplear son Análisis, diseño, codificación (o programación), pruebas y mantenimiento.

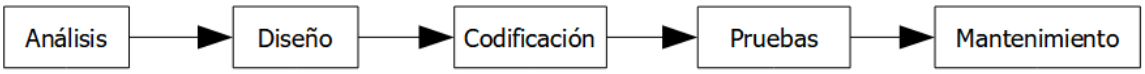

Figura 2. Etapas de un modelo de desarrollo lineal

Otro modelo que también se utiliza en gran medida es el modelo en espiral o iterativo. En él se trabaja con prototipos que son versiones del programa con funcionalidad parcial. Estos prototipos se van mejorando en etapas, y se validan a medida que se enseñan y/o entregan a los clientes.

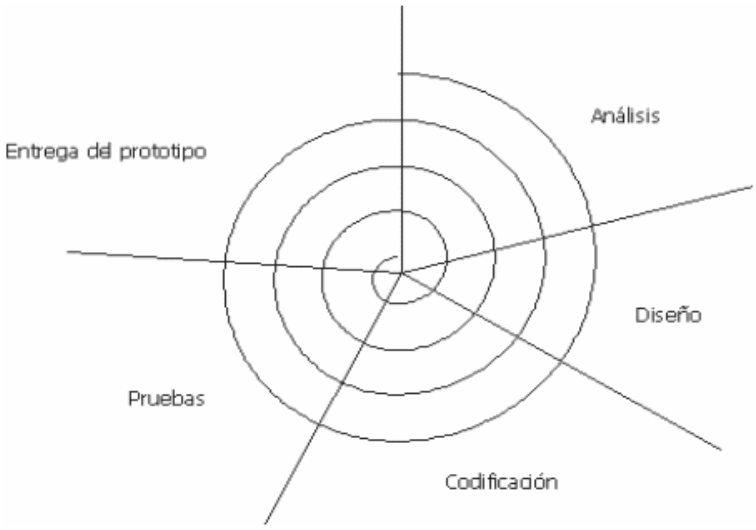

Figura 3. Modelo de desarrollo en espiral

El modelo en cuestión a emplear depende de varios factores. En general, en aquellos proyectos en los que los requisitos y las especificaciones están muy claros (tanto para los clientes como para los desarrolladores), se puede usar de forma bastante segura modelos de tipo lineal o en cascada. Por contra, en aquellos proyectos en los que pueden no estar claros los requisitos y especificaciones (de funcionamiento, de interfaz de usuario, etc.) suele ser conveniente utilizar algún modelo de desarrollo en espiral, ya que es más fácil tomar acciones de corrección de diseño.

Adicionalmente a los modelos de desarrollo, hoy en día este campo se caracteriza por el empleo de ciertas metodologías de desarrollo que se están mostrando muy exitosas sobre todo en el ecosistema de pequeñas empresas de tipo Start-up.

Entre estos modelos de desarrollo nos encontraríamos con:

Agile: Se caracteriza por dividir el desarrollo el proyecto en proyectos más pequeños. Suele emplearse en el desarrollo de nuevas versiones por actualización de productos existentes.

Scrum: En este caso, las tareas de un proyecto se suelen dividir en unos ciclos más cortos llamados sprints. Se suele proponer a los participantes en función de su experiencia y sus habilidades, otorgándoles más peso en aquellas áreas de las que son especialistas.

Extreme programming: Es un método evolutivo. Se centra en el desarrollo prioritario de las tareas críticas. Se suele emplear cuando no está clara la estabilidad del entorno de desarrollo o cuando los requisitos son muy inciertos.

Pair programming: Se da cuando el número de desarrolladores es dos. Cada uno se centra en las tareas que domina. El éxito depende de las características de ambos programadores.

Kanban: Metodología que se basa en el control de tareas por fases y por fechas de vencimiento. Si el proyecto no es muy complejo puede ser una buena elección.

Existen otros muchos modelos. En general, todas las etapas de Análisis, Diseño y los distintos modelos de desarrollo son objeto de estudio de la disciplina conocida como **Ingeniería del Software**, que resulta de vital importancia a medida que el software se hace más complejo.

3. Lenguajes de Programación

En la etapa de la implementación de los algoritmos obtenidos en la etapa de diseño, una de las cuestiones con las que se encuentran los programadores es elegir el lenguaje de programación concreto con el que se codificarán los mismos. Lenguajes de programación hay muchos, y cada uno se especializa en resolver un tipo de problemas concreto o en seguir un **paradigma de programación** determinado.

La evolución de los lenguajes de programación ha ido un poco paralela a la evolución del hardware. Si al principio eran lenguajes muy cercanos al propio lenguaje máquina que es el que

entiende la computadora (distintos tipos de ensamblador), con el tiempo fueron evolucionando, adoptando en cada generación filosofías distintas.

Haciendo un esquema un poco simplificado, tendríamos los siguientes tipos de lenguajes:

1ª generación: Eran los lenguajes máquina y de muy bajo nivel. Se caracterizaban por depender mucho de la arquitectura hardware. Las operaciones tenían más que ver con las propias de la máquina (operaciones con registros). Debido a esto, había poca portabilidad. Además, los programas eran difíciles de entender o de verificar. Esto conlleva que no sean muy fiables.

2ª generación: Eran lenguajes de tipo imperativo, es decir, que se basan en la operación de asignar datos a una posición de memoria. Principalmente compilados. Algunos ejemplos: COBOL o FORTRAN. Aparecen operaciones más complejas.

3ª generación: También eran lenguajes de tipo imperativo, pero se hace más énfasis en la programación estructurada (contienen más elementos en la sintaxis para poder seguir esa filosofía de programación). Algunos ejemplos: PASCAL o C. En esta generación ya empiezan a aparecer una serie de técnicas para diseñar los programas. Se empieza a hablar de Ingeniería del Software

4ª generación: Programas orientados a objetos. Son lenguajes que facilitan el tratamiento de grandes cantidades de datos y están muy cercanos al lenguaje humano, por lo que el desarrollo se simplifica. La Ingeniería del software representa una parte muy importante del desarrollo de las aplicaciones. Algunos ejemplos son C++, Java, C#, etc.

5ª generación: Lenguajes orientados hacia la Inteligencia Artificial. Suelen ser declarativos (basados en la composición y relación de funciones o en las relaciones lógicas entre objetos o datos). Suelen ser interpretados. Ejemplos: LISP, PROLOG.

Como vemos, la tendencia ha sido por una parte la de acercar los lenguajes de programación a la forma de expresarse del ser humano, y por otra a la de potenciar la ingeniería del software a la hora de desarrollar aplicaciones. Actualmente conviven en la empresa los lenguajes de 3ª y 4ª generación principalmente, pudiéndose encontrar también casos de la 5ª, 2ª y 1ª generación, pero en mucha menor proporción.

4. Herramientas de Programación

A la hora de implementar los programas será necesario hacer uso de una serie de herramientas. Algunas son indispensables para esta tarea, mientras que otras es bastante recomendable usarlas. A continuación, vamos a enumerar las diversas herramientas que podemos usar. En la próxima unidad didáctica, extenderemos en detalle este listado.

Herramientas CASE: Dentro de la ingeniería del software asociada a un proyecto, antes de programar se pueden utilizar una serie de herramientas que nos permitan generar la documentación (textual o diagramas) necesaria para las etapas de recogida de requisitos, análisis y diseño. Normalmente algunas de estas herramientas estarán ligadas a algún tipo de metodología concreta.

Editor: Una vez que estemos ya en la etapa de codificación, lo primero que necesitaremos será un editor de texto para escribir los ficheros que contienen el código fuente de nuestra aplicación. Aunque editores los hay de muchos tipos y básicamente cualquiera que permita editar y cargar/guardar los resultados bastaría, hay una serie de características adicionales que pueden ser muy interesantes desde el punto de vista de la programación:

- Resaltado de sintaxis (syntax highlighting)
- Ayuda contextual y autocompletación de código
- Operaciones de refactorización:
- Otras características que pueden facilitar la tarea de programación pueden ser la de poder abrir varios ficheros simultáneamente, capacidad de revisar la sintaxis antes de compilar, capacidades de formateo del código, etc.

Compilador/Intérprete: La mayoría de los lenguajes de programación se pueden clasificar en lenguajes compilados o lenguajes interpretados.

En los lenguajes compilados, a partir del código fuente se genera un fichero en código objeto, que es una traducción del programa en el lenguaje que entiende la máquina. El encargado de generar este código objeto es un programa llamado compilador.

Otros lenguajes en cambio son interpretados. En estos lenguajes, no se produce un fichero de código objeto, sino que un cierto programa llamado intérprete se encarga de ir leyendo las

instrucciones de nuestro código objeto e irlas traduciendo en tiempo de ejecución a instrucciones de la máquina.

También tenemos lenguajes que se ejecutan de forma interpretada dentro de otro programa. Tal sería el caso por ejemplo de Javascript, que se ejecuta dentro de otro programa (el navegador web).

Depurador: Cuando desarrollamos programas, lo normal es que exista una posibilidad de introducir algún fallo en la etapa de codificación. Los fallos se pueden catalogar en varias clases.

Por una parte tendríamos los fallos o errores de sintaxis. Cada lenguaje de programación introduce una gramática que debe ser seguida. Este tipo de fallos es detectado siempre por el compilador, que conoce cual son las reglas de dicha gramática.

Otro tipo de fallos que podemos encontrar son los que se producen en tiempo de ejecución. Algunos lenguajes por la manera en la que están diseñados nos permiten disponer de compiladores que detectan estos fallos. En otros lenguajes en cambio, necesitaremos de alguna herramienta adicional que ejecute el programa una vez compilado. Estas herramientas suelen llamarse depuradores y requieren que en tiempo de compilación se haya producido código extra que ayude a la depuración.

En la depuración podríamos tomar acciones como establecer puntos de ruptura (pausas en la ejecución del programa), inspeccionar los valores que toman las variables en ese momento, etc.

Gestión de Proyectos: Una aplicación software es mucho más que los meros ficheros de código. Entre otras cosas, comprende los ficheros de código, la documentación de las etapas de análisis y diseño, los manuales de instalación y uso, etc.

Es muy beneficioso el utilizar alguna herramienta que tenga en cuenta todo esto y nos permita gestionar un proyecto software de forma general. Los beneficios de gestionar todo lo relativo a un proyecto son grandes, disminuyendo las posibilidades de pérdida de la información y de mezcla de proyectos.

Control de versiones: Cuando uno desarrolla un programa, lo normal es que el desarrollo no se haga de una sola vez, sino que con el tiempo se obtienen ficheros de fuentes que implementan más y más cosas. Para proyectos pequeños desarrollados por un solo programador esto no es un problema, pero cuando los programas son grandes o el número de programadores asignados es mayor que uno, podemos encontrarnos con situaciones en las cuales no sabemos si un fichero fuente era o no la última versión. Esto a lo que conduce normalmente es a la pérdida de trabajo.

Desde hace algún tiempo existen una serie de soluciones software que gestionan el control de versiones. Estos programas se basan en la idea de un repositorio software (normalmente en un servidor) que contiene las distintas versiones del programa. Los programadores entonces bajan las últimas fuentes, las modifican y luego las suben. El sistema de control de versiones se encarga de que no se produzcan dos actualizaciones simultáneas a la vez. Normalmente todo esto funciona de forma incremental: Se almacenan los cambios y no todo el fichero.

Algunos de los sistemas de este tipo más populares en la actualidad son Subversión (solución de servidor centralizada), Git (solución que favorece la descentralización), Mercurial o Bazar.

Herramientas de generación automática de documentación: Una de las partes de la documentación de un programa más importantes es el manual del programador. Este manual es necesario si queremos retomar el trabajo sobre el programa en el futuro o si queremos que otros programadores distintos puedan mantener un programa dado.

Existen herramientas que nos permiten generar dicha documentación de forma automática, a partir del código fuente.

IDE: Un IDE (Entorno de desarrollo integrado) es una aplicación software que implementa o permite el uso de varias de las herramientas anteriores desde un mismo entorno de trabajo. Actualmente hay muchos IDE's para todas las plataformas que son muy potentes. Aunque un IDE normalmente exige un periodo de entrenamiento, una vez conseguido el aumento de productividad suele ser considerable.

5. Documentación del Software

La documentación es una parte muy importante del desarrollo de cualquier aplicación software. Existe una doble vertiente en lo referente a la documentación:

Introducción a la programación de sistemas de computación

1. Documentación necesaria para la explotación del software

Esta es la documentación que debe generarse para poder usar el software una vez que haya acabado la etapa de implementación. Comprende los manuales del administrador y el manual de usuario.

El administrador es la persona que se encarga de la instalación y puesta en marcha de un programa. Por tanto, entrarían aquí los manuales de instalación y de inicialización de la aplicación.

El manual de usuario contendría todos aspectos necesarios para la ejecución diaria del programa y normalmente existe en versión impresa y on-line. En la medida de lo posible hay que tender a usar elementos visuales, como puedan ser las capturas de pantalla en este tipo de ayuda y evitar un poco lo que es el lenguaje muy técnico, ya que el usuario normalmente no tendrá conocimientos muy técnicos.

2. Documentación necesaria para el mantenimiento del programa.

Este tipo de documentación va dirigida a los futuros programadores que tengan que modificar el programa bien sea para corregir fallos que se encuentren durante la etapa de explotación o bien sea porque se va a ampliar la funcionalidad del programa.

Esta documentación es de tipo más técnico y debe incluir todos aquellos documentos que se generen durante el ciclo de vida del software. Esto incluye desde especificaciones de requisitos y diagramas generados en la etapa de análisis hasta documentación de tipo interno, como pueda ser los comentarios dentro del código del programa.

Un programa sin este tipo de documentación es muy difícil de modificar, especialmente si los programadores que se van a encargar de la tarea no son los que originalmente hicieron la aplicación.

Existen herramientas que generan parte de toda esta documentación de forma automática. Muchas veces, estas herramientas se encuentran integradas en los IDE's

V. RESUMEN

En este tema hemos visto en qué consiste la programación de software. Es una rama de la ingenería que permite que utilicemos las computadoras para procesar información, dándolas las instrucciones precisas para ello.

Como toda rama de la ingeniería, ésta tiene sus metodologías e instrumentos. La metodología aplicada, tendrá su efecto en el ciclo de vida del software que se está realizando

Por otra parte, los instrumentos empleados (lenguaje de programación y herramientas de programación) condicionarán toda la etapa de implementación del ciclo de vida.

El resultado final será un programa software, que se ejecutará bajo un sistema operativo para producir un resultado.

VI. GLOSARIO

Volvemos a repasar algunos de los principales conceptos de esta unidad:

Instrucción	Acción escrita que el computador deberá realizar dentro de un programa. Será más o menos elemental dependiendo del lenguaje de programación empleado
Programa	**Conjunto finito de instrucciones de computadora más o menos elementales que resuelven un determinado problema mediante el procesamiento de información y/o la interacción con el usuario**
Proyecto software	**Conjunto de recursos tales como códigos fuente, manuales, y documentación técnica necesarios para la construcción de uno o varios programas**
Compilador	Programa informático encargado de generar un programa ejecutable a partir de un código fuente formado por instrucciones de un cierto

	lenguaje de programación
Intérprete	Programa informático encargado de leer y ejecutar un código fuente formado por instrucciones de un cierto lenguaje de programación
Depurador	Programa informático encargado de leer y ejecutar un ejecutable y mostrar los errores que se puedan producir
IDE	Aplicación software dedicada al desarrollo de programas. Permite editar, ejecutar/interpretar y depurar entre otras funcionalidades
Ciclo de vida	Es el tiempo que transcurre entre que se concibe un programa hasta que se retira de su uso. Está compartimentado en etapas
Lenguaje de Programación	Gramática que se establece para escribir programas, formada por palabras reservadas, una sintaxis y un conjunto de instrucciones predefinidas