



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 3

1. Introducción-definiciones.
 - 1.1. Algoritmo.
 - 1.2. Pseudocódigo.
 - 1.3. Ordinograma o diagrama de flujo.
 - 1.4. Instrucción.
 - 1.5. Programa.
 - 1.6. Lenguaje de programación.
 - 1.7. Subrutina o subprograma.
 - 1.8. Función.
 - 1.9. Procedimiento.
 - 1.10. Compilador.
 - 1.11. Intérprete.
2. Características técnicas en función del nivel.
 - 2.1. Lenguajes Máquina.
 - 2.2. Lenguajes Simbólicos.
 - 2.2.1. Lenguaje Ensamblador.
 - 2.2.2. Lenguajes de Alto Nivel (LANs).
3. Especialización funcional.
4. Lenguajes orientados a objetos (LOOS).
 - 4.1. Clase.
 - 4.2. Objetos.
 - 4.3. Tipos Abstractos de Datos (TDA) o Encapsulación.

- 4.4. Polimorfismo/Sobrecarga.
- 4.5. Herencia.
- 4.6. Enlace dinámico.
- 4.7. Tipos de LOOs.
- 5. Lenguajes multiplataforma: JAVA (C# , PHYTON).
 - 5.1. Características de Java.
 - 5.2. Diferencias entre Java y C++.
 - 5.3. Entornos de desarrollo.
 - 5.3.1. Del lado del cliente.
 - 5.3.2. Del lado del servidor.
- 6. Lenguajes de cuarta generación: SQL (4GL).
 - 6.1. Características.
 - 6.2. DDL.
 - 6.3. DML.
 - 6.4. DCL.
- 7. Productividad: las métricas orientadas al tamaño del código.
 - 7.1. Características.
 - 7.2. Relaciones de tamaño y complejidad para desarrollar procesos.
- 8. Estandarización: valoración lenguajes en función de sus características.



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 3

Lenguajes actuales de programación. Características técnicas. Especialidad funcional. Productividad. Estandarización.

1. INTRODUCCIÓN-DEFINICIONES.

1.1. ALGORITMO.

Es un conjunto de acciones o secuencia de operaciones que ejecutadas en un determinado orden resuelven el problema. Existen n algoritmos, hay que coger el más efectivo.

Representa una secuencia ordenada de pasos -sin ambigüedades-, repetible, que es solución de un determinado problema.

Las características fundamentales que debe cumplir todo algoritmo son:

1. Debe ser preciso e indicar el orden de realización de cada paso.
2. Debe estar definido (si se repiten n veces los pasos se debe obtener siempre el mismo resultado).
3. Debe ser finito (debe tener un número finito de pasos).
4. Es independiente del lenguaje de programación que se utilice.
5. La definición de un algoritmo debe describir tres partes: Entrada, Proceso, Salida.
6. La programación es adaptar el algoritmo al ordenador.
7. El algoritmo es independiente de donde se implemente (hardware o software).

La representación de los algoritmos:



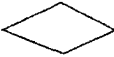

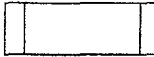

Una vez que tenemos la solución hay que implementarla con alguna representación. Las representaciones más usadas son los ordinogramas, los diagramas N-S (Nassi-Schneiderman) y el pseudo-código.

1.2. PSEUDOCÓDIGO.

Es un lenguaje de especificación de algoritmos próximo al lenguaje natural humano y que permite una rápida traducción a los lenguajes de programación de alto nivel. El algoritmo utiliza para representar las sucesivas acciones del algoritmo palabras similares a sus homónimas de los lenguajes de programación: inicio, fin, parar, si entonces ... mientras ...

1.3. ORDINOGRAMA O DIAGRAMA DE FLUJO.

Se trata de otra forma de especificación de algoritmos basada en la utilización de una serie de símbolos para indicar las secuencias de acciones contenidas en el algoritmo. Su uso ha disminuido notablemente, en particular desde la aparición de los lenguajes estructurados y orientados a objetos. Los símbolos más frecuentes serían los siguientes:

SÍMBOLO	SIGNIFICADO
	Terminal, inicio o fin de algoritmo
	Proceso
	Decisión o Alternativa
	Entrada / Salida cualquiera
	Llamada a subrutina
	Salida a impresora

1.4. INSTRUCCIÓN.

Cada una de las tareas elementales a realizar por un elemento de computación.

1.5. PROGRAMA.

Un programa es un conjunto de instrucciones que al ser ejecutadas resuelven un problema.

Un programa tiene tres partes:

1. Entrada de datos: normalmente se va a ejecutar a través de instrucciones de lectura, y en lo que se le pide al usuario la información que el programa va a necesitar para ejecutarse, o bien se leen los datos desde un sistema de almacenamiento secundario.
2. Acciones de un algoritmo: parte en la que se resuelve el problema usando los datos de entrada.

En la parte de las acciones a ejecutar se distinguirán dos partes:

- Declaración de variables.
 - Instrucciones del programa.
3. Salida: mostrar en un dispositivo de salida los resultados de las acciones anteriormente realizadas o grabarlos en un sistema de almacenamiento secundario. Son acciones de escritura.

1.6. LENGUAJE DE PROGRAMACIÓN.

Conjunto de normas «lingüísticas» que permiten escribir un programa y que éste sea entendido por el ordenador y pueda ser trasladado a ordenadores «similares» para su funcionamiento en otros sistemas.

1.7. SUBROUTINA O SUBPROGRAMA.

Fragmento de programa que resuelve un subproblema con entidad propia.

Los subprogramas dividen las tareas grandes de computación en varias más pequeñas y especializadas:

- Permiten reusabilidad.
- Distribuyen la programación.
 - Distintos niveles de abstracción.
 - Desarrollo por un equipo de personas.

1.8. FUNCIÓN.

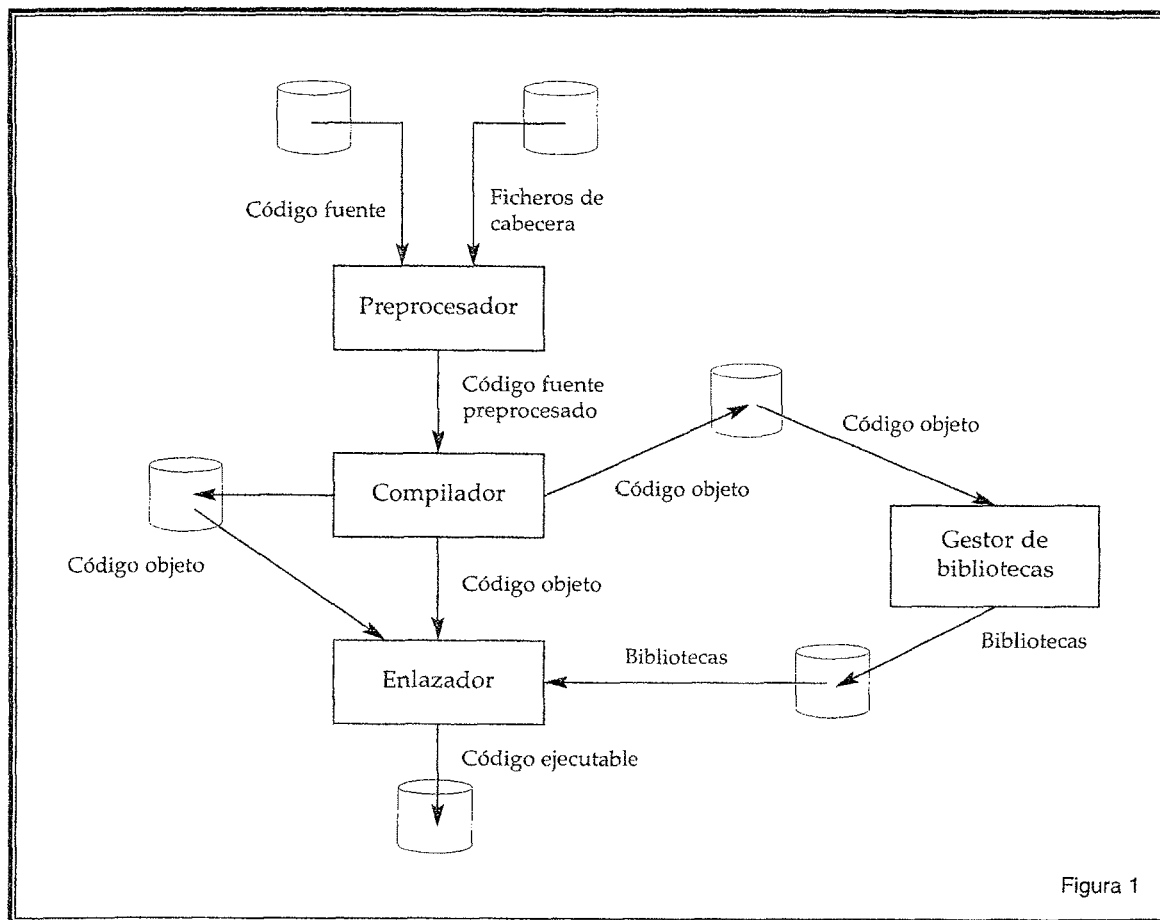
Subprograma que se caracteriza por disponer de «n» entradas, pero únicamente de una salida -siempre una-.

1.9. PROCEDIMIENTO.

Subprograma que se caracteriza por disponer de «n» entradas y «n» salidas, siendo $n = \{0, 2, \dots n\}$.

1.10. COMPILADOR.

Es un traductor que convierte un texto escrito en un lenguaje fuente de alto nivel en un programa objeto en código máquina. El esquema de un compilador (Lenguaje C) sería el siguiente:



1.11. INTÉRPRETE.

Es un traductor que realiza la operación de compilación paso a paso. Para cada sentencia que compone el texto de entrada, se realiza una traducción, ejecuta dicha sentencia y vuelve a iniciar el proceso con la sentencia siguiente. No se genera código/s intermedio/s.

La principal ventaja del proceso de compilación frente al de interpretación es que los programas se ejecutan mucho más rápidamente una vez compilados; por el contrario, es más cómodo desarrollar un programa mediante un intérprete que mediante un compilador puesto que en el intérprete las fases de edición y ejecución están más integradas. La depuración de los programas suele ser más fácil en los

intérpretes que en los compiladores puesto que el código fuente está presente durante la ejecución. Estas ventajas pueden incorporarse al compilador mediante la utilización de entornos de desarrollo y depuradores simbólicos en tiempo de ejecución.

2. CARACTERÍSTICAS TÉCNICAS EN FUNCIÓN DEL NIVEL.

2.1. LENGUAJES MÁQUINA.

Son lenguajes que permiten la representación de las instrucciones que componen el programa mediante combinaciones de ceros y unos (binario puro). Se trata de instrucciones directamente interpretables por la Unidad de Control. Todo lenguaje máquina ha de aportar instrucciones de, al menos, los siguientes tipos:

1. Ejecutivas: sumas, desplazadores, XOR...
2. De control:
 - a) Salto incondicional («goto»).
 - b) Salto condicional.
3. De entrada/salida.

2.2. LENGUAJES SIMBÓLICOS.

2.2.1. Lenguaje Ensamblador.

Se trata de lenguajes que asignan a cada instrucción del lenguaje máquina (ceros y unos) un nombre nemotécnico con el fin de facilitar su uso y significado. Características:

- Dependen de procesador (igual que los lenguajes máquina). Cada procesador deberá tener su propio ensamblador. Incluso dentro de un mismo fabricante de procesadores serán necesarios lenguajes distintos cuando se cambia de modelo.
- Es necesario traducirlos a lenguaje máquina mediante un lenguaje denominado «Assembler».

2.2.2. Lenguajes de Alto Nivel (LANs).

Se trata de lenguajes próximos al lenguaje natural que proporcionan:

- Instrucciones para representar la lógica del programa.
- Estructuras de datos (tipos) independientes de la arquitectura hardware subyacente.

Ventajas de los LANs:

1. Transportabilidad: conocido el código fuente y llevando a cabo diferentes procesos de compilación/interpretación.
2. Independencia de la arquitectura hardware.
3. Facilidad de programación y mantenimiento.

Inconvenientes de los LANs:

1. No aprovechamiento de la arquitectura interna.
2. No optimización de la RAM.
3. Aumento del tiempo de compilación y ejecución.

3. ESPECIALIZACIÓN FUNCIONAL.

Los lenguajes de programación normalmente se conciben para ser utilizados en ciertos ámbitos o entornos para los cuales se les trata de dotar de características especiales. Teniendo en cuenta este extremo podemos clasificar los lenguajes en los siguientes tipos:

- Propósito General: VBasic (Basic), Delphi (Pascal), C++ (C), Java, C#...
- Propósito Específico:
 - Gestión de Empresa: COBOL, RPG, SQL, PL I ...
 - Científicos: Fortran, Mathlab...
 - Inteligencia Artificial: Prolog, Lisp...
 - Aplicaciones Web: PHP, Perl, Phython, JavaScript...

4. LENGUAJES ORIENTADOS A OBJETOS (LOOs).

Actualmente la orientación a objetos es un conjunto completo de tecnologías que cubre prácticamente todas las facetas relacionadas con la ingeniería del software. Sin embargo, inicialmente surgió únicamente como alternativa a los métodos clásicos de programación, por lo que lo primero que aparecieron en función de este nuevo paradigma fueron los Lenguajes Orientados a Objetos (LOOs). Todo LOOs se caracteriza por incorporar dentro de su sintaxis un repertorio de expresiones que permitan representar la características fundamentales de la orientación a objetos. Estas características son las siguientes:

4.1. CLASE.

Una clase es una abstracción de objetos, es un conjunto de objetos con propiedades (atributos y/o métodos) comunes. Las clases sólo existen en tiempo de definición y compilación, y se almacenan en el dispositivo correspondiente. No tienen una vida fuera de ese contexto. Es un concepto estático.

Las clases son las «plantillas» de las que luego se pueden crear múltiples objetos del mismo tipo. Representan la abstracción de las características comunes de un tipo determinado de objetos.

Componentes de las clases:

- Atributos: elementos que mantienen el estado del objeto. Pueden ser de instancia o de clase.
- Métodos: mensaje para realizar alguna acción en un objeto. Es similar a las funciones en los lenguajes de programación tradicionales.

4.2. OBJETOS.

Representan la instanciación o ejemplificación de una clase. Por ejemplo, una clase será LIBRO, sin embargo «El Quijote», «Manual de C++», etc., serán objetos definidos con el conjunto de propiedades y métodos que se hayan establecido para la clase a la que pertenecen. Los objetos, a diferencia de las clases, sólo existen en tiempo de ejecución. Se generan cuando se ejecuta la aplicación, y lo hacen en base a la definición que se ha hecho de la clase a la que pertenecen. Son entes dinámicos, que ocupan memoria, y que deben eliminarse cuando ya no se les referencia.

En el paradigma de orientación a objetos, una aplicación es un conjunto de objetos que interactúan, que se comunican entre sí mediante mensajes. Los mensajes son, por tanto, el medio a través del cual los objetos interaccionan. También se les llama peticiones. La estructura de un mensaje es siempre la misma: nombre del objeto receptor, nombre del método a ejecutar, y una lista de datos eventualmente vacía, a los cuales se les denomina argumentos o parámetros del mensaje.

4.3. TIPOS ABSTRACTOS DE DATOS (TDA) O ENCAPSULACIÓN.

El objeto puede ser visto como una caja negra. La estructura interna permanece oculta, tanto para el usuario como para otros objetos diferentes, aunque formen parte de la misma jerarquía, reduciendo la propagación de efectos colaterales cuando ocurren cambios.

La información contenida en el objeto será accesible sólo a través de la ejecución de los métodos adecuados creándose una interfaz para la comunicación con el mundo exterior.

Cada objeto es una «cápsula» que contiene todos los datos y métodos ligados a él. Para garantizar los TDAs se dispone en los LOOs de operadores para la protección de objetos, de tal forma que los atributos y métodos de un objeto pueden ser, básicamente, de tres tipos:

- Privados: sólo pueden ser utilizados por el propio objeto al que pertenecen.
- Públicos: pueden ser accedidos por cualquier objeto.
- Restringidos: son privados pero en las clases derivadas (con herencia) son públicos.

Asociado al concepto de encapsulación está el de ocultación. Realmente, ambos términos expresan el mismo concepto. El encapsulado oculta los detalles de la implementación interna a los usuarios de un objeto. Ello permite la modificación de un objeto sin por eso afectar al resto de módulos del sistema, siempre que se mantenga constante la interfase del mensaje.

4.4. POLIMORFISMO/SOBRECARGA.

Es demostrar comportamientos distintos según la situación. Puede darse de tres formas diferentes:

- Métodos: en este caso se denomina «sobrecarga» y ocurre cuando en una clase existen dos métodos con idéntico nombre pero con distinta lista de parámetros. El compilador los considera como dos métodos distintos y aplicará cada uno de ellos en la situación apropiada.
- Clases: es al que se refiere normalmente el concepto de polimorfismo y coincidiría con el concepto anteriormente expuesto. En este caso el polimorfismo puede referirse tanto a métodos como a atributos.
- Enlace dinámico: métodos virtuales, tal y como se trata más adelante.

4.5. HERENCIA.

La herencia es la propiedad por la cual una clase asume como propios todos los atributos y métodos definidos por otra. La que hereda se denomina «subclase» o «clase derivada». La que transmite la herencia recibe el nombre de «super-clase» o «clase base». Tiene una importancia extraordinaria.

Cuando únicamente se puede heredar de una clase base, se habla de herencia simple. En caso contrario, se trataría de herencia múltiple. Esta última no existe en Java, aunque sí en C++.

4.6. ENLACE DINÁMICO.

Capacidad de retrasar hasta el instante de ejecución la decisión sobre la clase de objeto que lo recibe y el método concreto que debe aplicarse.

Hasta que no se produzca la llamada durante la ejecución de la aplicación no se tiene decidido cuál es el método que se va a ejecutar, lo que permite una asignación dinámica de los recursos del sistema.

4.7. TIPOS DE LOOs.

Los LOOs se han clasificado tradicionalmente de la siguiente forma:

- Puros: todo método ha de pertenecer a una clase. Por lo tanto, todo ha de estar referido a una estructura «class». Ejemplos: Java, Smalltalk, Eiffel, etc.
- Híbridos: permiten la existencia de métodos fuera de una clase. Ejemplos: C++, ObjectPascal, Delphi, etc.

5. LENGUAJES MULTIPLATAFORMA: JAVA (C#, PHYTON).

Un lenguaje multiplataforma es aquel que permite la codificación de programas que pueden ser ejecutados en cualquier arquitectura hardware o software y el código resultante en la ejecución puede provenir de múltiples orígenes. El primer ejemplo de lenguaje de este tipo fue Java, en la actualidad existen otros como C# o Python.

5.1. CARACTERÍSTICAS DE JAVA.

Java supone un significativo avance en el mundo de los entornos software, y esto viene avalado por tres elementos clave que diferencian a este lenguaje desde un punto de vista tecnológico:

- Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable.
- Proporciona un conjunto de clases potente y flexible. Además la biblioteca de clases de Java proporciona un conjunto único de protocolos de Internet.
- Pone al alcance de cualquiera la utilización de aplicaciones que se pueden incluir directamente en páginas web (aplicaciones denominadas applets).

Las principales características de Java son:

- Orientación a objetos.

En este aspecto Java fue diseñado partiendo de cero, no siendo derivado de otro lenguaje anterior y **no tiene compatibilidad** con ninguno de ellos.

En Java el concepto de objeto resulta sencillo y fácil de ampliar. Además se conservan elementos «no objetos», como números, caracteres y otros tipos de datos simples.

- Robusto. Interpretado.

Java verifica su código al mismo tiempo que lo escribe, y una vez más antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación, ya que Java es estricto en cuanto a tipos y declaraciones y así, lo que es rigidez y falta de flexibilidad, se convierte en eficacia.

- Arquitectura neutral.

Java está diseñado para que un programa escrito en este lenguaje sea ejecutado correctamente independientemente de la plataforma en la que se esté actuando (Macintosh, PC, UNIX,...). Para conseguir esto utiliza una compilación en una representación intermedia que recibe el nombre de **bytecode**, que puede interpretarse en cualquier sistema operativo con un intérprete de Java (**Virtual Machine**).

- Trabajo en red.

Java anima las páginas web y hace posible la incorporación de aplicaciones interactivas y especializadas. Aporta la posibilidad de distribuir contenidos ejecutables, de manera que los suministradores de información de la web pueden crear una página de hipertexto (página web) con una interacción continuada y compleja en tiempo real; el contenido ejecutable es transferido literalmente al ordenador del usuario.

- Applets.

Una **applet** (miniaplicación gráfica contenida en una página web) es un pequeño programa en Java transferido dinámicamente a través de Internet. Presentan un comportamiento inteligente,

pudiendo reaccionar a la entrada de un usuario y cambiar de forma dinámica. Sin embargo, la verdadera novedad es el gran potencial que Java proporciona en este aspecto, haciendo posible que los programadores ejerzan un control sobre los programas ejecutables de Java que no es posible encontrar en otros lenguajes.

- Seguridad.

Los niveles de seguridad que presenta son:

- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de los bytecodes que asegura que no se viole ninguna construcción del lenguaje.
- Verificación del nombre de clase y de restricciones de acceso durante la carga.

Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles.

5.2. DIFERENCIAS ENTRE JAVA Y C++.

Java fue desarrollado basándose en C++, pero eliminando rasgos del mismo poco empleados, optándose por una codificación comprensible. Básicamente, encontramos las siguientes diferencias con C++:

- Java no soporta los tipos struct, unión ni punteros.
- No soporta typedef ni #define.
- Se distingue por su forma de manejar ciertos operadores y no permite una sobrecarga de operadores.
- No soporta herencia múltiple.
- Java maneja argumentos en la línea de comandos de forma diversa a como lo hacen C o C++.
- Tiene una clase string que es parte del paquete java.lang y se diferencia de la matriz de caracteres terminada con un nulo que usan C y C++.
- Java cuenta con un sistema automático -recolector de basura- para asignar y liberar memoria, con lo que no es necesario utilizar las funciones previstas con este fin en C y C++.

5.3. ENTORNOS DE DESARROLLO.

5.3.1. Del lado del cliente.

- Applets: aplicaciones Java que se ejecutan bajo el contexto de un navegador. Se ejecuta en un contexto seguro, es decir, no puede acceder a los recursos (memoria, disco, etc.) de la máquina donde se está ejecutando.

- Aplicaciones ordinarias: programas desarrollados con la misma idea y funcionalidad que con los lenguajes convencionales. En este caso se mejora desde el punto de vista de la instalación, ya que es posible su descarga on-line y en bytecode a cada máquina donde se vaya a ejecutar. Se ejecutan en modo local y sin limitaciones.

5.3.2. Del lado del servidor.

- Servlets: aplicaciones Java que devuelven código HTML, normalmente se utilizan para la generación de páginas dinámicas o control de acceso.
- JSP: se trata de páginas de contenido dinámico formadas por código HTML y Java embebido. Normalmente el código Java es el encargado de acceder a alguna fuente de datos. En la primera invocación la página JSP se transforma en un servlet.
- EJBs: componentes de software implementados en Java, reutilizables y ejecutables en una arquitectura diseñada al efecto: J2EE (Java two Enterprise Edition).

6. LENGUAJES DE CUARTA GENERACIÓN: SQL (4GL).

Un lenguaje 4GL es aquel que se caracteriza por tener que indicar únicamente a qué datos se van a acceder y nunca el cómo. Esta última propiedad es lo que los diferencia de los 3GL.

6.1. CARACTERÍSTICAS.

El SQL es un lenguaje que permite expresar operaciones diversas, por ejemplo, aritméticas, combinatorias y lógicas, con datos almacenados en Bases de Datos Relacionales, que son aquellas que se caracterizan porque la información está contenida en estructuras, llamadas tablas, donde los datos están dispuestos en filas y columnas. SQL significa Structured Query Language (Lenguaje Estructurado de Consultas).

El concepto de Base de Datos Relacional arranca de un artículo publicado en 1970 por Codd, empleado de IBM, donde se sentaban los conceptos básicos de un modelo relacional de datos y de un sublenguaje para acceder a ellos basado en el cálculo de predicados. La idea se desarrolló en IBM, dando lugar a un primer prototipo llamado System R que utilizaba un lenguaje llamado SEQUEL (que posteriormente daría lugar al SQL).

El ANSI (American National Standards Institute) ha adoptado este lenguaje como estándar, publicando y desarrollando unas especificaciones para este lenguaje, que han sido posteriormente aceptadas por el ISO. No significa esto que los productos existentes sigan estrictamente esta norma, principalmente porque el estándar no cubre todas las necesidades planteadas. Del mismo modo, existen diferencias entre distintos productos comerciales.

Las peticiones de datos se expresan en SQL mediante sentencias que deben seguir las normas sintácticas y semánticas del lenguaje. Estas sentencias se pueden escribir directamente en la pantalla de un terminal interactivo, o pueden ser utilizadas embebidas en programas, incorporándose así su capacidad expresiva a la lógica y funciones de éstos. A esta última forma de utilizar el SQL se la llama SQL dinámico o embebido.

El SQL permite la realización de consultas y actualizaciones sobre datos almacenados en tablas relacionales. Éste es el principal uso que harán de él usuarios y programadores. Pero también hay otras tareas que se pueden realizar mediante sentencias SQL, aunque pertenecen más a las responsabi-

lidades de los administradores de las bases de datos (DBA). Entre estas funciones adicionales se encuentran la definición y destrucción de objetos, y la gestión de autorizaciones de acceso. Las sentencias de SQL se puede clasificar en:

6.2. DDL.

Data Definition Language. Permiten definir nuevos objetos y/o destruir otros existentes. Algunos ejemplos de sentencias son las de tipo CREATE y DROP.

6.3. DML.

Data Manipulation Language. Permiten realizar consultas y mantenimiento de los datos. Comienzan con las siguientes palabras del lenguaje: SELECT, INSERT, UPDATE y DELETE.

6.4. DCL.

Data Control Language. Permite establecer y denegar privilegios y roles sobre objetos existentes en la base de datos. Además posibilita la gestión de transacciones. Algunos ejemplos serían: GRANT, REVOKE, COMMIT, ROLLBACK...

7. PRODUCTIVIDAD: LAS MÉTRICAS ORIENTADAS AL TAMAÑO DEL CÓDIGO.

7.1. CARACTERÍSTICAS.

La métrica del software es un factor realmente importante en el análisis de un proyecto. Las métricas orientadas al tamaño proporcionan medidas directas del software y del proceso por el cual se desarrolla. Se basan en la medición del número de Líneas De Código -LDC- que contiene el desarrollo, entendiendo por línea de código una sentencia del lenguaje de programación (se excluyen comentarios y líneas en blanco de las fuentes).

Una forma de clasificarlos es atendiendo al número de líneas de código, como se muestra en la tabla:

CATEGORÍA DE UN PROYECTO EN FUNCIÓN DE SUS LÍNEAS DE CÓDIGO

CATEGORÍA	PROGRAMADORES	DURACIÓN	LÍNEAS DE CÓDIGO	EJEMPLO
Trivial	1	0 – 4 semanas	< 1k	Utilidad de ordenación
Pequeño	1	1 – 6 meses	1k – 3k	Biblioteca de funciones
Media	2 – 5	0,5 – 2 años	3k – 50k	Compilador de C
Grande	5 – 20	2 – 3 años	50k – 100k	SO pequeño
Muy grande	100 – 1000	4 – 5 años	100k – 1M	Grandes SO
Gigante	1000 – 5000	5 – 10 años	> 1M	Sistema de distribución

El método COCOMO.

Una metodología que se encarga de medir proyectos software es COCOMO. La metodología COCOMO (COConstructive COst MOdel) se debe a Barry Boehm, y está orientada a líneas de código.

Hay una jerarquía de modelos COCOMO: básico, intermedio y avanzado, la cual se aplica a tres tipos diferentes de software:

1. Orgánico: proyectos relativamente sencillos, menores de 50.000 líneas de código. Se tiene experiencia en proyectos similares y se encuentra en un entorno estable.
2. Semiacoplado: proyectos intermedios en complejidad y tamaño. La experiencia en este tipo de proyectos es variable, y las restricciones intermedias.
3. Empotrado: proyectos bastante complejos, en los que apenas se tiene experiencia y en un entorno de gran innovación técnica. Se trabaja con unos requisitos muy restrictivos y de gran volatilidad.

Dado que sólo se va a emplear una variable para la estimación (la línea de código), se empleará COCOMO básico, ya que es un modelo univariable estático, con lo que se obtiene una valoración objetiva del esfuerzo realizado.

La ecuación del esfuerzo de COCOMO básico tiene la siguiente forma:

$$E = \text{Esfuerzo} = a \text{KLDC}^b \text{ (persona} \times \text{mes)}$$

donde KLDC es el número de líneas de código, distribuidas en millares, para el proyecto.

La ecuación del tiempo de desarrollo es:

$$T = \text{Tiempo de duración del desarrollo} = c \text{Esfuerzo}^d \text{ (meses)}$$

Por su parte los coeficientes a, b, c y d se obtienen empíricamente del estudio de una serie de proyectos, y sus valores son:

COEFICIENTES COCOMO

PROYECTO DE SOFTWARE	a	b	c	d
Orgánico	2,4	1,05	2,5	0,38
Semiacoplado	3,0	1,12	2,5	0,35
Empotrado	3,6	1,20	2,5	0,32

La controversia: líneas de código frente a puntos de función.

Existe en el mundo de la Ingeniería del Software una viva polémica sobre qué tipo de métricas son mejores para evaluar un proyecto: las orientadas a tamaño o las que utilizan puntos de función.

El centro de controversia está en considerar las líneas de código como medida clave, ya que los que se oponen a su uso, aducen que las medidas basadas en líneas de código son dependientes del lenguaje de programación. En cualquier caso esta polémica queda apartada gracias a Casper Jones, quien creó la siguiente tabla de correspondencia entre algunos de los lenguajes de programación más conocidos con su número de equivalencia entre líneas de código por punto de función:

CONVERSIÓN LÍNEAS DE CÓDIGO A PUNTOS DE FUNCIÓN

LENGUAJE	LDC/PF
Ensamblador	320
C	150
Cobol	106
Pascal	91
Basic	64
TCL	64
Java	53
C++	29

7.2. RELACIONES DE TAMAÑO Y COMPLEJIDAD PARA DESARROLLAR PROCESOS.

PROYECTO DE SOFTWARE TAMAÑO Y COMPLEJIDAD	CONSIDERACIONES DEL PROCESO DE DESARROLLO
PEQUEÑO Y NO COMPLEJO	<ul style="list-style-type: none">• Comprar un producto comercial existente.• Si no, se debe desarrollar uno para éste.• Usar un lenguaje 4GL para este tipo de aplicaciones.• Si no, utilizar un recurso existente o comprar un medio de desarrollo barato.• Esperar que el software permanezca independiente y se convierta en obsoleto dentro de algún año.

MEDIANO Y MODERADAMENTE COMPLEJO	<ul style="list-style-type: none"> • Incluir proyectos pequeños que puedan crecer en grandes proyectos. • Consultar planes de organizaciones de negocio para asegurarse la compatibilidad de las decisiones de software. • Considerar COSTES de los productos. • Si no, usar prácticas de ingeniería descritas abajo para grandes sistemas.
GRANDE Y MUY COMPLEJO	<ul style="list-style-type: none"> • Incluir proyectos medianos que puedan crecer en grandes proyectos. • Considerar COSTES de los productos satisfaciendo alguna de las partes que el sistema requiere. • Usar buenas y sólidas prácticas de ingeniería. • Consultar planes de organizaciones de negocio. • Elegir el lenguaje apropiado usando la Tabla 3. • Elegir el producto apropiado.
MUY GRANDE E INMENSAMENTE COMPLEJO	<ul style="list-style-type: none"> • Incluir proyectos grandes que puedan crecer en muy grandes proyectos. • Considerar COSTES de los productos satisfaciendo alguna de las partes que el sistema requiere. • Usar buenas y sólidas prácticas de ingeniería. • Consultar planes de organizaciones de negocio. • Elegir el lenguaje apropiado usando la Tabla 3. • Elegir el producto apropiado. • Control immense complexity from the outset. • Considerar definir separadamente subsistemas donde cada función es un sistema independiente.

8. ESTANDARIZACIÓN: VALORACIÓN LENGUAJES EN FUNCIÓN DE SUS CARACTERÍSTICAS.

LENGUAJE	4GL	3 GL							2GL
CARACTERÍSTICAS DE LOS LENGUAJES	S Q L	A d a 9 5	C	C + +	C O B O L	F O R T R A N	J a v a	S m a l l t a l k	A s s e m b l e r

LENGUAJE	4GL	3 GL							2GL
Claridad del código fuente	5	9	5	6	7	5	8	9	1
Complejidad de manejo (arquitectura de soporte)	2	9	5	6	2	4	7	6	2
Soporte de concurrencia	0	8	0	0	0	0	7	2	2
Soporte de sistemas distribuidos	0	5	0	0	0	0	7	0	0
Mantenimiento	5	9	2	7	2	2	9	7	0
Soporte de lenguajes mixtos	0	8	5	7	0	5	5	3	0
Soporte de programación orientado a objetos	0	10	0	10	0	0	10	10	0
Portabilidad	1	8	5	7	3	3	9	3	1
Soporte de tiempo-real	0	7	7	7	0	5	0	0	5
Confianza	3	9	1	5	3	1	8	3	0
Reusabilidad	1	8	3	8	3	3	8	8	1
Seguridad	0	6	0	3	0	0	4	0	0
Estandarización	1	10	5	5	5	5	8	3	0
Soporte para métodos de ingeniería modernos	3	9	1	7	1	1	9	7	0
Promedio de los lenguajes	1,5	8,2	2,8	5,6	1,8	2,4	7,1	4,35	0,8

BIBLIOGRAFÍA

- D. APPLEBY Y J.J. VANDEKOPPLE, *Lenguajes de programación: paradigma y práctica*, McGraw-Hill Interamericana, 1998.
- T.W. PRATT Y M.V. ZELKOWITZ, *Lenguajes de programación: diseño e implementación*, Prentice-Hall Hispanoamericana, 3.ª ed., 1998.
- R. SETHI, *Lenguajes de programación: conceptos y constructores*, Addison-Wesley Iberoamericana, 1992.

