UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

PARADIGMAS DE PROGRAMACIÓN Y CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN DE ACUERDO AL PARADIGMA QUE UTILIZAN

T R A B A J O E S C R I T O

E N L A M O D A L I D A D D E
A L T O N I V E L A C A D É M I C O

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A :

M A R I O O Z I E L M A R T Í N E Z R A M Í R E Z

ASESOR: M. en C. Marcelo Pérez Medel

México 2011

ÍNDICE

INTRODUCCIÓN

Los lenguajes de programación, al igual que los lenguajes naturales, están diseñados para facilitar la expresión y la comunicación de ideas entre las personas. Las ideas expresadas en lenguajes naturales cubren un amplio espectro de la comunicación humana incluyendo la prosa y la poesía, además de un amplio intervalo de materias. Sin embargo, los lenguajes de programación difieren de los naturales en dos aspectos importantes: para empezar, tienen un dominio expresivo más limitado ya que únicamente facilitan la comunicación de ideas algorítmicas entre las personas; en segundo lugar, estos lenguajes permiten también la comunicación de ideas algorítmicas entre personas y equipos computacionales. Así pues, el diseño de un lenguaje de programación debe responder a requisitos diferentes que el de un lenguaje natural.

El presente trabajo tiene por objetivo el presentar las diferentes metodologías o estilos de programación utilizadas hoy en día, conocidas también como “Paradigmas de Programación” los cuáles presentan sus principales diferencias dependiendo del tipo de aplicación que se desea desarrollar a partir de ellos y las necesidades del programador para implementar un problema de la vida real con una solución computacional.

Antes de empezar a hablar sobre las características principales de cada paradigma de programación, es importante saber por qué es fundamental el tener un conocimiento de fondo sobre los conceptos generales de ellos, por eso en el Capítulo 1 “Los Paradigmas de Programación” se habla sobre la importancia del estudio de los conceptos de programación y desarrollo de *software*.

Los cuatro paradigmas de programación más diferenciados por varios de los autores son el Paradigma de Programación Imperativo, el Funcional, Lógico y el Orientado a Objetos. Sin embargo, algunos autores los clasifican de acuerdo al tipo de sentencias que utilizan, misma clasificación se presenta también en el Capítulo 1 y a continuación se dan las principales áreas de aplicación sobre el desarrollo de *software*, esto se da como introducción a la descripción de cada uno de los paradigmas de programación contemplados en este trabajo y con esto sea más fácil entender por qué ciertos paradigmas presentan ciertas características en específico. En la parte final del

primer capítulo se presentarán las principales características de los cuatro principales paradigmas de programación, sus aplicaciones y el auge que tienen hoy en día entre la familia de programadores y desarrolladores de *software*.

En el Capítulo 2 “Clasificación de los Lenguajes de Programación de acuerdo al Paradigma que utilizan” se da una breve reseña sobre qué es un lenguaje de programación y la historia de cómo han ido evolucionando, pasando desde la codificación en sistema binario hasta los sistemas visuales con utilización de librerías preprogramadas y reutilizables. A partir de este punto, se dan algunos criterios de evaluación sobre los lenguajes de programación, para en dado caso, definir cuál herramienta es la más adecuada para encontrar la solución a un problema computable.

Al final del capítulo se dan los lenguajes de programación que se caracterizan con cada uno de los paradigmas descritos en el Capítulo 1; algunos de ellos están ya en desuso y algunos otros son los lenguajes de programación más utilizados actualmente.

En el Capítulo 3 “Conclusiones y comparación de los Paradigmas de Programación” se dan las conclusiones al presente trabajo tomando en cuenta las necesidades que requiera el desarrollador de *software* para implementar la solución a un problema de la vida real en forma computacional y retomando algunas de las principales características de los lenguajes de programación identificables con cada uno de los paradigmas de programación.

CAPÍTULO 1

"LOS PARADIGMAS DE PROGRAMACIÓN"

RAZONES POR LAS CUÁLES ESTUDIAR CONCEPTOS DE PROGRAMACIÓN

Para los estudiantes de informática, ingeniería en computación, ingeniería en sistemas y demás carreras afines, es muy importante el saber las diferentes técnicas de programación que existen hoy en día y que son las más utilizadas para el desarrollo de aplicaciones. A continuación se mencionan algunas de las principales razones por las cuáles tener conocimientos sobre los estilos y lenguajes de programación, así como los beneficios que esto trae.

Ilustración 1-1. Incremento en la capacidad para expresar las ideas del programador

- Incrementa la capacidad para expresar las ideas del programador.

Es evidente que la coherencia sobre lo que podemos pensar y expresar depende únicamente del lenguaje en el cuál expresemos nuestras ideas. Si estas ideas están basadas en un lenguaje limitado, también será limitada su complejidad, y el nivel de abstracción de las mismas. En otras palabras, para la gente es difícil el conceptuar estructuras que ellos mismos no pueden describir, ya sea de forma escrita o hablada. Los programadores que se encuentran en la etapa del desarrollo de *software*[1] se encuentran con el mismo problema. El lenguaje en el cuál se decide realizar alguna aplicación o sistema pone limites en la forma de controlar las estructuras, los datos, y las diferentes abstracciones que consideran importantes para plasmar un problema real en un programa de cómputo.

Si un programador no está consciente de la gran variedad de paradigmas de programación y además de los lenguajes que los utilizan como base, sus habilidades como desarrollador de *software*[1] se pueden ver limitadas. Los programadores pueden incrementar su rango de desarrollo de aplicaciones a través del aprendizaje de nuevas

[1] Término en inglés utilizado para llamar a las partes intangibles de una computadora, (programas, rutinas, datos, sistemas operativos, etc.).

técnicas de programación y los diferentes lenguajes que aportan beneficios sobre otros dependiendo de la técnica más apropiada para resolver el problema en cuestión.

Podría pensarse que el aprender las ventajas de algún lenguaje de programación no le ayudaría a un programador que es forzado a utilizar un determinado lenguaje en su trabajo y que carece de esas capacidades. Este argumento no es del todo válido, ya que algunas estructuras de un determinado lenguaje de programación pueden ser simuladas en otros que no soportan esas estructuras de manera directa.

Por ejemplo, si se aprenden las estructuras de los arreglos asociativos en Perl[2], un programador del lenguaje C[3] puede diseñar estructuras que simulen los arreglos asociativos en ese lenguaje.

El estudio de las técnicas de programación y de los lenguajes de programación construye una amplia visión de las capacidades de cada uno de los lenguajes y acercan a los programadores a utilizarlas.

El hecho de que algunas acciones de unos lenguajes de programación puedan ser simuladas en otros no resulta siempre positivo ya que es mejor utilizar las estructuras en el lenguaje para el cuál fueron diseñadas e incorporadas que en aquellos que tratan de imitarlas del todo pero que resultan más difíciles su aprendizaje y aplicación, además de que pueden ser menos elegantes para su escritura y menos seguras en aquellos lenguajes que no las soportan del todo.

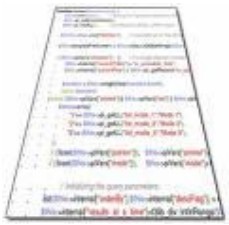

Ilustración 1-2. Provee de un conocimiento de fondo para la elección de la técnica y lenguaje apropiados

- Provee de un conocimiento de fondo para la elección de la técnica y lenguaje apropiados.

Muchos programadores profesionales han tenido una pequeña educación formal en ciencias de la computación; sin embargo, han aprendido técnicas de programación por su propia cuenta ya sea en cursos o en la práctica realizada en la misma casa o trabajo. Algunos lenguajes de programación que se aprenden son los más relevantes para el desarrollo de proyectos dentro de la oficina, sin embargo, hay otros lenguajes de programación que se aprendieron alguna vez y que son olvidados ya que nunca más son puestos en práctica por el programador. Por este motivo, cuando el desarrollador de *software*[1] es destinado a realizar un nuevo sistema

[2] Remítase a la página 66 para la descripción del lenguaje de programación Perl
[3] Remítase a la página 63 para la descripción del lenguaje de programación C

utiliza la técnica y el lenguaje de programación con los cuáles está familiarizado aún cuando éste ya sea obsoleto en comparación con los nuevos paradigmas de programación o con los nuevos lenguajes que se están utilizando y que pueden proporcionar ventajas notables sobre el que ha estado desarrollando en los últimos años.

Si estos programadores tuvieran conocimientos sobre las nuevas técnicas de programación y las nuevas capacidades que tienen los lenguajes más recientes, y mejor aún, supieran explotarlas al máximo, tendrían una amplia visión sobre qué paradigma y lenguaje son los más apropiados para la solución del problema que se le está presentando.

Ilustración 1-3. Incrementa la habilidad para aprender nuevas técnicas y lenguajes de programación

- Incrementa la habilidad para aprender nuevas técnicas y lenguajes de programación.

La programación de computadoras continúa siendo relativamente una disciplina joven. El diseño de metodologías, desarrollo de herramientas de *software*[1] y lenguajes de programación continúan en un estado de evolución continua. Esto hace al desarrollo de *software*[1] una profesión excitante, pero también significa que el aprendizaje continuo es esencial para un programador que quiere sobresalir. El proceso de aprendizaje puede resultar lento y tedioso principalmente para aquellos programadores que se han visto envueltos y a tal grado han llegado a ser conformistas con uno o dos lenguajes de programación y que no han estudiado los conceptos de programación en una forma general. Una vez que se han adquirido los conocimientos generales sobre los diferentes paradigmas de programación y las características de los diferentes lenguajes que existen, el proceso de aprendizaje de las nuevas metodologías y herramientas será más fácil al ver cómo estos conceptos generales son incorporados en el diseño basado en un lenguaje que se esté aprendiendo.

Por ejemplo, los programadores que comprenden el concepto del paradigma de programación orientada a objetos tendrán un aprendizaje mucho más rápido y fácil del lenguaje de programación Java[4] que aquellos que nunca habían oído hablar de esos conceptos. El mismo fenómeno ocurre en los lenguajes naturales. Lo mejor de conocer la gramática de nuestro idioma natural es la facilidad con la que podremos comprender

[4] Remítase a la página 78 para la descripción del lenguaje de programación Java

un segundo idioma. Aunado a esto, el aprender un segundo lenguaje trae el beneficio de enseñarnos más acerca de nuestro lenguaje nativo.

Finalmente, es esencial para la práctica de los programadores, el conocer el vocabulario y conceptos fundamentales de los diferentes lenguajes de programación existentes ya que esto hace más fácil la lectura y comprensión de código, manuales de lenguajes de programación y de artículos o libros que hablen acerca de las metodologías y los lenguajes de programación.

- Mejor comprensión sobre la manera de implementación.

Ilustración 1-4. Mejor comprensión sobre la manera de implementación

En el aprendizaje de los conceptos de los paradigmas y lenguajes de programación, es tanto interesante como necesario el ver como la implementación de algunos problemas puede verse afectada en el diseño de los mismos lenguajes de programación. En algunos casos, al estar desarrollando un sistema de *software*[1] y al estar utilizando el lenguaje de programación más apropiado para resolverlo podemos comprender por qué éste lenguaje fue diseñado de la manera en que está hecho. Esto nos lleva a utilizar más inteligentemente las técnicas y los lenguajes de programación, esto es, utilizándolos en la manera para la cuál fueron diseñados. Podemos llegar a ser mejores programadores si estudiamos las diferentes opciones que nos dan los lenguajes de programación con sus estructuras soportadas directamente y sabiendo las consecuencias que nos traerá la elección de una técnica o lenguaje de programación sobre los demás.

Cierto tipo de errores de programación pueden ser encontrados y solucionados solamente por aquellas personas que estuvieran envueltas en el desarrollo del mismo, que saben de qué manera se trató de resolver el problema con el *software*[1] en cuestión y que saben por qué fue escogido ese lenguaje de programación para la implementación del sistema.

DEFINICIÓN Y CLASIFICACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN

Un paradigma está constituido por los supuestos teóricos generales, las leyes y las técnicas para su aplicación que adoptan los miembros de una determinada comunidad científica.

- Las leyes explícitamente establecidas y los supuestos teóricos. Por ejemplo, las leyes de movimiento de Newton forman parte del paradigma newtoniano y las ecuaciones de Maxwell forman parte del paradigma que constituye la teoría electromagnética clásica.
- El instrumental y las técnicas instrumentales necesarios para hacer que las leyes del paradigma se refieran al mundo real. La aplicación en astronomía del paradigma newtoniano requiere el uso de diversos telescopios, junto con técnicas para su utilización y diversas metodologías para interpretar los datos recopilados.
- Un componente adicional de los paradigmas lo constituyen algunos principios metafísicos muy generales que guían el trabajo dentro del paradigma. Todos los paradigmas, además, contienen prescripciones metodológicas muy generales tales como: "Hay que intentar seriamente compaginar el paradigma con la naturaleza".

Podemos decir que, los paradigmas son marcos de referencia que imponen reglas sobre cómo se deben hacer las cosas, indican qué es válido dentro de él y qué está fuera de sus límites, es un determinado marco desde el cuál miramos el mundo, lo comprendemos, lo interpretamos e intervenimos en él. Puede abarcar desde el conjunto de conocimientos científicos que imperan en una época determinada hasta las formas de pensar y de sentir de la gente en un determinado lugar y momento histórico. Un paradigma distinto implica nuevas reglas, elementos, límites y maneras de pensar lo que se traduce en un cambio; establecen los límites dentro de los cuáles se resuelven los problemas, mejorando o proporcionando nuevas soluciones, filtrando las experiencias, percepciones y creencias creando así el “efecto paradigma”.

Científicamente, el paradigma debe ser concebido como una forma aceptada de resolver un problema y que más tarde es utilizada como modelo para la investigación y la formación de una teoría. También el paradigma debe ser concebido como un conjunto de métodos, reglas y generalizaciones utilizadas conjuntamente por aquellas personas capacitadas para realizar el trabajo científico de investigación.

En el contexto de la computación, los paradigmas pueden ser considerados como patrones de pensamiento para la resolución de problemas desde luego siempre teniendo en cuenta los lenguajes de programación según nuestro interés de estudio, es decir.

Los lenguajes de programación permiten un amplio margen de expresión algorítmica que soporta una gran variedad de aplicaciones de computación a través de varios dominios de aplicación como los sistemas de cómputo científico o de gestión de la información. Para conseguir semejante versatilidad, las distintas comunidades de programación de estos dominios han desarrollado caminos especiales y diferentes, o paradigmas, para expresar algoritmos que se ajustan especialmente bien a sus propias áreas de aplicación.

Si buscamos en un diccionario, la palabra *paradigma* significa:

> *Conjunto de teorías generales, suposiciones, leyes o técnicas de que se vale una escuela de análisis o comunidad científica para evaluar todas las cosas.*[5]

Así mismo, la palabra *programación* tiene el siguiente significado:

> *Preparación en un ordenador, de una secuencia detallada de instrucciones operativas para el desarrollo y solución de un problema particular.*[6]

De lo anterior podemos concluir que un paradigma de programación es una forma de interpretar, manipular y representar el conocimiento. Representa un enfoque o filosofía particular para la construcción del *software*[1] y así plantear una metodología para la solución del problema.

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas, que permite producir sistemas con unas directrices específicas, tales como:

[5] Diccionario Enciclopédico Grijalbo, Editorial Grijalbo, España, 2000
[6] Ibid.

estructura modular, fuerte cohesión, alta rentabilidad, etc. El resultado de esto es el modelado del proceso de diseño que determina la estructura del programa.

La estructura conceptual del modelo está pensada para determinar la forma correcta de los programas y controle el modo en el que pensamos y formulamos soluciones a los problemas que se presenten, para que una vez que lleguemos a ella, ésta sea expresada mediante un lenguaje de programación. Para que éste proceso sea efectivo, las características del lenguaje deben reflejar adecuadamente las características del paradigma de programación.

Ilustración 1-5. Los paradigmas de programación determinan la estructura de los programas dependiendo de la solución que busque el programador

CLASIFICACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN DE ACUERDO A LA CAPACIDAD DE EXPRESIÓN

Floyd describió tres categorías de paradigmas de programación de acuerdo a la capacidad de expresión:

a) Los que soportan técnicas de programación de bajo nivel.

La actividad de estos paradigmas de programación se aleja de la forma de pensar que se busca en los lenguajes de programación más actuales ya que basan su funcionamiento en una operación similar a la del *hardware*[7] del ordenador, tal y como o hacen los componentes físicos de la máquina.

[7] Término inglés que hace referencia a cualquier componente físico tecnológico, que trabaja o interactúa de algún modo con la computadora

Se puede decir que en este esquema operan los lenguajes de segunda generación, tales como versiones primarias del BASIC[8], FORTRÁN[9] o COBOL[10], los cuáles trataban de emular al lenguaje ensamblador perteneciente a la primera generación y que era capaz de comandar directamente sobre la operación de la máquina accediendo a los componentes disponibles del *hardware*[7].

b) Los que soportan métodos de diseño de algoritmos.

A este tipo de paradigmas pertenecen los que abarcan los lenguajes de tercera generación que ya poseen capacidad de manejar procedimientos, funciones, variables locales y punteros para la asignación dinámica de memoria. A esta categoría pertenecen actualmente la mayoría de los lenguajes de programación ya que manejan con solvencia el manejo del diseño algorítmico. El lenguaje C[3] es un caso especial ya que aunque bien podría pertenecer a esta categoría, debido a su capacidad de manejo directo sobre el *hardware*[7] de la máquina, Floyd lo colocó dentro de los lenguajes de bajo nivel.

c) Los que soportan soluciones de programación de alto nivel.

Estos paradigmas recurren para su codificación a los lenguajes orientados a objetos que tienen entre sus primeros representantes a PROLOG[11] aunque actualmente los lenguajes más utilizados en este ámbito son C++[12], Java, Visual Basic[13] y todos los derivados de los mismos.

Para Floyd, las soluciones de programación de alto nivel, pasan de usar la lógica como método de programación a utilizar las reglas de producción y la programación funcional. Estas variantes de la programación tradicional estarían soportadas por lenguajes como PROLOG, ML y LIPS; o bien, por entornos de ingeniería del conocimiento como ART, KC o KEE, los cuáles son utilizados principalmente en el ámbito académico y no tanto en la industria.

[8] Remítase a la página 62 para la descripción del lenguaje de programación BASIC
[9] Remítase a la página 65 para la descripción del lenguaje de programación FORTRÁN
[10] Remítase a la página 64 para la descripción del lenguaje de programación COBOL
[11] Remítase a la página 70 para la descripción del lenguaje de programación PROLOG
[12] Remítase a la página 72 para la descripción del lenguaje de programación C++
[13] Remítase a la página 81 para la descripción del lenguaje de programación Visual Basic

CLASIFICACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN SEGÚN EL TIPO DE PROGRAMACIÓN

Debido a que los paradigmas también son colecciones de modelos conceptuales que definen procesos de diseño destinados a determinar la estructura, sintaxis y semántica de los programas, el tipo de programación utilizada puede asimilarse en los siguientes tipos:

a) Programación basada en reglas.

De aplicación en la ingeniería del conocimiento para desarrollar sistemas expertos con núcleo de reglas de producción del tipo *if then.*

b) Programación lógica.

Entorno de programación conversacional, deductivo, simbólico y no determinista apoyado en asertos y reglas lógicas.

c) Programación funcional.

Entorno de programación interpretativo, funcional y aplicativo, de formato funcional.

d) Programación heurística.

En este tipo de programación, se moldean los problemas para aplicar heurísticas según sistemas de visualización, búsqueda y métodos de solución que pueden ser:

- Programación paralela.
- Programación orientada a objetos.
- Programación basada en restricciones.
- Programación basada en flujo de datos.

CLASIFICACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN SEGÚN EL TIPO DE SOLUCIÓN

Otra forma de clasificar a los paradigmas de programación es de acuerdo al tipo de solución que aportan para resolver un problema, así tenemos por ejemplo:

a) Paradigmas de solución procedimental u operacional.

Este tipo de paradigmas describen etapa a etapa el modo de construir la solución describiendo cómo obtener un resultado a partir de un estado inicial.

Los paradigmas procedimentales fueron los utilizados originalmente en la historia de los lenguajes de programación y quizá todavía son los que se encuentran en uso más común, incluyendo los lenguajes clásicos de la primera a la tercera generación, hasta los orientados a objetos y funcionales.

La característica fundamental de estos paradigmas de programación es la secuencia computacional realizada etapa a etapa para resolver el problema. Su principal desventaja es determinar si el valor computado es una solución correcta al problema, por lo que se han desarrollado multitud de técnicas de depuración y verificación para probar la corrección de los sistemas desarrollados basándose en este tipo de paradigmas.

Pueden ser de dos tipos básicamente: los que actúan modificando repetidamente la representación de sus datos (con efecto de lado); y los que actúan creando nuevos datos continuamente (sin efecto de lado).

Los paradigmas con efecto de lado utilizan un modelo en el que las variables están estrechamente relacionadas con direcciones de la memoria del ordenador. Cuando se ejecuta el programa, el contenido de estas direcciones se actualiza rápidamente, pues las variables reciben múltiples asignaciones, y al finalizar el trabajo, los valores finales de las variables representan el resultado.

Existen dos tipos de paradigmas con efectos de lado:

- El paradigma de programación imperativo
- El paradigma de programación orientado a objetos

Los paradigmas de programación sin efecto de lado son aquellos que van creando continuamente nuevos datos y aunque no incluyen a los que tradicionalmente son denominados paradigmas funcionales, es importante distinguir la solución funcional procedimental de la solución funcional declarativa.

Los paradigmas procedimentales definen la secuencia explícitamente, pero esta secuencia se pude procesar en serie o en paralelo. En este segundo caso, el procedimiento paralelo puede ser asíncrono (cooperación de procesos paralelos) o asíncrono (procesos simples aplicados simultáneamente a muchos objetos).

b) Paradigmas de solución declarativa.

En este tipo de paradigmas de programación, un programa se construye señalando hechos, reglas, restricciones, ecuaciones, transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución.

A partir de esta información el sistema debe de proporcionar un esquema que incluya el orden de evaluación que compute una solución. Aquí no existe la descripción de las diferentes etapas a seguir para alcanzar una solución como se dan en los paradigmas procedimentales.

Estos paradigmas permiten el uso de variables para almacenar valores intermedios, pero no para actualizar estados de información.

Se dice que una operación computacional es declarativa si en cualquier ocasión en que es llamada a ejecutarse, regresa los mismos resultados independientemente del estado de cualquier otro programa. Una operación declarativa es independiente (no depende de ningún otro estado de operación que se encuentre fuera de ella), no establece estados de control sobre otras rutinas (no tiene ejecución de algún estado interno que sea enviado entre rutinas) y es determinista (siempre entrega el mismo resultado cuando es llamada con los mismos argumentos).

Dado que estos paradigmas especifican la solución sin indicar cómo construirla, desde el principio eliminan la necesidad de probar que el valor calculado es el valor esperado para la solución. En la práctica, algunos lenguajes de programación que entran en este tipo de paradigmas, aunque no utilizan ni secuencias de control ni requieren la noción de estado, resuelven los problemas con soluciones construidas y no por medio de especificaciones. Es por tal motivo que estos lenguajes de programación no se consideran declarativos si no pseudodeclarativos. Algunos ejemplos de lenguajes declarativos son el Datatrieve, PROLOG[11], SQL[14] y los lenguajes específicos de la Ingeniería del Conocimiento, los cuáles utilizan representaciones complejas del conocimiento sobre un problema para manipularlas y generar soluciones de forma declarativa.

El concepto de expresiones regulares (regular expressions en inglés) se refiere a una familia de lenguajes compactos y potentes para la descripción de conjuntos de cadenas de caracteres. Numerosos editores de texto y otras utilidades (especialmente en el sistema operativo UNIX), como por ejemplo sed y awk, utilizan estos lenguajes para buscar ciertas estructuras en el texto y, por ejemplo, remplazarlas con alguna otra cadena de caracteres.

Los lenguajes pseudodeclarativos requieren al menos un limitado grado de secuencia y por lo tanto admiten versiones en serie y paralelo.

[14] Remítase a la página 66 para la descripción del lenguaje de programación SQL

Dentro del grupo de paradigmas de programación declarativos se encuentran el funcional y el lógico.

c) Paradigmas de solución demostrativa.

Cuando se programa bajo un paradigma demostrativo (también llamado programación por ejemplos), el programador no especifica procedimentalmente cómo construir la solución. En su lugar, presenta soluciones de problemas similares y permite al sistema que generalice una solución procedimental a partir de estas demostraciones. Los esquemas individuales para generalizar tales soluciones van desde simular una secuencia procedimental o inferir intenciones.

Los sistemas que infieren, intentan generalizar usando razonamiento basado en el conocimiento. Una solución basada en la inferencia intenta determinar en qué son similares un grupo de datos u objetos, y, a partir de ello, generalizar estas similitudes.

Otra solución es la programación asistida en donde el sistema observa acciones que el programador ejecuta, y si son similares o acciones pasadas, intentará inferir cuál es la próxima acción que hará el programador. Las dos principales objeciones al sistema de inferencia son:

- Si no se comprueban exhaustivamente pueden producir programas erróneos que trabajan correctamente con los ejemplos de prueba, pero que fallen posteriormente en otros casos.
- La capacidad de inferencia es tan limitada, que el usuario debe de guiar el proceso en la mayoría de los casos.

Los resultados más satisfactorios de los sistemas de inferencia son en áreas limitadas, donde el sistema tenía un conocimiento semántico importante de la aplicación.

El mayor problema que se presenta con estos sistemas, es conocer cuándo en programa es correcto. En el caso de los sistemas procedimentales, se consigue estudiando el algoritmo y el resultado de juegos de ensayo apropiados.

En el caso de los sistemas demostrativos el algoritmo se mantiene en una representación interna, y su estudio se sale del ámbito de estos sistemas. Por lo que la veracidad de la decisión se debe hacer exclusivamente sobre la base de la eficiencia del algoritmo sobre los casos específicos de prueba.

La programación demostrativa es del tipo "*bottom-up*" y se adapta bien a nuestra capacidad de pensar. Si embargo, en la mayor parte de los paradigmas la resolución del problema se efectúa aplicando métodos abstractos de "*top-down*".

Existe una variedad de paradigmas de programación atendiendo a alguna particularidad metodológica o funcional, los principales paradigmas son:

- La programación imperativa: el programa es una serie de pasos, cada uno de los cuales realiza un cálculo, recupera una entrada o tiene como resultado una salida. La abstracción procedimental es un bloque de creación esencial en la programación imperativa, al igual que las sentencias condicionales, asignaciones, bucles y secuencias. Se caracteriza por la ejecución secuencial de instrucciones, el uso de variables en representación de localizaciones de memoria y el uso de la asignación para cambiar el valor de las variables. Los lenguajes de programación imperativa principales son Cobol[10], Fortrán[9], C[3] y C++[12].
- Programación Orientada a Objetos (POO): este tipo de programación ha tomado una enorme importancia en la última década y se caracteriza porque el programa es una colección de objetos que interactúan los unos con los otros trasladándose mensajes que transforman su estado. El modelado, la clasificación y la herencia de objetos son bloques de creación esenciales de la POO. Los lenguajes orientados a objetos principales son Smalltalk[15], Java[4], C++[12] y Eiffel[16].
- Programación funcional: El programa es una colección de funciones matemáticas, cada una de ellas con su entrada (dominio) y su resultado (intervalo). Las funciones interactúan y se combinan las unas con las otras utilizando condicionales, recursividad y composición funcional. Define un entorno de programación interpretativo, funcional y aplicativo. Los lenguajes de programación funcional principales son Lisp[17], Scheme[18], Haskell[19] y ML. Tiene una gran aplicación en el desarrollo de sistemas de inteligencia artificial y de sistemas expertos.
- Programación lógica (declarativa): el programa es una colección de declaraciones lógicas sobre el resultado que debería conseguir una función, en lugar de cómo debería conseguirse dicho resultado. La ejecución del

[15] Remítase a la página 80 para la descripción del lenguaje de programación SMALLTALK
[16] Remítase a la página 75 para la descripción del lenguaje de programación Eiffel
[17] Remítase a la página 68 para la descripción del lenguaje de programación Lisp
[18] Remítase a la página 69 para la descripción del lenguaje de programación Scheme
[19] Remítase a la página 67 para la descripción del lenguaje de programación Haskell

programa aplica estas declaraciones para obtener una serie de soluciones posibles a un problema. La programación lógica ofrece un vehículo natural para la expresión no determinista, ya que las soluciones a muchos problemas no son únicas, sino múltiples. Esta programación está basada en asertos y reglas lógicas que definen un entorno de programación de tipo conversacional, deductivo, simbólico y no determinista. El lenguaje de programación lógica más importante es el Prolog[11].

La importancia de los dos últimos paradigmas mencionados es su correspondencia con las bases matemáticas, que permiten que el comportamiento del programa sea descrito de manera abstracta y precisa, haciendo por lo tanto mucho más fácil juzgar si un programa se ejecutará correctamente, permitiendo la escritura de código muy preciso, incluso para tareas muy complicadas.

Para algunos resulta sorprendente que existan varios paradigmas de programación. La mayoría de los programadores está familiarizado con uno solo: el de la programación procedimental (imperativa), y esto se debe a que también es el más difundido en las escuelas, universidades e institutos que se dedican a la enseñanza de la programación.

En realidad, no existe un paradigma que sea mejor que otro para la construcción de los programas de cómputo, de ahí que una de sus características fundamentales es que sean inconmensurables, tampoco existe una tendencia a seguir para escribir programas en determinados lenguajes que se basan en uno u otro paradigma de programación; en vez de esto, los paradigmas de programación son más apropiados que otros en determinadas situaciones que se desean programar, por ejemplo, no es lo mismo programar un sistema de inteligencia artificial que trata de determinar cuándo sucederá un sismo, a programar un sistema de nómina de una empresa privada. Por tal motivo, es de vital importancia, antes de empezar a escribir un programa para resolver una situación que se desea automatizar, el estudio, análisis y conceptualización del problema que se desea solucionar; plasmar las ideas en un pseudocódigo visualizando las acciones que llevará a cabo el programa para resolverlo y a partir de esto, determinar que metodología o paradigma de programación es el más adecuado para resolver dicho problema.

Cuando un lenguaje de programación refleja bien a un cierto paradigma, se dice que soporta el paradigma, y en la práctica, un lenguaje que soporta correctamente un paradigma es difícil distinguirlo del propio paradigma, por lo que se identifica con él.

Uno podría tomar un paradigma de programación "puro" para realizar un programa o sistema computacional y resolver satisfactoriamente el problema que tenga en cuestión, sin embargo, la práctica de los programadores los ha llevado a utilizar lenguajes de programación que mezclan distintos paradigmas, tal es el caso por ejemplo de lenguajes como C++[12], Delphi[20], Visual Basic[13] o Java[4] que combinan el lenguaje imperativo con el orientado a objetos. Otros casos son el Scheme[18] (basado en paradigma funcional) y el Prolog[11] (basado en paradigma lógico) que utilizan estructuras repetitivas que son una forma de representación del paradigma imperativo. Existen lenguajes de programación catalogados como multiparadigma que permiten la mezcla de varios de ellos en una forma totalmente natural para el programador, tal es el caso de Oz y el lenguaje experimental Leda.

[20] Remítase a la página 74 para la descripción del lenguaje de programación Delphi

DOMINIOS DE APLICACIÓN

Las computadoras han sido utilizadas en una infinidad de áreas, desde el control de plantas nucleares, hasta una simple agenda personal. Debido a que hay una gran diversidad en el uso de las computadoras, cualquiera que sea este su tamaño, se han diseñado lenguajes de programación que tienen por objetivo ser la herramienta principal para el desarrollo de diferentes sistemas con características muy particulares y diferentes unos de otros.

Podemos agrupar las comunidades de programación que representan los distintos dominios de aplicación de la siguiente forma:

- Computación científica
- Sistemas de gestión de información
- Inteligencia artificial
- Sistemas
- Centrada en la Web

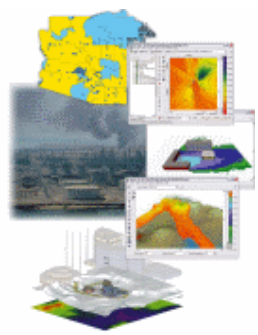

Ilustración 1-6. Dominio de aplicación científico

- Científica.

La comunidad de programación científica surgió muy pronto en la historia de la computación; los primeros programas se escribieron en los años cuarenta para predecir las trayectorias de los misiles durante la Segunda Guerra Mundial, utilizando la conocida fórmula física que caracteriza a los cuerpos en movimiento. Estos programas los escribían los matemáticos especialmente capacitados en lenguaje máquina y ensamblador. El primer lenguaje de programación científica fue el Fortrán[9]. ALGOL 60[21] y varios de sus descendientes fueron incursionados para su uso en ésta área también, aunque también fueron diseñados para ser utilizados en otras áreas.

Típicamente, las aplicaciones científicas tienen estructuras de datos sencillas, pero requieren del poder para manejar números grandes en punto flotante y una gran cantidad de operaciones aritméticas. La programación científica se preocupa principalmente de la realización de cálculos complejos de forma muy rápida y precisa. Los cálculos están definidos por modelos matemáticos que representan fenómenos

[21] Remítase a la página 61 para la descripción del lenguaje de programación Algol

científicos. Se implementan principalmente a partir del paradigma de programación imperativa.

La estructura de datos más común son los arreglos y las matrices, las estructuras de control más utilizadas son los ciclos de conteo y las estructuras de selección. Los lenguajes de programación de alto nivel inventados para aplicaciones científicas fueron diseñados para satisfacer esas necesidades.

El paradigma de la programación paralela está muy motivado por las necesidades de las aplicaciones científicas, como los modelos de sistemas climáticos o de flujos oceánicos.

Los lenguajes de programación científica incluyen Fortrán[9] 90 y Fortrán de alto rendimiento.

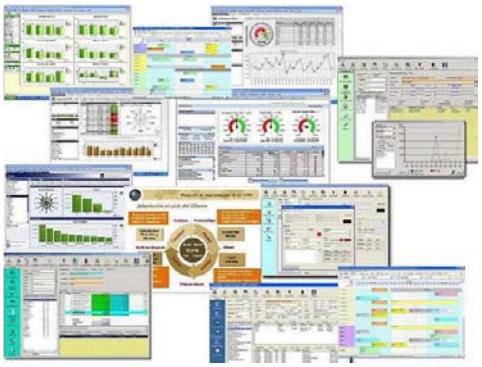

Ilustración 1-7. Dominio de aplicación de gestión de información (MIS)

- Sistemas de gestión de información (MIS).

Los programas diseñados para que las instituciones los utilicen con el propósito de gestionar sus sistemas de información comenzaron en la década de 1950 y son probablemente los más numerosos del mundo. Estos sistemas incluyen sistemas de nóminas, contabilidad, ventas en línea, marketing, de inventario y producción de la empresa, etc., y por eso puede decirse que también son aplicaciones para negocios. Estos sistemas se desarrollan en lenguajes de programación como Cobol[10] (que fue el primer lenguaje de alto nivel utilizado para fines de negocios que tuvo éxito), RPG, y SQL[14]. Cobol soporta un estilo de programación imperativa, mientras que RPG y SQL son herramientas de programación declarativas para la definición de archivo, la generación de informes y la recuperación de información de las bases de datos.

Los lenguajes utilizados para la gestión de la información son caracterizados por la facilidad de producir reportes ya elaborados, la precisión en la descripción y almacenaje de información, y la habilidad para especificar las operaciones aritméticas decimales.

Las aplicaciones de comercio electrónico utilizan frecuentemente un modelo "cliente – servidor" para el diseño de los programas, en el que dicho programa interactúa con los usuarios situados en sitios remotos y ofrece acceso simultáneo a

bases de datos compartidas. La programación guiada por eventos es esencial en estas aplicaciones, y los programadores utilizan para la implementación lenguajes como Java[4] y Tcl/Tk.

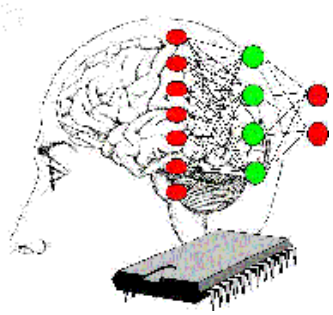

Ilustración 1-8. Dominio de aplicación en la inteligencia artificial

- **Inteligencia artificial.**

La comunidad de programación de inteligencia artificial está en activo desde los años sesenta. Se dedican al desarrollo de programas que imitan el comportamiento de la inteligencia humana, la deducción lógica y la cognición. Sus herramientas más importantes han sido la manipulación de símbolos, las expresiones funcionales y el diseño de sistemas de prueba lógica, más que el cálculo numérico.

El cálculo simbólico significa que los símbolos (en lugar de números) son manipulados. Para este efecto, las listas ligadas unas a otras son más utilizadas, desplazando así a su equivalente en otros paradigmas de programación que son los arreglos.

Tanto el paradigma de programación funcional como el de programación lógica han estado muy relacionados con la programación de inteligencia artificial, utilizando los mismos lenguajes descritos anteriormente para estos paradigmas.

El primer lenguaje de programación utilizado para fines de crear un sistema de inteligencia artificial fue el Lisp[17] que apareció en 1959. En la década de 1970 apareció una alternativa para el desarrollo de este tipo de aplicaciones utilizando el paradigma de la programación lógica y llamado Prolog[11].

Ilustración 1-9. Dominio de aplicación en el desarrollo de sistemas

- **Sistemas.**

Los programadores de sistemas son aquellos que diseñan y mantienen el *software*[1] básico que ejecutan los sistemas (componentes del sistema operativo, *software* de red, compiladores y depuradores del lenguaje de programación, equipos e intérpretes virtuales, etc.). Este *software* es el más utilizado y por lo tanto debe de ser el más eficiente, lo que hace que los lenguajes de programación utilizados para este fin deben de proveernos una ejecución rápida de su código y deben de permitir al programador escribir acciones que

le permitan al ordenador manipular, escribir y leer información a muy bajo nivel, incluso sobre aquellos dispositivos externos a la computadora.

Algunos de estos programas están escritos en lenguaje ensamblador de los distintos equipos de procesador, mientras que otros están escritos en un lenguaje diseñado específicamente para la programación de sistemas. En las décadas de 1960 y 1970 algunos fabricantes de computadoras como IBM, Digital y Burroughs (ahora UNISYS), desarrollaron lenguajes especiales de programación de alto nivel orientados a la máquina para desarrollar sistemas que utilizarían sus equipos en fabricación. Así pues, para las supercomputadoras de IBM se creó el PL/S que es un dialecto del PL/I; para Digital, fue el BLISS, un lenguaje que en nivel estaba justo arriba del lenguaje ensamblador; y para Burroughs, fue el Algol Extendido.

El sistema operativo UNIX fue escrito casi por completo en el lenguaje de programación C[3], lo que lo ha hecho fácil de portar o mover de una máquina a otra. Algunas de las características de C lo hacen buena opción para el desarrollo de sistemas. Estas son su capacidad de manipulación de la máquina a bajo, una ejecución eficiente y el no agobiar a los programadores con tantas restricciones.

Esta programación, se lleva a cabo, principalmente, a partir del paradigma imperativo, aunque también se debe de tratar a nivel fundamental los paradigmas paralelo y guiado por eventos.

Ilustración 1-10. Dominio de aplicación centrado en la web

- **Centrada en la Web.**

Los lenguajes aplicados para el desarrollo de sistemas W*eb* han sido un punto de atención muy importante de los programadores en los últimos 25 años.

El concepto de la computación centrada en la *Web* y, por tanto, de la programación centrada en la *Web*, deriva de un modelo interactivo, en el que un programa permanece en un bucle infinito, esperando que llegue la siguiente solicitud o evento, respondiendo a dicho evento y volviendo a su estado de bucle.

Los lenguajes de programación que soportan estos sistemas necesitan un paradigma que fomente la interacción entre el usuario y el sistema, o programación guiada por eventos. La programación de sistemas *Web* también se facilita de la programación orientada a objetos, ya que varias de las entidades residentes en la

pantalla del usuario se pueden modelar de forma más natural como objetos que envían y reciben mensajes los unos a los otros.

Ilustración 1-11. Dominio de aplicación de propósito específico

- **Sistemas de propósito específico.**

Un importante grupo de lenguajes de programación de propósito específico han aparecido alrededor de los últimos 40 años. Este grupo abarca desde el RPG, utilizado para producir reportes de negocios; el APT, utilizado para programar instrucciones y herramientas de máquinas programables; y el GPSS, utilizado para sistemas de simulación, por mencionar algunos.

ELECCIÓN DEL PARADIGMA DE PROGRAMACIÓN

Actualmente podemos encontrar una gran variedad de sistemas computacionales escritos en un sin fin de lenguajes de programación y con distintas metodologías de programación. Así pues, observamos aplicaciones muy comunes para nosotros como son los procesadores de texto o las hojas de cálculo, sistemas de gestión de datos de empresas, juegos que utilizan muchos recursos de la máquina debido a los gráficos y toda la gama de caminos por la que nos puede llevar, se observan sistemas que sirven para rastreo y monitoreo de fenómenos naturales, sistemas que deben de tener una fina precisión en sus cálculos para el trazo de rutas de naves espaciales, o aplicaciones de propósito específico como son el caso de los microcontroladores o los programas que hacen funcionar correctamente los teléfonos celulares.

Como podemos ver, existe una gran diversificación en los objetivos que buscan cada uno de estos sistemas, y es de acuerdo a ellos que se escoge en primer lugar la metodología de programación que se utilizará para su diseño y con base en esto, la elección del lenguaje de programación más óptimo para el desarrollo del mismo.

En el momento de adoptar un estilo de programación para el desarrollo de un sistema se deben de tomar en cuenta deseablemente una gran cantidad de aspectos que se someterán a evaluación para saber si el paradigma de programación es el más óptimo de acuerdo a los objetivos que buscamos conseguir en nuestro proyecto.

A continuación se citan una serie de aspectos que se deben de considerar en el momento de la elección del estilo de programación para el desarrollo de un programa:

- Los programas de cómputo se idean buscando conseguir ciertos objetivos. Es por este motivo que la metodología que se elija para el desarrollo del mismo debe de recoger el aspecto filosófico del sistema deseado, es decir, que los objetivos generales del proyecto deben de estar implícitos en el paradigma elegido.
- El estilo de programación debe de considerar y cubrir con el ciclo entero del desarrollo de la aplicación, que consta de la investigación, el análisis de los requisitos y el diseño.
- El paradigma de programación debe de integrar las distintas fases del ciclo de desarrollo: debe de cumplir con una rastreabilidad en donde sea posible

referirse a otras fases de un proyecto y fusionarlo con las fases previas, siendo posible no sólo poder moverse hacia delante, sino hacia atrás de forma que se pueda comprobar el trabajo realizado y se puedan efectuar correcciones; además, debe de haber facilidad de interacción entre las etapas del ciclo de desarrollo ya que se necesita una evaluación formal de cada fase antes de pasar a la siguiente, ya que si esto no es posible, la información que se pierda en una fase determinada queda perdida para siempre, con un impacto fatal en el sistema resultante.

- El estilo de programación utilizado debe de incluir la realización de validaciones, ya que se deben de detectar y corregir los errores cuanto antes. Uno de los problemas más frecuentes y costosos en el aplazamiento de la detección y corrección de problemas en las etapas finales del proyecto. Por tal motivo, cada fase del proceso de desarrollo de *software*[1] debe incluir una etapa de validación explícita.
- El paradigma elegido debe de ser la base de una comunicación efectiva entre los informáticos y el área usuaria. Deben de proporcionarse pasos bien definidos para realizar progresos visibles durante el desarrollo del *software*.
- El estilo de programación debe de desarrollarse en un entorno dinámico orientado al usuario. A lo largo de todo el ciclo del desarrollo del sistema se debe de hacer un intercambio de opiniones entre los desarrolladores del sistema y los usuarios para definir exactamente las necesidades del usuario, las cuáles están en constante evolución. Por parte de los informáticos, se involucra a los usuarios en el análisis, diseño y administración de datos para que se tomen las decisiones correctas.
- El estilo de programación elegido debe de soportar la eventual evolución del sistema. Normalmente durante su tiempo de vida, los sistemas tienen muchas versiones, pueden durar incluso más de 10 años. Existen muchas herramientas para la gestión de la configuración, así como la ingeniería inversa que ayuda en el mantenimiento de los sistemas no estructurados, permitiendo estructurar los componentes de éstos facilitando así su mantenimiento.
- Es muy importante contar con programadores que entiendan perfectamente la filosofía de programación del paradigma elegido, ya que no será lo mismo para un programador acostumbrado a la orientación a objetos, el enfrentarlo

ahora a una aplicación de inteligencia artificial con un lenguaje de programación totalmente nuevo para él.

PARADIGMA DE PROGRAMACIÓN IMPERATIVA

La programación imperativa es un paradigma de programación en el cuál se describe la programación en términos del estado del programa y sentencias que cambian dicho estado, en ocasiones también es conocida como programación procedural o procedimental. Los programas escritos en un lenguaje imperativo, están compuestos de instrucciones que le indican a la computadora cómo realizar una tarea.

El paradigma imperativo debe su nombre al papel dominante que desempeñan las sentencias imperativas. Su esencia es el cálculo iterativo, paso a paso, de valores de nivel inferior y su asignación a posiciones de memoria.

Este paradigma suele ser conocido también como algorítmico, esto debido a que el significado de algoritmo es análogo al de receta, método, técnica, procedimiento o rutina, y se define como:

Conjunto finito de reglas diseñadas para crear una secuencia de operaciones para resolver un tipo específico de problemas.

De esta forma, para N. Wirth, un programa viene definido por la ecuación:

```
Algoritmos + Estructura de datos =  Programa
```

No obstante, aunque el concepto de algoritmo encaja en otros paradigmas, es privativo del tipo de programación procedimental en el que la característica fundamental es la secuencia computacional.

Si se analizan las características fundamentales de este tipo de paradigma se detectan las siguientes:

- Concepto de celda de memoria (variable) para almacenar valores. El componente principal de la arquitectura es la memoria, compuesto por un gran número de celdas donde se almacenan los datos. Las celdas tienen nombre (concepto de variable) que las referencían y sobre los que se producen efectos de lado y definiciones de alias.
- Operaciones de asignación. Estrechamente ligado a la arquitectura de la memoria, se encuentra la idea de que cada valor calculado debe ser

almacenado, es decir, asignado a una celda. Esta es la razón de la importancia de la sentencia de asignación en el programa imperativo. Las nociones de celda de memoria se extienden a todos los lenguajes de programación e imponen en los programadores un estilo de pensamiento basado en la arquitectura Von Neumann.

- Repetición. Un programa imperativo, normalmente realiza su tarea ejecutando repetidamente una secuencia de pasos elementales, ya que en este modelo computacional la única forma de ejecutar algo complejo es repitiendo una secuencia de instrucciones.

Derivado de esto, un lenguaje de programación imperativo es un lenguaje de recorrido completo que además soporta un cierto número de características fundamentales que han nacido con la evolución del paradigma de programación imperativa desde los años cuarenta. Estas características incluyen:

- Tipos de datos para números reales, caracteres, cadenas, booleanos y sus operadores.
- Estructuras de control, bucles *for* y *while*, instrucciones *case* (*switch*).
- Asignación de elementos y *arrays*.
- Asignación de elementos y estructuras de grabación.
- Comandos de entrada y salida.
- Punteros.
- Procedimientos y funciones.

El paradigma se caracteriza por un modelo abstracto de ordenador que consiste en un gran almacenamiento de memoria.

El ordenador almacena una representación codificada de un cálculo y ejecuta una secuencia de comandos que modifican el contenido de ese almacenamiento. Este paradigma viene bien representado por la arquitectura Von Neumann, ya que utiliza este modelo de máquina para conceptuar las soluciones:

> *"Existe un programa en memoria que se va ejecutando secuencialmente y que toma unos datos de la memoria, efectúa unos cálculos y actualiza la memoria".*

La programación en el paradigma imperativo consiste en determinar qué datos son requeridos para el cálculo, asociar a esos datos unas direcciones de memoria, y

efectuar paso a paso una secuencia de transformaciones en los datos almacenados, de forma que el estado final represente la solución a un problema específico.

Un ejemplo claro de la programación imperativa son los sistemas bajo los cuáles esta implementado el *hardware*[7]. La mayoría del equipo del ordenador está diseñado para reconocer únicamente lenguaje máquina que está escrito en forma imperativa de bajo nivel esto se debe a que los computadores implementan el paradigma de la Máquina de Turing.

Los lenguajes imperativos de alto nivel utilizan variables y sentencias de mayor nivel de complejidad, pero siguen la misma filosofía que los de bajo nivel en donde el programa está definido por los contenidos en memoria y las sentencias son instrucciones en el lenguaje máquina nativo del ordenador (por ejemplo el lenguaje ensamblador).

En su forma pura este paradigma sólo soporta sentencias simples que modifican la memoria y efectúan bifurcaciones condicionales e incondicionales. Incluso cuando se añade una forma simple de abstracción procedimental, el modelo permanece básicamente sin cambiar. Los parámetros de los procedimientos son “alias” de las zonas de memoria, por lo que pueden alterar su valor, y no retorna ningún tipo de cálculo. La memoria también se puede actualizar directamente mediante referencias globales.

Los primeros lenguajes imperativos fueron los lenguajes máquina que surgieron en cada modelo de las primeras computadoras. Las instrucciones que se empleaban en estos lenguajes resultaron muy simples y facilitaron la implementación de *hardware* en los equipos, sin embargo, no resultaban muy eficientes para realizar programas que realizaran tareas más complejas.

Los lenguajes de programación imperativos están clasificados en orientados a expresiones y orientados a sentencias, esto dependiendo de cómo jueguen las expresiones o sentencias un papel más predominante en el lenguaje respectivamente. Ambos son términos relativos y no se pueden aplicar de forma absoluta. Se puede decir que C[3], FORTRÁN[9], Algol[21] y Pascal[22] son lenguajes orientados a expresiones; mientras que COBOL[10] y PL/1 están orientados a sentencias.

Las expresiones se han encontrado útiles principalmente porque son simples y jerárquicas, y pueden combinarse uniformemente para construir expresiones más complejas.

[22] Remítase a la página 65 para la descripción del lenguaje de programación Pascal

Dentro del paradigma de la programación imperativa, nos encontramos que existen distintas formas de escribir el código de un programa, así pues, entre estas formas de escribirlo, nos podemos encontrar con la programación estructurada y la programación modular que se definen a continuación.

Programación Estructurada

La programación estructurada es un criterio de programación basado en el Teorema de la Estructura de Bohn y Jacopini que dice que todo programa propio, es decir, con un solo punto de entrada y de salida, puede ser escrito utilizando únicamente tres estructuras de control: secuencial de instrucciones, instrucción condicional e iteración o bucle de instrucciones; siendo innecesario y no permitiéndose el uso de la instrucción o instrucciones de transferencia incondicional.

Cualquier programa largo y complejo siempre se puede desarrollar mediante el anidamiento apropiado de estos tres tipos de estructuras.

Estructura secuencial: una estructura de programa es secuencial si se ejecutan una tras otra de modo de secuencia como en el siguiente ejemplo:

```
auxiliar:= x
x:= y
y:= auxiliar
```

- Esta secuencia de instrucciones permuta los valores de *x* y y, con ayuda de una variable `auxiliar`, intermedia.
- 1º se guarda una copia del valor de `x` en `auxiliar`.
- 2º se guarda el valor de `y` en `x`, se pierde el valor anterior de `x` pero no importa porque tenemos una copia en `auxiliar`.
- 3º se guarda en `y` el valor de `auxiliar`, que es el valor inicial de `x`.
- El resultado es el intercambio de los valores de `x` e `y`, en tres operaciones secuenciales.

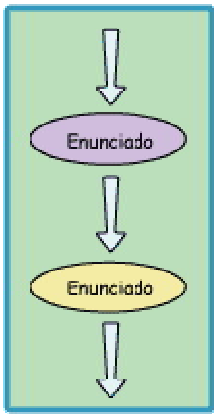

Ilustración 1-12. Estructura secuencial de la programación estructurada

Estructura selectiva: la estructura selectiva permite la realización de una instrucción u otra según un criterio, solo una de estas instrucciones se ejecutara.

Ejemplo:

```
si a > b entonces
Escribir a es mayor que b
si_no
Escribir a no es mayor que b
fsi
```

- Esta instrucción selectiva puede presentar dos mensajes, uno `a es mayor que b`, y el otro `a no es mayor que b`, solo uno de ellos será presentado, según el resultado de la comparación de a y b, si el resultado de *a > b* es cierto, se presenta el primer mensaje, si es falso el segundo mensaje, las palabras `si, entonces, si_no, fsi`; son propias de la instrucción (palabra reservadas) que tienen un significado en el lenguaje, sirven de separadores, y el usuario no debe utilizarlas salvó para este fin.
- *`si`* señala el comienzo de la instrucción condicional, y se espera que después este la condición de control de la instrucción.
- `entonces` señala el fin de la condición, y después estará la instrucción a realizar si la condición es cierta.
- `si_no` separa la instrucción que se ejecutara si la condición es cierta de la que se ejecutara si es falsa.
- *`fsi`* indica que la instrucción condicional finaliza y el programa seguirá su curso.

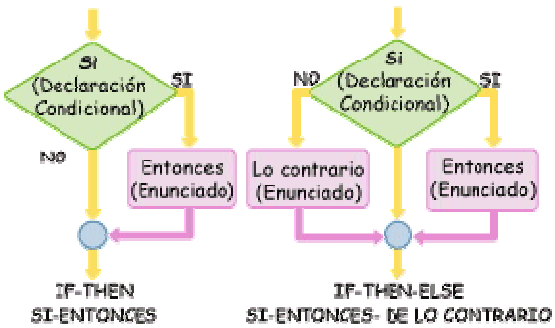

Ilustración 1-13. Estructura selectiva de la programación estructurada

Estructura iterativa: un bucle iterativo o iteración de una secuencia de instrucciones, hace que se repetirán mientras se cumpla una condición, en un principio el número de iteraciones no tiene por qué estar determinado.

Ejemplo:

```
a:= 0
b:= 7
mientras b > a hacer
Escribir a
a:= a + 1
fmientras
```

Esta instrucción tiene tres palabras reservadas `mientras, hacer` y `fmientras`.

- `mientras`: señala el comienzo del bucle y después de esta palabra se espera la condición de repetición, si la condición es cierta, se pasa al cuerpo del bucle, si no, al final de la instrucción mientras.
- `hacer`: señala el final de la condición, lo que este después será el cuerpo del bucle.
- `fmientras`: señala el final del cuerpo del bucle y de la instrucción `mientras`.

El bucle `mientras` se repetirá mientras la condición sea cierta, la condición se comprueba al principio por lo que puede ser que el cuerpo del bucle no se ejecute nunca, esto cuando la condición resulta ser falsa desde un principio, o que se repita tantas veces como sea necesario mientras la condición resulte ser afirmada.

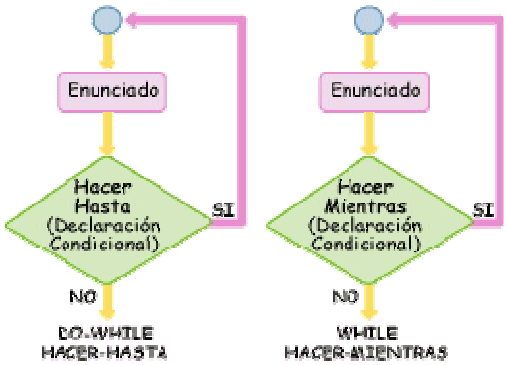

Ilustración 1-14. Estructura iterativa de la programación estructurada

Con la programación estructurada se obtienen las siguientes ventajas:

- Un programa estructurado puede ser leído en forma secuencial, de arriba hacia abajo, sin tener la necesidad de saltar de un lado a otro dentro del código. La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre sí, haciendo más fácil de entender lo que hace cada una de ellas.
- Se reduce el tiempo de pruebas ya que los errores son más fáciles de detectar por la facilidad de comprensión de la lógica que sigue el programa.
- Programación sencilla y rápida.

La principal desventaja de la programación estructurada es que sólo se obtiene un único bloque de programa, que si se trata de un programa lo suficientemente grande, puede ocasionar dificultad para su manejo, lo cuál se puede resolver empleando la programación modular, en donde se pueden obtener módulos programados independientemente uno de otro y compilados por separado, pero que funcionan como un solo proyecto y en conjunto nos llevan a la solución del mismo problema.

Programación Modular

Dentro de la historia de la programación de *software*[1], con los años, en el diseño de programas se le dio un mayor énfasis al diseño de procedimientos que a la organización de la información. Entre otras cosas esto reflejó un aumento en el tamaño y complejidad de los programas. La programación modular surge como un remedio a

esta situación ya que agrupa un conjunto de procedimientos afines junto que los datos que manipulan para poder ser tratados como un todo.

Esta técnica de programación consiste en dividir un programa en partes bien diferenciadas lógicamente, llamadas módulos, que pueden ser analizadas y programadas por separado.

Un módulo se puede definir como un conjunto formado por una o varias instrucciones lógicamente enlazadas.

A cada módulo se le asigna un nombre, que elige el programador, para poder identificarlo. Cuando en un punto de un programa se hace referencia a un módulo, el programa le cede el control para que se ejecuten todas sus instrucciones. Una vez que se haya finalizado la ejecución del módulo, el control se devuelve al punto del programa desde donde se llamó al módulo y se continúa con la ejecución de la instrucción siguiente a la que realizó la llamada. El orden de ejecución está representado gráficamente en la siguiente figura.

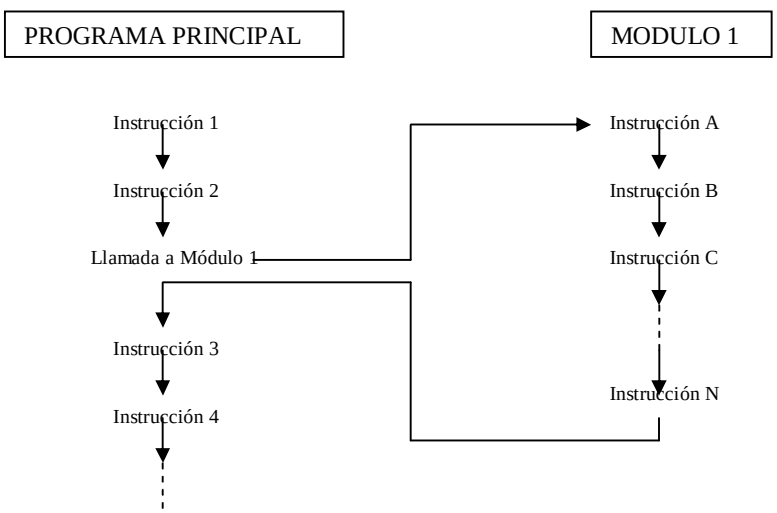

Ilustración 1-15. Programación modular

Si un módulo es lo suficientemente grande o complicado puede subdividirse en otros módulos, y éstos, a su vez, en otros, un programa completo se podría representar como la siguiente figura. Tal como se observa, siempre debe existir un módulo raíz o principal, que es el encargado de controlar y relacionar a todos los demás.

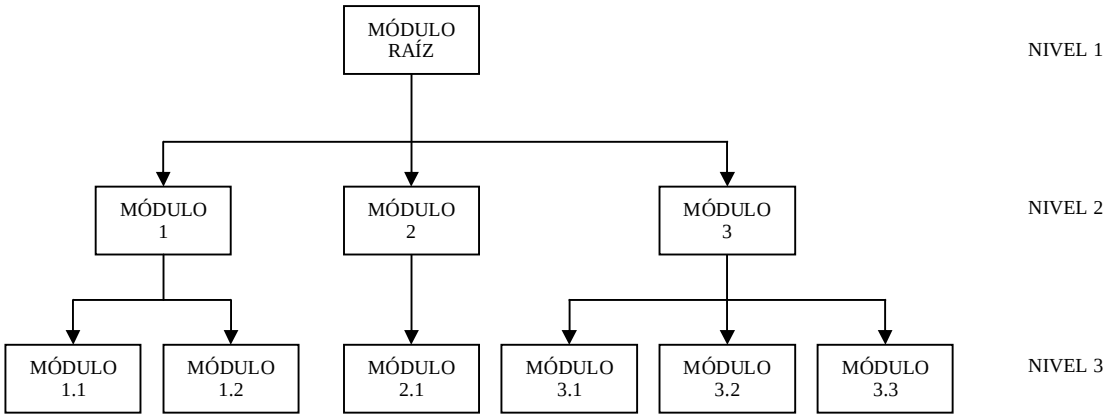

Ilustración 1-16. División de un sistema en distintos módulos

Si los módulos que componen un programa son parte integrante del mismo, se les llama rutinas, subrutinas, procedimientos o funciones. Si por el contrario son programas independientes, reclamados desde otros programas, se les denomina subprogramas.

No hay una norma fija para dividir un programa en módulos. Se hace a criterio del programador. Sin embargo, se deben seguir unas normas más o menos generalizadas:

- Cada módulo solo puede tener un punto de entrada y otro de salida.
- El módulo principal debe ser conciso, mostrando claramente los módulos que lo componen y su relación. Es decir, debe indicar la solución completa del problema.
- Los módulos deben tener la máxima independencia entre ellos.
- Un módulo debe representar por sí mismo una estructura lógica coherente y resolver una parte definida del problema.

La programación modular es, pues, una técnica que se basa en el desarrollo de programas de lo general a lo particular, o dicho de otra manera, del conjunto al elemento, (diseño *Top-Down*). Se comienza considerando qué funciones debe realizar el programa desde un punto de vista muy general, dándolas como resueltas y dejando su diseño (el cómo realizará esas funciones) para un paso posterior. De esta manera se avanza hasta llegar al máximo nivel de detalle.

Es decir, si dentro de un módulo aparece una función cuyo desarrollo completo queremos posponer, esto lo representamos indicando en ese punto el nombre que tendrá el módulo que lo desarrollará, ocupándonos posteriormente de dicho desarrollo.

La programación modular aporta una serie de ventajas con respecto a la programación convencional:

- Los programas son más sencillos de escribir y depurar, pues se pueden hacer pruebas parciales con cada uno de sus módulos.
- La corrección o modificación de un módulo se hace más cómoda y, en general, no tiene por qué afectar al resto de los módulos.
- Un programa se puede ampliar fácilmente con sólo diseñar los nuevos módulos necesarios.
- Un mismo módulo escrito una sola vez puede ser referenciado desde varios puntos del programa, evitando la repetición de instrucciones ya escritas.

La programación modular y la programación estructurada no son criterios contrarios, sino complementarios. El primero tiende a dividir un programa en partes más pequeñas, llamadas módulos, y el segundo se encarga de desarrollar estructuralmente cada una de estas partes.

PARADIGMA DE PROGRAMACIÓN FUNCIONAL

La programación funcional es un paradigma de programación declarativa basado en la utilización de funciones matemáticas.

La característica esencial de la programación funcional es que los cálculos se ven como una función matemática que hace corresponder entradas y salidas en donde el resultado de un cálculo es la entrada del siguiente y así sucesivamente hasta que una composición produce el resultado final. A diferencia de la programación imperativa, no hay notaciones implícitas de estado y, por tanto, no hay necesidad de una instrucción de asignación. No existe el concepto de celda de memoria que es asignada o modificada. Más bien, existen valores intermedios que son el resultado de cálculos anteriores. Carece también de construcciones estructuradas como la secuencia o la iteración.

Todo esto provoca que en la práctica todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas que se hacen llamar así mismas tantas veces como sea necesario ya que no hay manera de incrementar o disminuir el valor de una variable en el estado. Si embargo, como aspecto práctico para los programadores, la mayoría de los lenguajes funcionales soportan las nociones de variable, asignación y bucle.

Como cualquier programa imperativo determinista puede verse como una función matemática que hace corresponder entradas y salidas, se ha desarrollado una teoría de corrección de programas utilizando como base este punto de vista de los programas como funciones.

Debido a su pureza relativa, algunos ven en la programación funcional un paradigma más fiable que la programación imperativa para el diseño de *software*[1].

Sus orígenes provienen del Cálculo Lambda (o λ-cálculo), teoría matemática realizada por Alonzo Church como apoyo a sus estudios de computabilidad. Así, el primer lenguaje de programación diseñado específicamente para dar soporte a la programación a nivel funcional fue el FP.

La programación a nivel funcional en el estilo de FP tiene una fuerte relación con la lógica combinatoria de Haskell Curry, con los lenguajes de combinadores, antecedentes de Miranda y Haskell, así como con las categorías cartesianas cerradas, teoría que dio origen al lenguaje CAML, antecesor del lenguaje Ocaml.

El objetivo de la programación funcional es conseguir lenguajes que sean manejables matemáticamente y en los que no sea necesario bajarse al nivel máquina para describir los procesos que está realizando, evitando con esto el concepto de estado del cómputo. La secuencia de las actividades llevadas a cabo por el programa estará regida únicamente por la reescritura de definiciones más amplias a otra y que son cada vez más concretas y definidas.

Los programas diseñados a nivel funcional están escritos como combinación de otros programas con la ayuda de las operaciones de construcción de programas o funcionales. Bajo este enfoque, los programas con los funcionales como operadores forman un espacio matemático.

$$\sum_{i}^{k} i = i + \underbrace{(i+1) + (i+2) + \cdots + k}_{\sum_{i+1}^{k} i} = i + \sum_{i+1}^{k} i$$

Ilustración 1-17. La programación funcional se basa en un entorno matemático

El paradigma de la programación funcional nos lleva a la escritura de programas que están constituidos únicamente por definiciones de funciones, cabe notar que estas funciones no corresponden a las conocidas en la programación imperativa en donde las funciones son utilizadas como subprogramas y que son llamadas cada vez que sean necesarias. En este caso se trata de funciones puramente matemáticas en las que se revisan ciertas propiedades como que el significado de una expresión depende únicamente de sus subexpresiones, lo cuál es denominado transparencia referencial, y por lo tanto, no hay una secuencia de efectos laterales causados por la misma.

La programación funcional incorpora el concepto de función como objeto de primera clase, lo que significa que las funciones se pueden tratar como datos porque se pueden pasar como parámetros, ser calculadas y devueltas como valores normales, y después de esto, ser mezcladas con otras formas de datos.

En la actualidad se puede decir que existen dos grandes categorías de lenguajes funcionales: así por un lado se tiene a los lenguajes funcionales puros que conservan una mayor potencia expresiva y que por lo tanto conservan su transparencia referencial; por el otro lado, se encuentran los lenguajes funcionales híbridos que son menos estrictos que los puros ya que admiten conceptos tomados de la programación imperativa, como son las secuencias de instrucciones o la asignación de variables, perdiendo a su vez, la transparencia referencial que caracteriza a los lenguajes puros.

Entre los lenguajes funcionales puros, cabe destacar a Haskell[14] y Miranda. Los lenguajes funcionales híbridos más conocidos son Lisp[17], Scheme[18], Ocaml y Standar ML (estos dos últimos, descendientes del lenguaje ML).

Cálculo lambda

El cálculo lambda es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión. Fue introducido por Alonzo Church y Stephen Kleene en la década de 1930. Church usó el cálculo lambda en 1936

para resolver el *Entscheidungsproblem*[23]. Puede ser usado para definir de manera limpia y precisa qué es una función computable.

Church resolvió negativamente el *Entscheidungsproblem*: probó que no había algoritmo que pudiese ser considerado como una solución al *Entscheidungsproblem*.

El cálculo lambda ha influenciado enormemente el diseño de lenguajes de programación funcionales, especialmente Lisp[17].

Se puede considerar al cálculo lambda como el más pequeño lenguaje universal de programación. Consiste de una regla de transformación simple (substitución de variables) y un esquema simple para definir funciones.

El cálculo lambda es universal porque cualquier función computable puede ser expresada y evaluada a través de él. Por lo tanto, es equivalente a las máquinas de Turing. Sin embargo, el cálculo lambda no hace énfasis en el uso de reglas de transformación y no considera las máquinas reales que pueden implementarlo. Se trata de una propuesta más cercana al *software*[1] que el *hardware*[7].

Ilustración 1-18. El cálculo lambda influencia el desarrollo de lenguajes de programación funcionales

PARADIGMA DE PROGRAMACIÓN LÓGICA

La programación lógica se basa en la lógica simbólica y consiste en la aplicación del conocimiento sobre la lógica para el diseño de lenguajes de programación, aunque hay que tener presente que esta lógica no es la misma a la que se refiere la lógica computacional.

Esta programación se basa en un subconjunto del cálculo de predicados, incluyendo instrucciones escritas en formas conocidas como cláusulas de Horn[24]. Este paradigma puede deducir nuevos hechos a partir de otros hechos conocidos. Un

[23] En castellano, problema de decisión, fue el reto en lógica simbólica de encontrar un algoritmo general que decidiera si una fórmula del cálculo de primer orden es un teorema.
[24] Regla de inferencia lógica con una serie de premisas (cero, una o más), y un único consecuente.

sistema de cláusulas de Horn permite un método particularmente mecánico de demostración llamado resolución.

La programación lógica está basada en otros dos paradigmas de programación: la programación declarativa y la programación funcional. La programación declarativa esta basada en torno al concepto de predicado, o relación entre elementos, la programación funcional gira en torno al concepto de la función matemática.

La lógica matemática resulta una manera muy práctica para el humano de expresar formalmente problemas complejos y de resolverlos mediante la aplicación de reglas, hipótesis y teoremas. Es por este motivo que la programación lógica ha sido atractiva para diversos campos donde la programación imperativa usada tradicionalmente hoy en día no se puede aplicar para resolver dichos problemas.

En un lenguaje de programación lógico, un programa está formado por un conjunto de enunciados que describen lo que es verdad con respecto a un enunciado deseado, en oposición a dar una secuencia particular de enunciados que deben ser ejecutados en un orden fijo para producir el resultado. Están construidos únicamente por expresiones lógicas, es decir, que son ciertas o falsas, en oposición a una expresión interrogativa (una pregunta) o expresiones imperativas (una orden). Un lenguaje de programación lógico puro no tiene necesidad de abstracciones de control como ciclos o selección. El control es suministrado por el sistema subyacente. Todo lo que se necesita en un programa lógico es el enunciado de las propiedades del cómputo. Por esta razón a la programación lógica algunas veces se le conoce como programación declarativa, dado que las propiedades se declaran, pero no se especifica una secuencia de ejecución.

La programación lógica encuentra su campo de aplicación más amplio en el desarrollo de sistemas de inteligencia artificial y algunos otros como lo son:

- Los sistemas expertos, en donde un sistema informático trata de imitar las recomendaciones de algún humano experto sobre algún dominio de conocimiento.
- Demostración automática de teoremas, en los cuáles, el programa genera nuevos teoremas a partir de una teoría existente.
- Reconocimiento del lenguaje natural, en donde los programas son capaces de comprender lo que un humano esta tratando de expresar a través de la voz.

Ilustración 1-19. El reconocimiento de voz se impulsa por la programación lógica

Los programas que están escritos en este tipo de paradigma, están basados en la teoría lógica de primer orden, aunque también pueden incorporar comportamientos de orden superior. Tal es el caso de algunos programas que tienen ciertas características de la programación funcional que tiene su estructura basada en el cálculo lambda, que hasta el momento, se trata de la única teoría lógica de orden superior que puede ser computable.

La programación lógica puede transportar al mundo informático algunos hechos del mundo real, por ejemplo tenemos el siguiente caso:

```
Las aves vuelan.
Los pingüinos no vuelan.
“Kiko” es un ave.
“Laynus” es un perro.
“Motas” es un ave.
“Pichi” es un pingüino.
```

Aparte de los enunciados anteriores que describen situaciones del mundo real, la programación lógica también permite llevar a las computadoras ciertas restricciones, como son:-

```
Un animal vuela si es un ave y no es pingüino.
```

Ante estas situaciones, se pueden establecer ciertas hipótesis o preguntas como son:

```
¿Kiko vuela?
¿Qué mascotas vuelan?
```

Con la lógica de primer orden que se ha llevado a la computadora por medio de algún tipo de programación lógica, el ordenador será capaz de revisar en toda la información que tiene almacenada las restricciones que tiene entre estos elementos y contestar a las incógnitas, resolviendo que:

```
Como Kiko es un ave, y las aves vuelan, Kiko puede
volar.
Como Kiko y Motas son aves, las aves vuelan,y ninguno
de los dos son pingüinos, Kiko y Motas son las mascotas que
pueden volar.
```

Con esto nos damos cuenta que la programación lógica no sólo responde si una determinada hipótesis resulta ser cierta o falsa, también puede determinar qué valores de la incógnita hacen verdadera o falsa a la hipótesis.

Otro ejemplo de la aplicación de los programas lógicos puede ser el control de un sistema de semáforos de una ciudad. Los estados del mundo real que serán llevados a la computadora a través del programa consisten únicamente en decirle cuándo está cada uno de los semáforos en color verde (siga), amarillo (disminuir velocidad) o rojo (alto total). El programa constituiría en unas pocas reglas si es que sólo se quisiera realizar un programa de tipo secuencial, en donde los semáforos cambiarían de color cada determinado tiempo si importar otras situaciones como la cantidad del parque vehicular que hay en determinada calle o avenida, la hora del día, etc. En estos casos valdría la pena que se realizaran reglas más complejas con ayuda de sensores de presencia, de velocidad, entre otras mediciones que podrían alimentar al sistema. Así por ejemplo, un semáforo podría permanecer en verde si es que hay mucho parque vehicular pasando por la calle y en la calle perpendicular no existe tanto, o cambiar a amarillo y posteriormente a rojo si se da en caso contrario, afectando a otros semáforos de calles adyacentes a la misma.

Éste sería un claro ejemplo de inteligencia artificial en donde la computadora es capaz de pensar por sí misma y realizar ciertas acciones de acuerdo a la información que contiene y a la que procesa en cada momento.

Una variante de la programación lógica es la programación con restricciones que se describe a continuación.

Programación con restricciones

Se trata de un estilo de programación en informática utilizado actualmente como tecnología de *software*[1] para la descripción y resolución de problemas combinatorios particularmente difíciles, especialmente en las áreas de planificación y calendarización.

Es uno de los estilos que ha llamado mucho la atención desde 1990, y ha sido identificado por la ACM (Asociación de Maquinaria Computacional) como una dirección estratégica en la investigación en computación.

En este estilo de programación se especifica un conjunto de restricciones, las cuáles deben satisfacer una solución, en vez de especificar los pasos para obtener dicha solución.

La programación con restricciones se relaciona mucho con la programación lógica. De hecho, cualquier programa lógico puede ser traducido en programa basado en restricciones y viceversa. Muchas veces los programas lógicos son traducidos a una programación con restricciones debido a que la resolución de un programa basado en restricciones se puede desempeñar mejor que su contraparte.

La diferencia entre ambos radica principalmente en sus estilos y enfoques de modelado del mundo. Para ciertos programas es más natural (y por ende más simple) escribirlos como programas lógicos, mientras que en otros es más natural escribirlos como programas basados en restricciones.

El enfoque de este estilo de programación se basa principalmente en buscar un estado en el cual una gran cantidad de restricciones sean satisfechas simultáneamente. Los lenguajes de este tipo son típicamente incrustados en un lenguaje host. El primer lenguaje host usado fue Prolog[11], es por esta razón que este campo fue llamado inicialmente como programación lógica con restricciones.

PARADIGMA DE PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos es una de las más exitosas y persuasivas áreas dentro de la informática. Desde sus orígenes en la década de 1960, ha invadido tanto las áreas de la investigación científica como del desarrollo tecnológico.

La programación orientada a objetos es una técnica que ha mejorado la estructura de los datos, así como su manipulación, llegando a ser considerado como un

gran paso en la ingeniería del *software*[1], ya que presenta un modelo de objetos que parece encajar de manera adecuada con los problemas reales. Por tal motivo ha sido aplicado y fusionado e incorporado en otros tipos de paradigmas de programación como lo es el imperativo, el secuencial y funcional, dejando de ser paradigmas de programación puros pero convirtiéndose en opciones potentes de programación.

Los lenguajes que utilizan ese paradigma han tenido mucho éxito, pues les permiten a los programadores escribir código reutilizable y con la opción de ampliarlo que opera imitando al mundo real, permitiendo por lo tanto que los programadores utilicen su intuición natural con respecto al mundo para comprender el comportamiento de un programa y construir código apropiado. En cierto sentido, la programación orientada a objetos es una ampliación de la programación imperativa, ya que se basa principalmente en la misma ejecución secuencial con un conjunto cambiante de localizaciones de memoria. La diferencia consiste en que los programas resultantes están formados de un gran número de piezas muy pequeñas cuyas interacciones están cuidadosamente controladas y al mismo tiempo son fácilmente cambiadas.

Aunque hay muchas interpretaciones para la programación orientada a objetos, una primera idea es el diseñar un *software* de forma que los distintos tipos de datos que use estén unidos a los tipos de operaciones definidos para los mismos. La cuestión es saber descomponer de la mejor manera el dominio del problema que se está tratando de resolver, encapsulando cada aspecto en lo que se dio a conocer como objeto y haciéndolos interactuar entre sí, dotándolos de una cantidad de propiedades, definiendo los programas en “clases de objetos”. Así pues, los programas son expresados como un conjunto de objetos que colaboran entre sí para realizar una tarea. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

El paradigma nos dice que un objeto consta de:

1.- Métodos o funciones.

2.- Variables miembro.

Ilustración 1-20. Clase de objetos con características similares

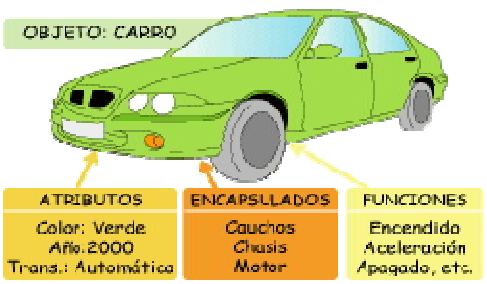

Ilustración 1-21. Objeto con identidad propia que lo identifica sobre los demás objetos de la misma clase

Así los datos (variables miembro) y el código (métodos o funciones), se combinan en entidades llamadas objetos.

Entre los conceptos básicos de la programación orientada a objetos se encuentran:

- Clase: es una plantilla que define la estructura de un conjunto de objetos, que al ser creados, se llamarán a las instancias de la clase. Esta estructura está definida por la definición de los atributos y la implementación de las operaciones (métodos).
- Objeto: es la instanciación de una clase, es decir, una ocurrencia de ésta que tiene los atributos definidos por una clase y sobre la que se puede ejecutar las operaciones definidas en ella.
- Identidad: característica de cada objeto que lo diferencia de los demás, incluyendo de aquellos que pudieran pertenecer a la misma clase y tener los mismos valores en sus atributos.
- Método: algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un mensaje. Desde el punto de vista del comportamiento, es lo que un objeto puede hacer. Un método puede provocar un cambio en las propiedades del objeto y/o la generación de un evento con un nuevo mensaje para otro objeto del mismo sistema.
- Mensaje: es una llamada enviada a un objeto que le ordena que ejecute alguno de sus métodos con ciertos parámetros asociados al evento que lo generó.

Un objeto puede verse como un paquete que contiene el comportamiento (todos los métodos y funciones definidas para su comportamiento) y el estado (la identidad propia de cada objeto que es un conjunto de propiedades o atributos y que constituyen los datos del mismo). El principio es separar aquellas características de los objetos que cambian entre cada uno de ellos, de aquellas características que permanecen inalterables para todos los objetos en conjunto. Ésta característica lleva a tratar a los objetos como unidades indivisibles, en las que no se separan (ni deben de separarse) información (los datos contenidos en la identidad) y el procesamiento de la misma (los métodos utilizados para ello).

Los objetos contienen una serie de variables que los definen y requieren de unos métodos para tratar dichas variables, lo que hace que estos términos estén

íntimamente entrelazados. Al hacer un programa basado en el paradigma de la programación orientada a objetos, el programador debe de pensar de igual manera en ambos términos, sin crear una separación entre ellos o dar mayor énfasis a alguno en especial, ya que de hacerlo, estaría cayendo en el error de crear clases definidas para el almacenamiento de la información y por otro lado, crear clases con métodos que manejan dicha información, logrando con esto una programación estructurada con simples características de programación orientada a objetos y sin llegar a explotar las ventajas que tiene ésta sobre la estructurada.

Las características más importantes soportadas por la programación orientada a objetos son:

- Abstracción: es el proceso en el que el desarrollador lleva la realidad a un modelo computacional que le permite desarrollar un *software*[1] que satisfaga las necesidades del sistema tratado. En la POO esto es identificar los objetos y las clases involucradas en el sistema. Cada objeto en el sistema sirve como un agente abstracto que puede realizar trabajo, informar y cambiar su estado, y comunicarse con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

  La Clasificación por Categorización Clásica (agrupar elementos con propiedades similares), Agrupamiento Conceptual (agrupar entidades que compartan significado conceptual, es decir, para lo que sirven) y Teoría de Prototipos pueden ayudar mucho. Lo recomendable es no realizar una sola abstracción sino varias y en cada una de ellas plasmar una parte del problema.
- Encapsulamiento: también conocido como ocultación de la información porque cada objeto está aislado del exterior, es un módulo natural y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones no deseadas. Algunos lenguajes son más

accesibles en este punto permitiendo un acceso directo a los datos internos de un objeto de manera controlada y limitando el grado de abstracción.

- Polimorfismo: comportamientos diferentes asociados a objetos distintos que pertenecen a una misma clase, pudiendo compartir incluso el mismo nombre, pero que al llamarlos se utilizarán los métodos correspondientes de acuerdo al objeto que se esté usando. Dicho de otro modo, las colecciones de objetos pueden contener objetos con distintas características, y en donde la invocación a un método de la clase producirá el comportamiento adecuado al tipo de objeto que se está llamando. Cuando esto ocurre en tiempo de ejecución, ésta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en tiempo de compilación) de polimorfismo, tales como las plantillas y la sobrecarga de operadores en C++[12].
- Herencia: el principio de la herencia esta basado en la observación de que la abstracción de los datos frecuentemente tienen muchas cosas en común. Las clases no están aisladas, sino que se relacionan entre sí formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Éstos pueden compartir (y extender) su comportamiento sin tener que reimplementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y éstas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto pertenece a más de una clase se llama herencia múltiple. Así pues, un objeto puede heredar las características de una o más clases, teniendo la misma funcionalidad que los otros, y con posibilidad de tener algunas extensiones y modificaciones.

  El concepto de herencia se introduce para reducir el problema de código duplicado y para hacer relaciones entre distintos objetos o abstracciones de la realidad.

Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable y fácil de manejar para el

diseño de un sistema. El objetivo es hacer que los grandes proyectos sean fáciles de gestionar y manejar, obteniendo con esto, una mejora en su calidad y reduciendo el número de proyectos fallidos.

Otra de las ventajas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización de código entre distintos proyectos, la cuál es una de las metas fundamentales en la Ingeniería del Software. Sin embargo, la reutilización de *software*[1] no ha sido aplicada como se esperaría, y esto se debe principalmente a dos factores: el diseño de objetos realmente genéricos no ha sido comprendido totalmente por los desarrolladores de *software*; la segunda causa es que falta una metodología para la amplia comunicación de oportunidades de reutilización de código ya elaborado. Algunas comunidades de "código abierto" (*open source*) quieren ayudar en este problema dando medios a los desarrolladores para difundir la información sobre el uso y versatilidad de objetos reutilizables y librerías de objetos.

El primer lenguaje orientado a objetos fue Simula 67, desarrollado en 1967 y que fue un descendiente de Algol 60[21]. Fue hasta a mediados de los años ochenta que tomó posición como la metodología de programación dominante y se convirtió popular dentro de la industria, en gran parte, debido a la influencia de C++[12]. Otro gran paso fue el desarrollo de Samalltalk[15], creado en 1980 como parte de una investigación realizada en los setentas. Tanto C++ como Smalltalk fueron directamente influenciados por Simula. Su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para las cuáles, el paradigma está particularmente bien adaptado. En este caso, también se habla de la programación orientada a eventos que es una ventaja de este paradigma y que se define más adelante en esta misma sección.

Los lenguajes de programación más populares en la actualidad, Java[4] y C++ son orientados a objetos. Las ventajas de la programación orientada a objetos sobre la programación imperativa incluyen mejoras entre las que están el uso de los patrones de diseño, diseño por contrato y lenguajes de modelado (por ejemplo UML y Desing Patterns o diseño de patrones).

Programación Orientada a Eventos

Es un tipo de programación en la que tanto la estructura como la ejecución de los programas están controlados por un bucle continuo que responde a los sucesos que ocurren en el sistema o que ellos mismos provoquen en un orden no predecible.

A diferencia de la programación secuencial en donde es el programador el que define el flujo que seguirá el programa una vez que esté en ejecución, y visualiza un proceso que siempre terminará cuando se completen sus pasos, en la programación orientada por eventos será el propio usuario (o el mismo sistema si es que se controla a sí mismo) el que dirija el flujo del programa, ya que no se predice la secuencia de control que se va a producir.

En la programación dirigida por eventos, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y demás código inicial y a continuación el programa quedará bloqueado hasta que se produzca algún evento, están escritos para reaccionar razonablemente a cualquier secuencia de eventos que se produzcan. Cuando alguno de estos eventos tenga lugar, el programa pasará a ejecutar el código del correspondiente manejador de evento. Los eventos pueden generarse debido a la recepción de señales desde elementos de *hardware*[7] como el ratón o el teclado, o pueden producirse al realizar alguna operación sobre un elemento de la propia aplicación (como abrir un conjunto de datos). Además, la ejecución de un programa guiado por eventos no termina normalmente; estos programas están diseñados para ejecutarse durante un periodo de tiempo arbitrario.

Por ejemplo, si el evento consiste en que el usuario ha hecho clic en el botón de *play* de un reproductor de películas, se ejecutará el código del manejador de evento, que será el que haga que la película se muestre por pantalla.

Así tenemos por ejemplo que en los lenguajes de programación como Lexico[25] y Visual Basic[13], a cada elemento de los programas escritos en ellos (objetos, controles, etc.), se les asigna una serie de eventos que generará dicho elemento, como la pulsación de un botón o el redibujado del control.

Así mismo los eventos también forman parte de un conjunto de objetos ya definidos que no tienen que codificarse para cada evento que se presente en el programa, así están por ejemplo los objetos denominados clic, dobleclic, etc.

[25] Remítase a la página 76 para la descripción del lenguaje de programación Lexico

CAPÍTULO 2

"CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN DE ACUERDO AL PARADIGMA QUE UTILIZAN"

LOS LENGUAJES DE PROGRAMACIÓN

Los lenguajes de programación son un conjunto de sintaxis y reglas semánticas que definen los programas del computador. Es una técnica estándar de comunicación para entregarle instrucciones al ordenador. Un lenguaje le da la capacidad al programador de especificarle al computador, qué tipo de datos actúan y que acciones tomar bajo una variada gama de circunstancias, utilizando un lenguaje relativamente próximo al lenguaje humano. Para asegurarse que la computadora entienda las instrucciones, se han establecido lenguajes bien definidos para especificar y poder generar la comunicación con al computadora. Estos lenguajes tienen características similares a los lenguajes comunes que utilizan las personas para comunicarse unas con otras, pues cuentan con reglas y estructuras que deben seguirse.

Ilustración 2-1. Los lenguajes de programación son un conjunto de sintaxis y reglas semánticas que definen los programas de las computadoras

Una vez que se identifica una tarea y se conoce el algoritmo para resolverla, el programador debe codificarlo en una lista de instrucciones propias de algún lenguaje de computación. Los programas pueden ser escritos en cualquier lenguaje de la amplia gama disponibles. La elección del más adecuado, dependerá de factores como la experiencia del programador con un lenguaje, determinar cuál producirá el programa menos complicado, la flexibilidad y la compatibilidad del programa como resultado del lenguaje usado.

Un programa escrito en un lenguaje de programación necesita pasar por un proceso de compilación, interpretación o intermedio, es decir, ser traducido al lenguaje de máquina para que pueda ser ejecutado por el ordenador.

Desde 1954 hasta hoy en día se han documentado más de 2,500 lenguajes de programación. Entre 1952 y 1972, la primera época de los lenguajes de programación, se desarrollaron alrededor de 200 lenguajes, de los que una decena fueron realmente significativos y tuvieron influencia en el desarrollo de los lenguajes posteriores.

Los lenguajes de programación pueden ser clasificados de acuerdo a diversos criterios:

- Lenguajes interpretados (Interpretes) como Basic, Dbase.
- Lenguajes compilados (Compiladores) como C, C++, Clipper.
- Lenguajes interpretados con recolectores de basura (Maquina Virtual) como Smalltalk, Java, Ocaml.
- Lenguajes Scripts (Motor de ejecución) como Perl, PhP.

Los equipos de ordenador (el *hardware*[1]) han pasado por cuatro generaciones, de las que las tres primeras (ordenadores con válvulas, transistores y circuitos integrados) están muy claras, la cuarta (circuitos integrados a gran escala) es más discutible.

Algo parecido ha ocurrido con la programación de los ordenadores (el *software*[2]), que se realiza en lenguajes que suelen clasificarse en cinco generaciones, de las que las tres primeras son evidentes, mientras no todo el mundo está de acuerdo en las otras dos. Estas generaciones no coincidieron exactamente en el tiempo con las de *hardware*, pero sí de forma aproximada, y son las siguientes:

Primera generación.

Los primeros ordenadores se programaban directamente en código binario, que puede representarse mediante secuencias de ceros y unos. Estos valores pueden corresponder a que la electricidad esté encendida o apagada en la máquina, o a la presencia o ausencia de carga magnética en un medio de almacenamiento. A partir de estos dos estados se forman los esquemas de codificación (como el código ASCII) que permiten generar letras, números, signos de puntuación y caracteres especiales. Cada modelo de ordenador tiene su propio código, por esa razón se llama lenguaje de máquina siendo así el lenguaje más básico.

Una computadora sólo es capaz de comprender su lenguaje máquina original, el conjunto de instrucciones para realizar sus operaciones elementales.

[1] Término inglés que hace referencia a cualquier componente físico tecnológico, que trabaja o interactúa de algún modo con la computadora

[2] Término en inglés utilizado para llamar a las partes intangibles de una computadora, (programas, rutinas, datos, sistemas operativos, etc.).

Segunda generación

Los lenguajes simbólicos, así mismo propios de la máquina, simplifican la escritura de las instrucciones y las hacen más legibles. Los lenguajes de segunda generación están constituidos por nemotécnicos similares a palabras en idioma inglés. En primer lugar, se crea un archivo fuente con las instrucciones que se desea que la computadora ejecute, usando un editor, que es una especie de procesador de palabras. El código fuente es traducido al lenguaje máquina mediante programas traductores (compiladores). Los programadores rara vez escriben programas de tamaño relevante en este tipo de lenguajes, debido a que, a pesar que es mucho más fácil que utilizar código de máquina, siguen siendo altamente detallados (instrucciones muy básicas). Sólo se hace en los casos donde la velocidad es clave (como en la programación de juegos de video) y para afinar partes importantes de los programas que son escritos en lenguajes superiores.

Tercera generación

Los lenguajes de alto nivel sustituyen las instrucciones simbólicas por códigos independientes de la máquina, parecidas al lenguaje humano o al de las matemáticas. Son llamados de alto nivel porque se asemejan más al lenguaje que utilizan los humanos al comunicarse. Fueron desarrollados con la finalidad de facilitar el proceso de programación. Cuentan con comandos que se acercan más a las palabras de uso común en lugar de tener que usar combinaciones binarias del código máquina o los comandos básicos del ensamblador. Estos lenguajes hacen más fácil la lectura, escritura y comprensión de los programas, aunque de manera semejante a los lenguajes ensambladores, deben ser convertidos a lenguaje máquina, para poder ser usados en la computadora.

Cuarta generación

Se ha dado este nombre a ciertas herramientas que permiten construir aplicaciones sencillas combinando piezas prefabricadas. Hoy se piensa que estas herramientas no son, propiamente hablando, lenguajes. Algunos proponen reservar el

nombre de cuarta generación para la programación orientada a objetos. Están caracterizados por una mayor facilidad de uso comparados con los de la tercera generación, permitiendo la creación de prototipos de una aplicación rápidamente. Los prototipos permiten tener una idea del aspecto y funcionamiento de la aplicación antes que el código sea terminado. Esto implica que quienes estén involucrados en el desarrollo de la aplicación pueden aportar retroalimentación en aspectos como, estructura y diseño desde el principio del proceso. A cambio de esta capacidad para trabajar más velozmente, debe sacrificarse parte de la flexibilidad con la que se disponía con los lenguajes anteriores.

Muchos de estos lenguajes tienen capacidad para bases de datos, permitiendo crear programas que sirvan de enlace con las mismas. Los programas incluyen formas y cuadros para introducir datos, así como solicitar reportes de información de las bases de datos. En general, ahorran tiempo puesto que el código requerido para "conectar" los cuadros de diálogo y las formas se genera de forma automática.

Quinta generación

Se llama así a veces a los lenguajes de la inteligencia artificial, aunque con el fracaso del proyecto japonés de la quinta generación el nombre ha caído en desuso. Esta generación incluye la inteligencia artificial y sistemas expertos. En lugar de sólo ejecutar un conjunto de órdenes, el objetivo de estos sistemas es "pensar" y anticipar las necesidades de los usuarios. Estos sistemas se encuentran aún en desarrollo. Aunque los sistemas de inteligencia artificial han demostrado ser más complicados para desarrollar de lo que se anticipaba, los investigadores afirman que pronto serán capaces de tomar como entrada hechos y luego usar un procesamiento de datos que formule una respuesta adecuada, de modo similar a como responden los humanos.

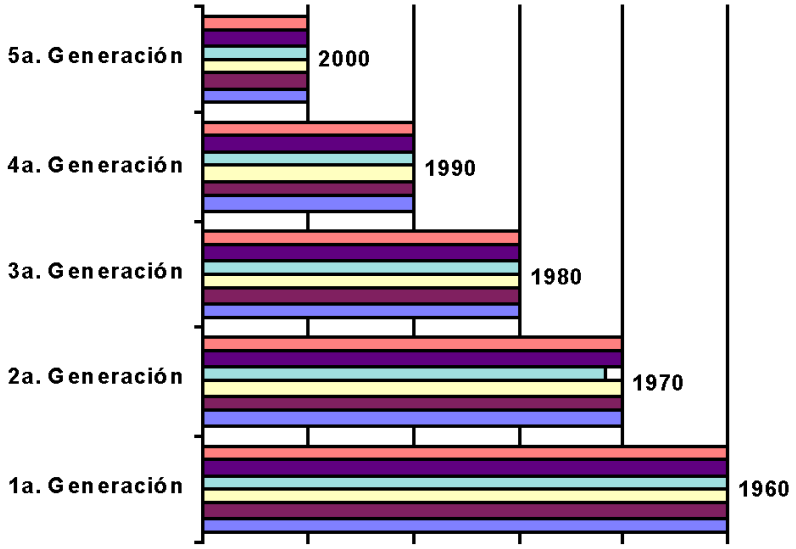

Ilustración 2-2. Cronología de las generaciones de los lenguajes de programación

El siguiente esquema muestra el desarrollo histórico de los lenguajes de programación más conocidos.

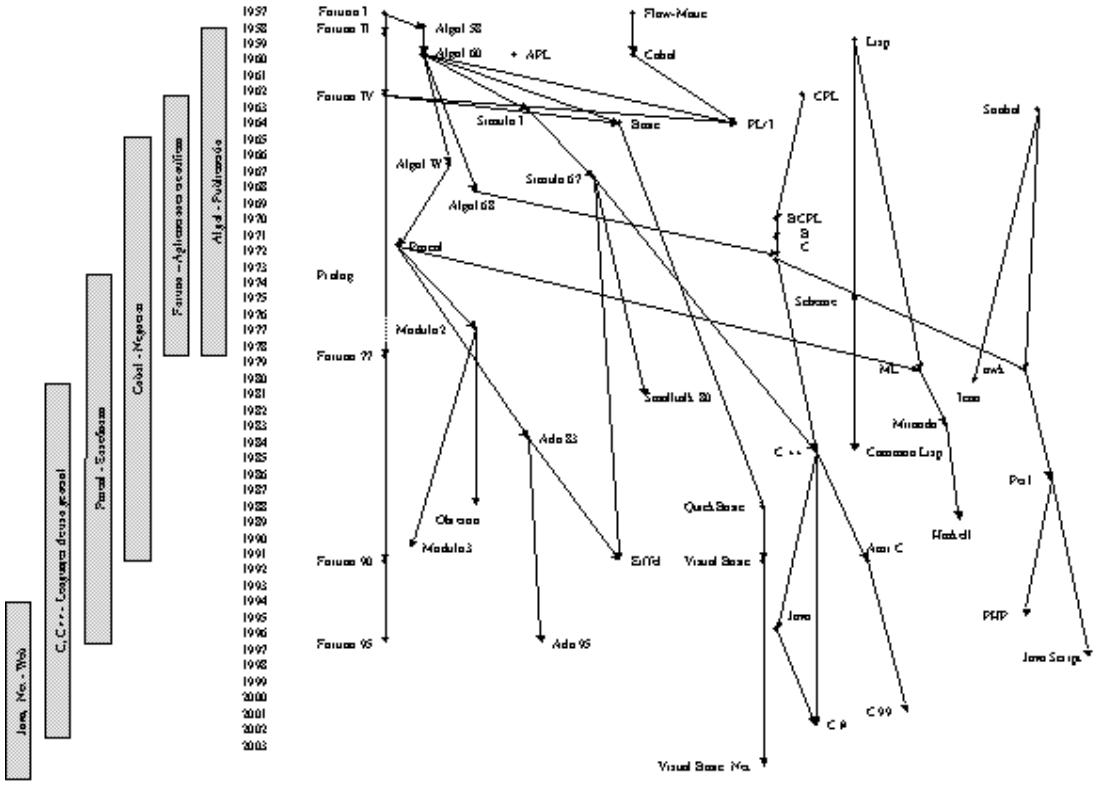

Ilustración 2-3. Desarrollo histórico de los lenguajes de programación

CRITERIOS DE EVALUACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

En este segundo capítulo revisaremos algunos de los principales lenguajes de programación que se identifican con algún o algunos de los paradigmas de programación descritos en el capítulo anterior, podremos observar que, por ejemplo, para el paradigma de programación orientado a objetos, tendremos una gran cantidad de lenguajes que lo soportan ya sea directa o indirectamente, pero, ¿cuál de todos ellos es el mejor?, o ¿cuál de ellos es el que debemos de utilizar para desarrollar nuestros sistemas? Si bien no se puede dar una respuesta concreta a estas preguntas, debido a que para algunos programadores un lenguaje de programación es mejor o más fácil de utilizar que otros, será muy difícil poner simplemente a dos programadores de acuerdo para que decidan cuál es mejor que otro. Por este motivo, son necesarios algunos criterios de evaluación que se enfocan en las características y capacidades de los lenguajes de programación y que ayuden al programador a decidir entre uno u otro, sin dejar a un lado el tipo de problema al cuál se presentan y quieren la solución computacional.

Entre los principales puntos en discusión para evaluar los lenguajes de programación está la facilidad de lectura, la facilidad de escritura, la integridad del código y el costo, los cuáles son explicados en los siguientes párrafos.

- **Facilidad de lectura**

Ilustración 2-4. Facilidad de lectura en los lenguajes de programación

Tal vez uno de los criterios de evaluación de los lenguajes de programación más importante es la facilidad con que los programas escritos con ellos pueden ser leídos y comprendidos. Antes de 1970, el desarrollo de *software*[2] fue una tarea tediosa y pensada en términos de escritura de código. Las principales características positivas de los lenguajes de programación eran su eficiencia para ser ejecutados y por lo tanto, para ser leídos por la máquina. Es decir, los lenguajes de programación eran diseñados más a partir del punto de vista de la máquina y no de los usuarios o los programadores. En los años setentas, el concepto de vida del *software* fue introducido en la educación de los nuevos programadores, y en este ciclo de vida, la codificación del programa fue relegado a una pequeña parte del ciclo y el mantenimiento del sistema fue reconocido como la parte más grande del ciclo de vida

del *software*[2], particularmente en términos de costo. Debido a que el mantenimiento es más barato si se dice que el código presenta facilidad de lectura y comprensión, estas dos características llegaron a convertirse en una importante medida de evaluación de los programas y los lenguajes de programación. Es entonces cuando empezó a haber un cambio en la orientación del diseño de los lenguajes, de programación, ya no eran orientados a las máquinas, sino al humano.

La facilidad de lectura de código debe de ser considerada en el contexto del tipo de problema que se presenta. Por ejemplo, si un programa que describe cierto procedimiento fue escrito en un lenguaje que no está diseñado para ese tipo de problemas, el código puede ser difícil de comprender incluso para aquellos programadores que están acostumbrados a ese lenguaje.

Ilustración 2-5. Facilidad de escritura en los lenguajes de programación

- **Facilidad de escritura**

La facilidad de escritura es una medida de evaluación para determinar qué tan fácil es crear programas en un determinado lenguaje de programación para resolver un problema categorizado dentro de los dominios de aplicación de la computación. Muchas de las características que afectan la facilidad de lectura y comprensión de código, afectan también a la facilidad de escritura del mismo. Esto se debe directamente al hecho de que el proceso de escritura de un programa requiere que el programador frecuentemente este repasando la parte del programa que ya está escrito.

Como es considerado en la facilidad de lectura del código, la escritura del mismo también debe de tomar en cuenta el contexto del problema del cuál se está tratando y del lenguaje de programación que se escogió para resolverlo. Sería simplemente irrazonable comparar la facilidad de escritura de dos lenguajes de programación en el caso de tratar de escribir el código que den la misma solución a un caso en particular, cuando un lenguaje fue diseñado para ese tipo de aplicaciones y el otro no lo fue. Por ejemplo, la facilidad de escribir un programa en Cobol y en Fortrán sería dramáticamente diferente si se deseara crear un programa que utilizara estructura de datos bidimensionales, siendo para estos casos, el lenguaje Fortrán el indicado. La facilidad de escribir un programa en estos lenguajes también diferiría si se tratara de escribir un programa para producir reportes complejos de negocios, siendo para estas aplicaciones el lenguaje Cobol el más apropiado.

Ilustración 2-6. Integridad en los lenguajes de programación

- Integridad

Se dice que un programa es integral si permite su ejecución bajo cualquier circunstancia que se presenta. En este caso, se habla de otras características que en su conjunto conllevan una integridad en los sistemas desarrollados bajo determinado lenguaje de programación. Así pues, podemos hablar de la inspección de tipos, que es una simple prueba de los tipos de errores de un programa dado, ya sea durante la fase de compilación o de ejecución. Debido a que los errores en tiempo de ejecución resultan ser más caros de resolver, son más deseables los errores en la etapa de compilación, de cualquier modo, cualquier error en cualquier etapa tendrá que ser reparado. El manejo de excepciones es otro punto que se toma en cuenta para la integridad de los programas, esto es, la detección de errores en tiempo de ejecución o de cualquier otra situación inusual ajena a la ejecución del mismo programa y que pudiera afectarlo; pero no solo es la detección de ello, sino el tomar las medidas correctivas y así continuar con la ejecución del programa. El tener dos o más métodos de referencia diferentes para la misma celda de memoria es llamado en inglés *aliasing*, esto es utilizado por ejemplo para que dos variables del mismo programa se refieran a la misma celda. Por último, la facilidad de lectura y escritura del mismo código, hace que un lenguaje de programación puede crear programas integrales.

Ilustración 2-7. Costo de los lenguajes de programación

- **Costo**

El costo total de utilizar un determinado lenguaje de programación es una función dependiente de muchas de sus características. En primer lugar, se debe de considerar el costo en la capacitación de los programadores en ese lenguaje. Esto también depende de la facilidad de sintaxis del propio lenguaje y de la experiencia de los programadores.

En segundo lugar, se debe de considerar el costo de la escritura de programas en ese lenguaje. Esto esta dado en función de su facilidad de escritura y de la manera acertada o desacertada en que se utilice para desarrollar *software*[2] para el cuál fue diseñado.

El tercer punto es el costo de la compilación de programas escritos en ese lenguaje. Entre más impedimentos se encuentren para compilar un determinado programa, mayor será su costo.

En cuarto lugar encontramos el costo de ejecución del programa, el cuál viene influenciado directamente por el diseño del lenguaje utilizado. Los lenguajes de programación que requieren una gran cantidad de inspecciones de tipos durante su ejecución producirán programas que son lentos en su ejecución.

El quinto punto en el costo de los lenguajes, es el costo del sistema de implementación del mismo lenguaje. Uno de los factores por los que Java ha sido tan popularmente aceptado es porque su sistema de compilador/intérprete estuvo disponible desde la aparición de su primera versión. Un lenguaje cuya implementación requiere de máquinas caras o de o de un determinado *hardware*[1], tenderá a no seguir siendo utilizado comúnmente por los programadores que no cuenten con esos recursos.

El sexto punto del costo de un lenguaje de programación es la integridad del mismo, ya que si un sistema falla en un caso crítico como lo es en una planta nuclear o en una máquina de rayos X, el costo por la misma falla podría ser muy alto. Las acciones de algunos sistemas no críticos pueden resultar muy caras en términos de prevenir la posible pérdida de información en un futuro o al prevenir costos de investigación sobre los errores del *software* en algún determinado caso de error.

El punto final en el costo de los lenguajes de programación, es el costo del mantenimiento de programas, el cuál incluye tanto correcciones como modificaciones para agregar nuevas capacidades a la aplicación. El costo del mantenimiento depende en un gran número a las características del lenguaje de programación, pero sobre todo, a la integridad del mismo. Debido a que el mantenimiento de las aplicaciones por lo general lo realiza personal ajeno a aquellos que lo diseñaron y escribieron el código fuente, la pobre integridad de los sistemas puede hacer que este costo sea extremadamente alto.

LENGUAJES DE PROGRAMACIÓN IMPERATIVOS

Ilustración 2-8. Lenguajes de programación imperativos

Algol

Algol es el acrónimo de *ALGOrithmic Language* (lenguaje algorítmico) y puede considerarse como el iniciador de la familia de lenguajes de programación estructurada. Fue diseñado para aplicaciones científicas en 1958 por un comité internacional en Zurich y fue el primero con sintaxis definida de manera normal y matemática.

Desde varios puntos de vista, Algol fue un descendiente de Fortrán ya que generalizó varias de las acciones de ese lenguaje de programación y agregó nuevas estructuras y conceptos. Una de las principales metas de Algol en comparación con su predecesor, fue el hacer que las aplicaciones diseñadas en él no se restringieran a un determinado tipo de *hardware*[1] para poder funcionar correctamente, así como el hacer el lenguaje más flexible y poderoso.

Es en 1960 cuando aparece el lenguaje Algol 6.0 el primer lenguaje estructurado en bloques. Este lenguaje fue muy popular en el segundo lustro de los 60's.

Su principal contribución es ser la raíz del árbol que ha producido lenguajes tales como pascal, C, C++, y Java[3].

[3] Para mayor referencia consultar http://www.prolog.org/

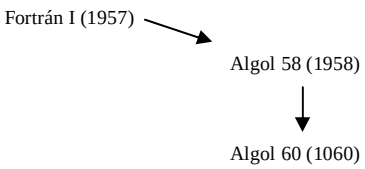

Ilustración 2-9. Genealogía de Algol

BASIC

BASIC es una familia de lenguajes de programación. Originalmente fue diseñado como una herramienta de enseñanza y para la programación de máquinas pequeñas, se difundió entre las computadoras caseras en la década de 1980 y sigue siendo popular hoy en día en muchas de sus versiones, algunas muy diferentes del original, por lo que se puede decir que es un lenguaje de programación no estandarizado.

BASIC es el acrónimo de *Beginners All-purpose Symbolic Instruction Code* que en español quiere decir "código de instrucciones simbólicas de propósito general para principiantes".

El lenguaje fue creado en 1964 por John George Kemeny y Thomas Eugene Kurtz en el Dartmouth College. El nuevo lenguaje debería de usar terminales como método de acceso a computadoras. Las metas del sistema fueron:

- Debería de ser un lenguaje de programación fácil de usar incluso para aquellos estudiantes que no fueran científicos.
- Debería de ser un lenguaje de programación amigable.
- Ser un lenguaje de propósito general.
- Ser interactivo.
- Debería de permitir accesos libre y privado.
- Proveer mensajes de error claros y amigables.
- Permitir que los expertos añadieran características avanzadas, mientras que el lenguaje permaneciera simple para los principiantes.
- Debería de considerar al tiempo del usuario más importante que el tiempo de la máquina.
- Responder rápido a los programas pequeños.
- No requerir un conocimiento de *hardware*[1] de la computadora.
- Proteger al usuario del sistema operativo.

Se trata principalmente de un lenguaje de programación imperativo, es decir secuencial, en donde una instrucción se ejecuta después de la anterior. La administración de memoria es más fácil que con muchos otros lenguajes procedimentales debido a un recolector de basura el cuál funciona a costa de la velocidad de ejecución. Variantes recientes como VisualBasic.Net han introducido características orientadas a objetos y hasta herencia en la última versión.

Los dialectos modernos de BASIC han añadido control de subrutinas, funciones y programación estructurada, así como los constructores de declaración de datos, tal y como los hacen C y Pascal.

C

C es un lenguaje de programación creado en 1969 por Ken Thompson y Dennis M. Rithcie en los Laboratorios Dell teniendo como base los lenguajes BCPL y B. Es un lenguaje orientado a la implementación de compiladores y sistemas operativos, por tal motivo, es el más popular para crear *software*[2] de sistemas, aunque también se utiliza para crear aplicaciones.

Dispone de las estructuras típicas de los lenguajes de alto nivel, aunque también dispone de construcciones del lenguaje que le permiten un control a muy bajo nivel. Es decir, su poder radica en que el código producido es bastante similar al que un programador lograría si lo escribiera en lenguaje ensamblador, pero con la diferencia de que en C está escrito casi en lenguaje natural. Su dificultad estriba, en que no es tan sencillo de aprender en comparación con otros lenguajes

Se trata de un lenguaje de programación muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado no llevado al extremo (permitiendo ciertas licencias rupturistas) y que como se mencionó, basa su funcionamiento en el paradigma de programación imperativa.

C ha sido un lenguaje muy popular entre los programadores, y por este motivo se ha visto en la necesidad de evolucionar al igual que las necesidades de las nuevas aplicaciones lo van requiriendo, por este motivo se han creado muchas versiones del lenguaje desde sus orígenes hasta hoy en día, algunas de ellas muy distintas al lenguaje C original. En algunas de las versiones más actuales se permite la

programación orientada a objetos como es el caso de C++, aunque la implementación original de éste fue un preprocesador que traducía código fuente de C++ a C.

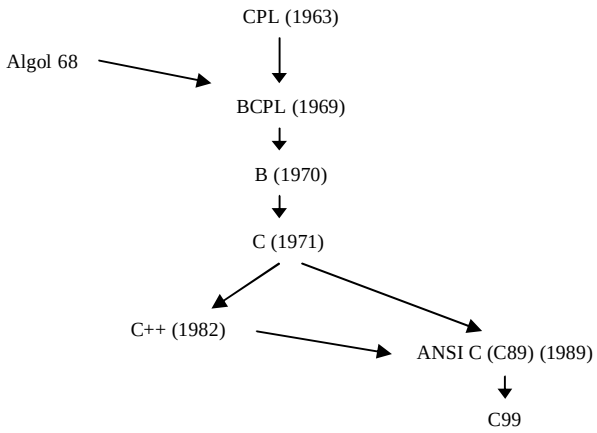

Ilustración 2-10. Genealogía de C

COBOL

COBOL corresponde a las siglas de *Common Business Oriented Language* (lenguaje general orientado a los negocios) y fue uno de los primeros lenguajes de programación diseñado en 1960 por la *Conference on Data Systems and Languages* (CODASYL); su predecesor fue el lenguaje Flow-Matic. El fin que se buscaba al desarrollar este lenguaje de programación fue el solucionar las incompatibilidades de los fabricantes de computadoras.

Las características que se pensaron que debía de tener a la hora de su diseño fue el ser un lenguaje de programación lo más parecido al idioma inglés, debería de ser fácil de usar y que no presentara problemas para su implementación en ningún ambiente y es así como su uso continúa vigente para aplicaciones comerciales y administrativas.

Sus únicos tipos de datos fueron cadenas y números. Lo que le dio la característica de poder agruparlos en arreglos sencillos, de modo que los datos podían ser organizados y seguidos de una mejor manera.

Un programa en COBOL consta de cuatro divisiones: identificadores, ambiente, datos y los procedimientos, que tienen que ser especificadas siempre. Los programas

tienden a ser grandes y están llenos de palabras y frases cortas en inglés, por lo que son bastante legibles, aunque no muy bien estructurados.

Su manejo de memoria es estático, aunque la mayoría de las versiones incluyen subsistemas de manejo dinámico de información en disco magnético.

Actualmente existen versiones que soportan la programación orientada a objetos.

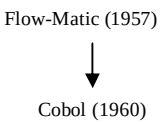

Ilustración 2-11. Genealogía de Cobol

FORTRÁN

Es el acrónimo de *FORmula TRANslation* (traducción de fórmulas). Diseñado en 1957 por IBM y cuyo líder de proyecto fue John Backus, fue el primer lenguaje de programación ampliamente usado, sobre todo en áreas de ingeniería y matemáticas, sin embargo, esto mismo hizo que no fuera muy común para el desarrollo de aplicaciones en computadoras personales. Su uso se dirige especialmente a equipos de cómputo dedicados a la investigación y la educación. La versión más actual (1990) fue denominada FORTRÁN-90.

Un programa en FORTRÁN puede estar constituido por un programa principal y varias subrutinas. Una de las ventajas de emplearlo es el acceso a una enorme cantidad de bibliotecas especializadas en manejos numéricos y matemáticos.

Pascal

Pascal es un lenguaje de programación de alto nivel estructurado, desarrollado por Niklaus Wirth en 1970. El Pascal, buscaba superar las limitaciones de los otros lenguajes de programación y demostrar la manera adecuada de implementar un lenguaje de cómputo. Se convirtió en uno de los lenguajes extensamente usados en los cursos de introducción a la programación, pues fue bien recibido como lenguaje de enseñanza para estudiantes universitarios.

Un programa en Pascal consiste en módulos que pueden ser anidados y llamados recursivamente. El lenguaje permite la definición de estructuras de datos más

allá de las tradicionales (entero, real, etc.), por lo que es muy adecuado para la programación estructurada.

Perl

Perl (*Practical Extraction and Report Language*) es un lenguaje de programación desarrollado por Larry Wall en 1987 e inspirado en algunas herramientas de UNIX como *sed, grep, awk, c-shell*, para la administración de tareas propias de UNIX.

El lenguaje puede considerarse como modular, estructurado u orientado a objetos ya que soporta todos estos estilos de programación aunque no es completamente soportado de forma directa por la orientación a objetos.

Se trata de un lenguaje de programación basado en scripts de tipo BCPL (como TCL o PHP) portable a casi cualquier plataforma. Es muy utilizado para escribir CGIs[4]. Uno de sus elementos más importantes son las expresiones regulares que a partir de su versión en Perl han sido adoptadas por otros lenguajes y plataformas como .Net y Javascript.

Entre otras de sus aplicaciones se encuentra que es un lenguaje optimizado para las labores de procesamiento de textos y archivos y para el escaneo de texto arbitrario en ficheros. Es también un buen lenguaje para tareas de administración de sistemas. Actualmente se utiliza para entornos gráficos en combinación con sistemas como Perl/TK o GTK[5].

SQL

Son las siglas de *Structured Query Languaje* (Lenguaje Estructurado de Consultas) y se trata de un lenguaje que auna las características del álgebra y cálculo relacionales que nos permite lanzar consultas contra una base de datos para recuperar información de nuestro interés y que se encuentra almacenada en ella[6].

[4] Interfaz Gráfica de Usuario
[5] Para mayor referencia consulte http://www.perl.org/
[6] Para mayor referencia consulte http://www.sql.org/

LENGUAJES DE PROGRAMACIÓN FUNCIONAL

Ilustración 2-12. Lenguajes de programación funcional

COMMON LISP

Common Lisp fue creado como un esfuerzo de combinar las acciones de varios dialectos de LISP creados en la década de los ochentas, incluyendo Scheme. Es un lenguaje de programación grande y complejo.

Es considerado como el estándar del lenguaje de programación LISP, se ha desarrollado como un lenguaje funcional procedimental, y que admite ambos tipos de construcciones: la composición funcional con transferencia referencial y la asignación de variables con secuencias de control en la ejecución del programa, por lo que no se considera como un lenguaje funcional puro.

Haskell

Con el lenguaje Haskell se pretendía unificar las características más importantes de los lenguajes funcionales, como las funciones de orden superior, evaluación perezosa, inferencia estática de tipos, tipos de datos definidos por el usuario, encaje de patrones y listas por comprensión. Al diseñar el lenguaje se observó que no existía un tratamiento sistemático de la sobrecarga con lo cual se construyó una nueva solución conocida como las clases de tipos[7].

[7] Para mayor referencia consulte http://www.haskell.org

ISWIM

ISWIM es una notación algorítmica en el estilo de un lenguaje de programación diseñado por Peter J. Landin y descrita por primera vez en 1966. El nombre del lenguaje es el acrónimo en inglés de If you See What I Mean.

Si bien nunca fue implementado, su influencia fue decisiva en el desarrollo de la programación funcional y podemos hablar de que los lenguajes SASL, Miranda y ML son sus sucesores directos.

LISP

LISP, acrónimo de *LIStProcessing,* es un lenguaje de programación funcional creado por John McCarthy y sus colaboradores en el MIT[8] en 1959.

El elemento fundamental de LISP es la lista, pues tanto los datos como los programas son listas, y es a través de estas que se expresan las funciones.

En LISP puro se distinguen dos elementos fundamentales:

- Átomos: los átomos son los símbolos, son datos elementales y pueden pertenecer a varios tipos (números, caracteres, cadenas de caracteres y símbolos).
- Listas: son secuencias de átomos o de listas encerradas entre paréntesis. Además existe una lista especial (*nil*) que es la lista nula ya que carece de algún elemento.

LISP sigue una filosofía de tratamiento no destructivo de los parámetros, de modo que la mayoría de las funciones devuelven una lista resultado de efectuar alguna transformación sobre la que recibieron, pero sin alterar esta última.

Uno de los motivos por los que LISP es especialmente adecuado para la IA y de que haya dominado este campo por un cuarto de siglo es el hecho de que el código y los datos tengan el mismo tratamiento (como listas); esto hace especialmente sencillo escribir programas capaces de escribir otros programas según las circunstancias, y así ser utilizado en aplicaciones tanto para el área de la psicología, lingüística y matemáticas[9].

[8] Siglas en inglés de Instituto Tecnológico de Massachussets.
[9] Para mayor referencia consulte http://www.lisp.org/

Scheme

El lenguaje de programación Scheme es un lenguaje funcional (impuro), y un dialecto de LISP. Fue desarrollado por Guy L. Steele y Gerald Jay Sussman en la década de los setenta.

La filosofía de Scheme es definitivamente minimalista. Su objetivo no es acumular un gran número de funcionalidades, sino evitar las debilidades y restricciones que hacen necesaria su adición. Scheme proporciona el mínimo número posible de nociones primitivas, construyendo todo lo demás basándose en este reducido número de abstracciones.

Fue el primer dialecto de LISP que usó ámbito estático o léxico (en lugar de dinámico) de forma exclusiva. También fue uno de los primeros lenguajes de programación con continuaciones explícitas.

Las listas son la estructura de datos básica del lenguaje. Debido a su especificación minimalista, no hay sintaxis explícita para crear registros o estructuras, o para programación orientada a objetos, pero muchas implementaciones ofrecen dichas funcionalidades.

Scheme facilita la programación funcional ya que no precisa de valores globales ni sufre de efectos secundarios, siendo por tanto, seguro en presencia de procesos concurrentes. También facilita la verificación de programas, por lo menos en comparación con los lenguajes de programación imperativos.

En Scheme, los procedimientos son objetos de primera clase. Ello permite la definición de funciones de orden superior, que facilita un mayor grado de abstracción de los programas. También es posible la creación de procedimientos anónimos.

LENGUAJES DE PROGRAMACIÓN LÓGICA

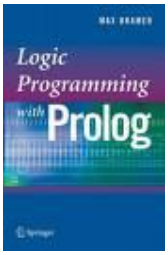

Ilustración 2-13. Lenguajes de programación lógica

PROLOG

El lenguaje de programación lógica por excelencia es Prolog, cuyo nombre proviene del francés *Programation Logique*. Se trata de un lenguaje de programación lógico e interpretado y que es bastante popular en el medio de la investigación y la inteligencia artificial (IA).

Se trata de un lenguaje de programación ideado a principios de los 70s en la Universidad de Aix-Marseille por los profesores Alain Colmerauer y Phillipe Roussel. Inicialmente se trataba de un lenguaje totalmente interpretado hasta que a mediados de los 70s, David H. D. Warren desarrolló un compilador que era capaz de traducir Prolog en un conjunto de instrucciones de una máquina abstracta denominada *Warren Abstract Machine* (WAM). Desde entonces Prolog es un lenguaje semi-interpretado.

En Prolog, el orden de ejecución de las instrucciones no tiene nada que ver con el orden en que fueron escritas. Tampoco hay instrucciones de control propiamente dichas. Para trabajar con este lenguaje, un programador debe acostumbrarse a pensar de una manera muy diferente a la que se utiliza en los lenguajes clásicos.

Las instrucciones de Prolog se llaman "reglas" o "cláusulas de Horn" y esencialmente pueden representarse así: "Hacer esto si se cumplen tales o cuales condiciones". Una instrucción se ejecutará automáticamente en cualquier momento en que se cumplan las condiciones especificadas. Además de las reglas, también se definen "*factors*" en la cual se aplicarán las reglas. Es decir, en Prolog, un programa consiste de una descripción lógica de una teoría y la computación es la deducción de la teoría para una consulta dada. Un programa escrito en este lenguaje se compone de

cláusulas que constituyen reglas del tipo *modus ponens*, es decir, "Si es verdad el antecedente, entonces es verdad el consecuente". No obstante, la forma de escribir las cláusulas es al contrario de lo habitual. Primero se escribe el consecuente y luego el antecedente. El antecedente puede ser una conjunción de condiciones que se denomina secuencia de objetivos. Cada objetivo se separa con una coma y puede considerarse similar a una instrucción o llamada a procedimiento de los lenguajes imperativos. Su ejecución se basa en dos conceptos: la unificación y el *backtracking*. Gracias a la unificación, cada objetivo determina un subconjunto de cláusulas susceptibles de ser ejecutadas. Cada una de ellas se denomina punto de elección. Prolog selecciona el primer punto de elección y sigue ejecutando el programa hasta determinar si el objetivo es verdadero o falso. En caso de ser falso entra en operación el *backtracking*, que consiste en deshacer todo lo ejecutado situando el programa en el mismo estado en el que estaba justo antes de llegar al punto de elección; entonces se toma el siguiente punto de elección que estaba pendiente y se repite de nuevo el proceso. Todos los objetivos terminan su ejecución ya sea en “verdadero” o en “falso”.

Prolog se enmarca en el paradigma de los lenguajes lógicos y cuenta con diversas variantes. La más importante de estas variaciones es la programación lógica con restricciones, que posibilita la resolución de ecuaciones lineales además de la resolución de hipótesis. Esto lo hace enormemente diferente de los lenguajes de programación más comunes como lo son C, Java, etc[10].

[10] Para mayor referencia consulte http://www.prolog.org/

LENGUAJES DE PROGRAMACIÓN ORIENTADA A OBJETOS

Ilustración 2-14. Lenguajes de programación orientada a objetos

Ada

Ada, que debe su nombre a Ada Byron, condesa de Lovelance (1815-1852), asistente de Babbage en el desarrollo de la máquina analítica, fue un intento más por tener el único lenguaje de programación de uso verdaderamente universal. Es un lenguaje de programación estructurado y fuertemente tipado de forma estática diseñado por Jean Ichbiah de Cii Honeywell Bull, aunque también es un lenguaje multipropósito orientado a objetos y concurrente por lo que es colocado en esta clasificación.

Ada se usa principalmente en entornos en los que se necesita una gran seguridad y confiabilidad como la defensa. Se trata de un lenguaje de características avanzadas, con manejo dinámico de memoria y recursividad así como facilidades integradas para programación concurrente aunque causó polémica en términos académicos, porque se argüía que el lenguaje (y su correspondiente compilador) era demasiado grande y poco elegante, y la tendencia moderna de lenguajes de programación es hacia los pequeños y funcionales.

C++

C++ fue diseñado por Bjarne Stroustrup en 1982 y fue el segundo intento de proporcionar características orientadas a objetos al lenguaje de programación C. Es la

versión más difundida del C original y aceptada. Se trata de un lenguaje híbrido que combina la flexibilidad y el acceso de bajo nivel de C con las características de programación orientada a objetos como la abstracción, encapsulación y ocultación, así mismo, soporta el uso de plantillas o programación genérica (templates) y utiliza recursos gráficos estandarizados.

Por todo lo anterior, se puede decir que C++ abarca tres estilos de programación: programación estructurada (que tiene características de la programación imperativa), la programación genérica y la programación orientada a objetos.

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que dificulta mucho su aprendizaje.

Además de lo anterior, C++ posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (RTTI)

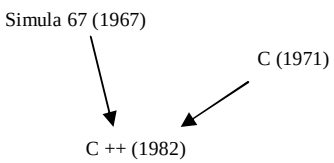

Ilustración 2-15. Genealogía de C++

C#

C# (que se pronuncia como “si sharp” o C sostenido), es un lenguaje de programación orientado a objetos desarrollado por Microsoft y estandarizado como parte de su plataforma .Net.

Su sintaxis toma como base a los lenguajes de programación C y C++ y utiliza el modelo de objetos de la plataforma .Net el cuál es similar al de Java aunque incluye mejoras derivadas de otros lenguajes. C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como Visual Basic.

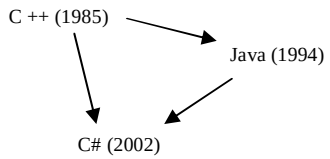

Ilustración 2-16. Genealogía de C#

Clarion

Clarion es un lenguaje de programación de cuarta generación además de ser un entorno integrado de desarrollo de Softvelocity orientado a la programación de aplicaciones de bases de datos. Es compatible con una gran cantidad de bases de datos incluyendo todas las de formato SQL, ADO, y XML.

El generador de aplicaciones junto con una serie de plantillas predefinidas y personalizadas y las clases ABC (*Application Builder Class*), trabajan para producir código orientado a objetos pre-testado.

Delphi

Delphi (que hace referencia al oráculo de Delfos) es un entorno de desarrollo rápido de *software*[2] diseñado para la programación de propósito general con énfasis en la programación visual, siendo un poderoso entorno de programación para el sistema operativo Windows. Utiliza como lenguaje de programación una versión moderna de Pascal denominada Object Pascal, la cuál tiene soporte para la programación orientada a objetos y mejoras a versiones anteriores en cuanto a objetos, encapsulación, herencia y polimorfismo, además de las mejoras en cuanto a la programación orientada a eventos que permite de manera sencilla ejecutar trozos de código en respuesta a acciones o eventos (sucesos) que ocurren durante el tiempo que un programa se ejecuta.

Como entorno visual, la programación de Delphi consiste en diseñar los formularios que componen al programa colocando todos sus controles (botones, etiquetas, campos de texto, etc.) en las posiciones deseadas. Luego se asocia código a los eventos de dichos controles y también se pueden crear módulos, que regularmente contienen los componentes de acceso a datos y las reglas de negocio de una aplicación.

Delphi introdujo la idea del uso de componentes, que son piezas reutilizables de código (clases) que pueden interactuar con el EID[11] en tiempo de diseño y desempeñar una función específica en tiempo de ejecución.

Se utiliza principalmente para el desarrollo de aplicaciones visuales y de bases de datos cliente-servidor y multicapas. Debido a que es una herramienta de propósito general, también es utilizado para proyectos de casi cualquier tipo, incluyendo aplicaciones de consola, CGI[12] y servicios del sistema operativo.

Una de las principales características de Delphi es su capacidad para desarrollar aplicaciones con conectividad a bases de datos de diferentes fabricantes. El programador de Delphi cuenta con una gran cantidad de componentes para realizar la conexión, manipulación, presentación y captura de los datos. Éstos componentes de acceso a datos pueden enlazarse a una gran cantidad de controles visuales, aprovechando las características del lenguaje orientado a objetos, gracias al polimorfismo.

Delphi permite de manera sencilla ejecutar trozos de código en respuesta a acciones o eventos (sucesos) que ocurren durante el tiempo en el que un programa se ejecuta. Los eventos pueden generarse debido a la recepción de señales desde elementos de *hardware*[1] como el ratón o el teclado, o pueden producirse al realizar alguna operación sobre el elemento de la propia aplicación (como abrir un conjunto de datos)

Existe una versión de Delphi para sistemas Unix y Linux denominada Kylix aunque fue abandonada por Borlan en su versión 3.0.

Eiffel

Eiffel es un lenguaje de programación orientado a objetos ideado en 1985 por Bertrand Meyer. Su principal aplicación es el diseño de *software*[2] robusto, utilizando una sintaxis parecida a la de Pascal pero caracterizado por permitir el diseño por contrato desde la base, con precondiciones, postcondiciones, invariantes y variantes de bucle, invariantes de clases y asertos. Es un lenguaje con tipos fuertes, pero relajado por herencia.

[11] De las siglas en inglés IDE (Integrated Development Environment)
[12] Interfaz Gráfica de Usuario

Lexico

Lexico es un lenguaje didáctico en español utilizado para el aprendizaje de la programación orientada a objetos. Se ejecuta sobre la plataforma .Net de Microsoft y soporta las características exigidas internacionalmente para considerarse puro con respecto al paradigma.

Es lo suficientemente sencillo para que los estudiantes puedan practicar los conceptos básicos de algoritmos y las estructuras fundamentales en lógica (estructuras secuenciales, selectivas e iterativas).

Posee el soporte necesario para la orientación a objetos y lo ha simplificado al máximo de manera que ayude a la inmersión en los conceptos.

Objective-C

Este lenguaje fue un primer intento de proporcionar soporte para la programación orientada a objetos en C, de escasa difusión, pero actualmente usado en Mac OS X y GNUstep.

Fue creado como un superconjunto de C por Brad Cox y la corporación StepStone en 1980, teniendo un estilo muy parecido al de Smalltalk. En 1988 fue adoptado como lenguaje de programación de NEXTSTEP y en 1992 fue liberado bajo la licencia de GNU para el compilador GCC.

Ocaml

El lenguaje de programación Objective CAML, también llamado Ocaml u O´Caml, es un lenguaje de programación avanzado de la familia de los lenguajes ML, desarrollado y distribuido por el INRIA.

Se trata de un lenguaje híbrido que admite los paradigmas de programación imperativa, funcional y orientada a objeto, precisamente, nace de la evolución del lenguaje CAML (*Categorical Abstract Machine Language*) al ingresársele la programación orientada a objetos.

Ocaml dispone de un análisis de tipos estático con interferencia de tipos, con valores funcionales de primera clase, polimorfismo parametrizado, llamada por

patrones, manejo de excepciones, recolección de basura y otras características avanzadas.

Oz

Oz es un lenguaje de programación multiparadigma desarrollado en el Laboratorio de Programación de Sistemas en la Universidad de Saarland por Gert Smolka a comienzos de 1990.

Contiene una forma simple y bien hecha de la mayoría de los conceptos de los principales paradigmas de programación, incluyendo programación lógica, funcional, imperativa, orientada a objetos, con restricciones, distribuida, y concurrente.

Como complemento a la programación multiparadigma, las principales ventajas de Oz radican en la programación con restricciones y la programación distribuida. Debido a su diseño, Oz implementa un modelo de programación distribuido que hace a la red transparente.

Para la programación con restricciones, Oz introduce la idea de espacios de computación, los cuáles permiten búsquedas definidas por el usuario y estrategias de distribución que son ortogonales al dominio de restricciones.

PHP

PHP es un lenguaje de programación usado generalmente para la creación de contenido para sitios *web*. El nombre es el acrónimo de *Hypertext Preprocessor* y se trata de un lenguaje interpretado usado para la creación de contenido dinámico para sitios web creado en 1994 por Rasmus Lerdorf.

En PHP también es posible crear aplicaciones con una interfaz gráfica para el usuario (GUI), utilizando la extensión PHP-GTK (la adaptación para PHP del entorno gráfico de Gimp), permitiendo desarrollar aplicaciones de escritorio tanto para los sistemas operativos basados en Unix, como para Windows y Mac OS y puede ser utilizado como lenguaje de *scripting* en consola, al estilo de Perl, en Linux, Windows y Mac.

Su interpretación y ejecución se realizan en el servidor que es donde se encuentra almacenado el script y el cliente sólo recibe el resultado de la ejecución. Cuando el cliente hace una petición al servidor para que le envíe una página *web*

generada por un script PHP, el servidor ejecuta el intérprete el cuál procesa el script solicitado que generará el contenido de manera dinámica, pudiendo modificar el contenido a enviar y regresa el resultado al servidor, el cuál se encarga de regresárselo al cliente.

El lenguaje permite la conexión a diferentes servidores de bases de datos tales como MySQL, Postgres, Oracle, ODBC, IBM DB2, Microsoft SQL Server, Firebird y SQLite, lo cuál permite la creación de aplicaciones web muy robustas.

Aunado a las características anteriores, PHP es un lenguaje multiplataforma que permite las técnicas de programación orientada a objetos[13].

Java

Java es un lenguaje de programación orientado a objetos desarrollado por James Gosling y sus compañeros de Sun Microsystems al inicio de la década de 1990. A diferencia de los otros lenguajes de programación más comunes, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un *bytecode* que es ejecutado (usando normalmente un compilador JIT), por una máquina virtual Java y que le permite ser ejecutados igualmente en cualquier tipo de *hardware*[1], a lo que se dice que Java es independiente de la máquina donde opera. Se debe tener presente que, aunque hay una etapa explícita de compilación, el *bytecode* generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (*Just In Time*). Hay implementaciones del compilador de Java que convierte el código fuente directamente en código objeto nativo, como lo es GCJ. Esto elimina la etapa intermedia donde se genera el *bytecode*, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura. Si el *bytecode* es interpretado, será más lento que si se usa el código máquina intrínseco de la plataforma destino. Si es compilado, durante la carga inicial o la ejecución, la penalización está en el tiempo necesario para llevar a cabo la compilación.

De este modo, la plataforma Java consta de tres partes:

- El lenguaje de programación
- Máquina virtual de Java
- API Java

[13] Para mayor referencia consulte http://www.php.org/

Algunas características del propio lenguaje conllevan una penalización en tiempo, aunque no son únicas de Java. Algunas de ellas son el chequeo de los límites de arreglos, chequeo en tiempo de ejecución de tipos y la indirección de funciones virtuales.

El uso de un recolector de basura para eliminar de forma automática aquellos objetos no recurridos, añade una sobrecarga que puede afectar al rendimiento, o ser apenas apreciable, dependiendo de la tecnología del recolector y de la aplicación en concreto. Las JVM modernas usan recolectores de basura que gracias a rápidos algoritmos de manejo de memoria, consiguen que algunas aplicaciones puedan ejecutarse más eficientemente.

Java fue diseñado para ofrecer seguridad y portabilidad, y no ofrece un acceso directo al *hardware*[1] de la arquitectura ni al espacio de direcciones. Java no soporta expansión de código ensamblador aunque las aplicaciones pueden acceder a características de bajo nivel usando librerías nativas JNI (*Java Native Interfaces*).

La sintaxis del lenguaje está basada en los lenguajes de programación C y C++, pero tiene un modelo de objetos mucho más simple y elimina elementos de bajo nivel como los punteros teniendo un funcionamiento muy similar al de Smalltalk. Incorpora sincronización y manejo de tareas en el lenguaje mismo (similar a Ada) e incorpora interfaces como un mecanismo alternativo a la herencia múltiple de C++.

Java es utilizado actualmente para numerosas aplicaciones. Por ejemplo, en la parte del cliente de una página *web*, puede ser encontrado en aplicaciones muy complejas como lo son los juegos y en el área del servidor, muchas de las páginas *web* contienen partes de código (si no es que todo el código) escrito con lenguaje Java, así podemos hablar de los Applets que son programas escritos en Java incrustados en otras aplicaciones, como lo pueden ser las páginas *web* que se muestran en el navegador.[14]

El lenguaje java se creó con los siguientes objetivos principales:

1.- Debería usar la metodología de la programación orientada a objetos.

2.- Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.

3.- Debería incluir por defecto soporte para trabajo en red.

[14] Para mayor descripción del lenguaje de programación Java se puede consultar la siguiente página de Internet: http://java.sun.com

4.- Debería diseñarse para ejecutar código en sistemas remotos de forma segura.

5.- Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos como C++.

En el estado del arte en la programación de red, Java promete extender el papel de Internet desde el terreno de las comunicaciones hacia una red en la cual puedan ejecutarse las aplicaciones completas. Su novedosa tecnología permitirá a los negocios proporcionar servicios de transacción a gran escala y en tiempo real y contener información interactiva en Internet. Java simplifica también la construcción de agentes *software*[2], programas que se mueven a través de la red y desempeñan funciones en ordenadores remotos en nombre del usuario. En un futuro cercano, los usuarios podrán enviar agentes *software* desde sus PC's hacia Internet para localizar información específica o para realizar transacciones en el menor tiempo posible en cualquier lugar del mundo.

Las aplicaciones de Java para la PC de escritorio no son tan comunes de encontrar debido a algunos requerimientos mínimos del sistema para su uso, sin embargo, hay aplicaciones Java cuyo uso está ampliamente extendido, como los NetBeans, el entorno de desarrollo (IDE) Eclipse, y otros programas como LimeWire y Azureus para intercambio de archivos. Java también es el motor que usa MATLAB para el renderizado de la interfaz gráfica y para parte del motor de cálculo. Las aplicaciones de escritorio basadas en la tecnología Swing y SWT (*Standard Widget Toolkit*) suponen una alternativa a la plataforma .Net de Microsoft.

Smalltalk

Smalltalk es un sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

Un sistema Smalltalk está compuesto por:

- Máquina virtual
- Imagen virtual que contiene todos los objetos del sistema
- Lenguaje de programación (también conocido como Smalltalk)
- Biblioteca de objetos reutilizables

- Entorno de desarrollo

El origen de este sistema radica en las investigaciones realizadas por Alan Kay, Dan Ingalls, Ted Kaehler, Adele Goldberg y otros durante los años sesenta en el Palo Alto Research Institute de Xerox.

Es considerado como el primero de los lenguajes orientados a objetos ya que en el sistema, todo es un objeto, incluidos los números reales o el propio entorno.

Como lenguaje de programación, tiene las siguientes características:

- Orientación a objetos pura
- Tipado dinámico
- Interacción entre objetos mediante envío de mensajes
- Herencia simple y con raíz común
- Reflexión completa
- Recolección de basura
- Multiplataforma
- Interpretado

Pero Smalltalk no es meramente un lenguaje, sino un entorno completo, prácticamente un sistema operativo que se ejecuta encima de una "máquina virtual". Esto asegura su máxima portabilidad entre plataformas[15].

Visual Basic

Visual Basic es un lenguaje de programación desarrollado por Alan Cooper para Microsoft y presentado en 1991 cuyo objetivo era simplificar la programación utilizando un ambiente de desarrollo completamente gráfico que facilitara la creación de interfaces gráficas. Es un dialecto de BASIC con un soporte mucho mayor que el lenguaje original.

Es un lenguaje fácil de aprender guiado por eventos y centrado en un motor de formularios que facilita el desarrollo de aplicaciones gráficas. Su sintaxis, basada en el antiguo BASIC, ha sido ampliada con el tiempo al incluirse características de los lenguajes estructurados modernos, se le ha agregado una implementación de programación orientada a objetos (por lo que se ha colocado en esta categoría), admitiendo el polimorfismo mediante el uso de interfaces, aunque no admite la herencia.

[15] Para mayor referencia consulte http://www.smalltalk.org/

Posee varias bibliotecas para manejo de bases de datos, pudiendo conectar con cualquier base de datos a través de ODBC (Informix, Dbase, Access, MySQL, SQL Server, Postgre SQL, etc.) a través de ADO.

Es utilizado principalmente para el desarrollo de gestión de datos en empresas, debido a la rapidez con la que se puede diseñar un programa que utiliza bases de datos y utilizando un ambiente gráfico. Sin embargo, no es muy recomendable para aplicaciones de grandes bases de datos, aplicaciones multimedia, videojuegos, editores gráficos, etc.

El principal inconveniente es que sólo existe un compilador e IDE llamado igual que el lenguaje, que al ser propiedad de Microsoft, sólo genera ejecutables para Windows y no existe forma alguna de exportar el código a otras plataformas.

VisualBasic.Net

Forma parte de la plataforma .Net y se trata de un lenguaje prácticamente equivalente en funcionalidad a C# añadiendo capacidades de programación orientada a objetos que en sus versiones anteriores de Visual Basic no se permitían, por ejemplo, la herencia de los objetos.

En esta nueva versión, se puede citar la oportunidad de definir ámbitos de tipo, clases que pueden derivarse de otras mediante herencia, sobrecarga de métodos, nuevo control estructurado de excepciones o la creación de aplicaciones con múltiples hilos de ejecución, además de contra con la extensa librería .Net, con la que es posible desarrollar tanto *Windows Applications* y *Web Forms*, así como un extenso número de clientes para bases de datos.

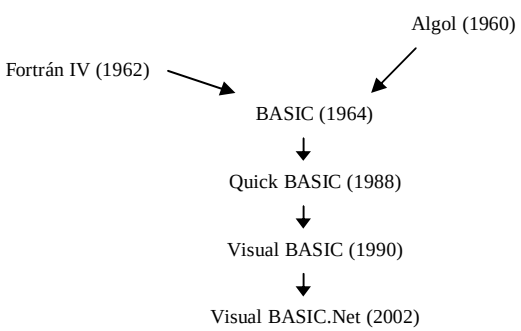

Ilustración 2-17. Genealogía de Visual BASIC.Net

CAPÍTULO 3

"CONCLUSIONES Y COMPARACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN"

COMPARACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN

El estudio de los paradigmas de programación es de vital importancia para todas aquellas personas que se dedican o tienen relación alguna con el desarrollo de sistemas computacionales y por qué no decirlo, ayuda en cierta forma a entender el funcionamiento de estos sistemas por parte de los usuarios finales quienes son los que estarán en contacto con el programa mucho más tiempo que el propio desarrollador.

Las principales ventajas del estudio de los paradigmas son que al programador lo llena de una amplia visión sobre las diferentes técnicas de programación de *software* y de los lenguajes de programación que son más óptimos para cada uno de ellos, desarrollando habilidades que tal vez el programador no habían explotado con el lenguaje de programación al cuál podría estar acostumbrado a usar ya sea por costumbre, por imposición a utilizarlo o por falta de experiencia en el ámbito del desarrollo. Esta habilidad no sólo será para realizar nuevos proyectos, sino también para manejar, comprender y dar mantenimiento a sistemas que se encuentran ya productivos y con los cuáles no hubo participación en la etapa de desarrollo.

La clasificación de los paradigmas de programación se puede dar desde diferentes aspectos, entre ellos están el tipo de expresiones básicas que utilizan en la codificación de un programa lo que va muy relacionado con el estilo de programación, ya sean sentencias de alto o bajo nivel; por otro lado, la principal clasificación de los paradigmas es dependiendo del tipo de solución que se puede alcanzar con cada uno de ellos, así están los paradigmas procedimentales u operacionales que nos dan un estado inicial del proceso y a partir de ahí se describe cómo se llegará a la solución; los que entregan una solución declarativa que se construye a partir de reglas, restricciones, ecuaciones y transformaciones que están descritas en el código del programa y aquellos paradigmas que entregan una solución demostrativa en los cuáles el programador no especifica un procedimiento de cómo se generará la solución ya que en lugar de esto presenta soluciones de problemas similares y le permite al sistema generalizar una solución procedimental a partir de estas demostraciones.

Los cuatro paradigmas de programación más difundidos y por lo tanto conocidos y puestos en práctica son el Paradigma de Programación Imperativa, Paradigma de Programación Funcional, Paradigma de Programación Lógica y el Paradigma de Programación Orientada a Objetos. Cada unos de ellos tienen características muy

específicas que lo diferencian de los otros, aunque actualmente los lenguajes de programación están permitiendo la cohesión de estas características dando la oportunidad al programador de utilizar distintas metodologías dentro del mismo desarrollo.

A continuación se presentarán algunas comparaciones entre los paradigmas de programación más significativos que se encuentran hoy en día en el mundo de la computación y que fueron detallados a lo largo del presente trabajo. Cabe aclarar que esta comparación no tiene como objetivo el indicar cuál es mejor sobre otro, ya que como se mencionó oportunamente, la elección del mejor paradigma de programación para resolver cierto problema computacional dependerá principalmente del tipo de problema al cuál nos estemos enfrentando, la solución que le queremos dar y el objetivo que queremos cumplir.

Variaciones de los paradigmas de programación

El paradigma de programación imperativa es tal vez la metodología más conocida y utilizada en el desarrollo de los sistemas, esto tiene razón ya que fue de las primeras técnicas utilizadas para el desarrollo de *software* y que es la técnica más difundida en las escuelas y universidades. De ahí que sea una metodología que cuenta con el mayor número de lenguajes de programación que basan su funcionamiento en este paradigma. Algunos de ellos han sido la base para las nuevas generaciones de los lenguajes de programación o han ido evolucionando para permitir la utilización de otros paradigmas dentro del mismo lenguaje y así ampliar su espectro de aplicación.

Entre las variantes de la programación imperativa se encuentra la programación estructurada que basa su funcionamiento en tres estructuras esenciales: estructuras secuenciales que indican paso a paso el flujo del funcionamiento del programa; estructuras selectivas que utilizan criterios de evaluación para determinar cuál es el siguiente punto de ejecución del programa; y estructuras iterativas o bucles de iteración que hacen repetir cierto bloque de instrucciones mientras se cumpla una condición. La programación modular es otra variante de la programación imperativa y permite agrupar un conjunto de procedimientos afines junto a los datos que manipulan para ser tratados como un todo, es decir, un programa se divide en partes bien diferenciadas

lógicamente, llamadas módulos, y éstos pueden ser analizados y programados por separado.

La programación con restricciones es una técnica utilizada para la descripción y solución de problemas combinatorios en donde se especifican un conjunto de restricciones, las cuáles deben satisfacer una solución, en vez de especificar los pasos para obtener dicha solución y está muy relacionado con la programación lógica, la diferencia radica principalmente en sus estilos y enfoques de modelado del mundo.

Por último podemos citar a la programación orientada a eventos como una variante de la programación orientada a objetos en la cuál tanto la estructura como la ejecución de los programas están controlados por un bucle continuo que responde a los sucesos que ocurran en el sistema o que ellos mismos provoquen en un orden no predecible, es decir, el mismo usuario (o sistema si es que se controla a sí mismo) será el que dirija el flujo del programa ya que no se predice la secuencia de control que se va a producir.

Elementos principales de los paradigmas

Las principales diferencias entre los diferentes paradigmas de programación radican en las estructuras básicas que utilizan como técnica de programación y su aplicación dentro de los códigos generados por los lenguajes de programación representativos de cada uno de ellos. Es decir, el programador deberá definir en la etapa de análisis del desarrollo qué estructuras son las más adecuadas para escribir el código que generará el programa que le dará solución al problema, así tendrá la opción de decidir si se tratará de una secuencia completa de pasos que deberá cumplirse de inicio a fin, o bien, utilizar módulos separados que resuelvan propósitos específicos dentro de la solución completa al problema; deberá decidir si se utilizarán variables que dependan o no de otros valores, o bien, la solución únicamente dependerá de las validaciones de elementos que se vayan generando durante la ejecución del sistema; entre otros puntos que son característicos de cada uno de los paradigmas. Estas decisiones son las que determinarán el paradigma más adecuado para el entendimiento de la situación a la cuál se está enfrentando y a la solución que le dará con un lenguaje de programación específico.

En el paradigma de programación funcional, el elemento principal que determina el funcionamiento de un programa son las funciones y éstas transforman datos, los cuales sólo tienen posibilidad de existir como valores que se pasan a una función o que son devueltos por ella pero ahí termina su existencia; no tienen vida fuera de las funciones.

En la programación imperativa, por otra parte, los datos se pueden considerar los elementos fundamentales de un programa. Existen por sí mismos y son usados y modificados por las funciones. La posibilidad de la programación imperativa de mantener y modificar un cierto estado (datos, valores de variables) la hace muy potente para representar y modelar procesos del mundo real que serían complicados de expresar en forma de programación funcional. Las funciones son utilizadas para modificar el estado del programa. Es así, como la programación imperativa consiste en determinar qué datos son requeridos para el cálculo, asociar a esos datos unas direcciones de memoria y efectuar paso a paso una secuencia de transformaciones en los datos almacenados, de forma que el estado final represente la solución a un problema específico.

En el paradigma de programación lógica se basa en un subconjunto del cálculo de predicados incluyendo instrucciones escritas en formas conocidas como cláusulas de Horn.

En la programación orientada a objetos el concepto fundamental es el objeto que es una entidad con un estado y unas funciones que pueden acceder y modificar este estado. Las clases son las plantillas que definen la estructura de un conjunto de objetos que tienen ciertas características semejantes y se diferencían entre sí por su identidad.

Dominios de aplicación

Debido a la gran variedad de uso de las computadoras, se han diseñado técnicas y lenguajes de programación que tienen por objetivo ser la herramienta principal para el desarrollo de diferentes sistemas con características muy particulares y diferentes unos de otros. Así por ejemplo tenemos que la programación científica que tiene estructuras de datos sencillas y que requieren del cálculo de operaciones complejas con punto flotante, basan el desarrollo de sus sistemas en la programación

imperativa. La programación imperativa también es la base para el desarrollo de los sistemas operativos, ya que se basan en ejecuciones de instrucciones secuenciales de bajo nivel.

Los sistemas de gestión de información requieren del uso de lenguajes de programación que se caractericen por la facilidad de generación de reportes ya elaborados, la precisión en la descripción y almacenaje de información y la habilidad para especificar las operaciones aritméticas decimales. Es por estos puntos que los lenguajes de programación imperativos son utilizados principalmente para el detallado de las reglas de negocios dentro de este tipo de sistemas, en conjunto, los lenguajes de programación declarativos se utilizan para la generación de informes y recuperación de la información de las bases de datos y como interfaz con los usuarios se puede utilizar la programación orientada a eventos para aquellas aplicaciones de cliente-servidor dentro del denominado comercio electrónico. Es aquí en donde encontramos la principal aplicación de los lenguajes de programación orientados a objetos, la *web,* en donde se han explotado como en ningún otro caso el modelado de los objetos y la herencia de clases, y la programación orientada a eventos es indispensable para la interacción con los usuarios tal y como lo es también en los sistemas multimedia.

Por otro lado tenemos que tanto la programación lógica como la funcional encuentran su principal aplicación en el desarrollo de sistemas de inteligencia artificial y en los sistemas expertos, ya que buscan la imitación del comportamiento humano a través de un universo de conocimientos que se les proporciona a los sistemas para que sean capaces de deducir el camino para llegar a la solución al problema a través de la lógica y la cognición.

Principales lenguajes de programación

Como se comentó en el capítulo 2, existe una gran variedad de lenguajes de programación que son utilizados en la actualidad para el desarrollo de sistemas independientemente de cuál sea la solución que se busca implementar, sin embargo existen algunos lenguajes que han sido más populares o difundidos entre el mundo de los programadores. Algunos de ellos han alcanzado tal difusión debido a los avances y mejoras que se les han ido generando por parte de los proveedores de los mismos, o bien por la facilidad de aprender y utilizar el lenguaje. De esta manera podemos

encontrar lenguajes de programación que han ido evolucionando a través de distintas versiones o bien que incluso se han generado nuevos lenguajes de programación basados en aquellos nativos de las primeras generaciones.

En el paradigma de programación imperativa se encuentra la mayor cantidad de lenguajes para el desarrollo de *software* esto se debe en gran parte a que este estilo de programación es el más antiguo y aplicado incluso en la actualidad ya sea directa o indirectamente por lenguajes puros de éste paradigma o que se basan en él. De esta manera podemos encontrar lenguajes como C, Algol, Pascal y Cobol entre sus principales representantes.

Dentro de la programación funcional se puede decir que existen dos grandes categorías de lenguajes de programación: por un lado se tiene a los lenguajes funcionales puros que conservan una mayor potencia expresiva y que por lo tanto conservan su transparencia referencial; por el otro lado, se encuentran los lenguajes funcionales híbridos que son menos estrictos que los puros ya que admiten conceptos tomados de la programación imperativa, como son las secuencias de instrucciones o la asignación de variables, perdiendo a su vez, la transparencia referencial que caracteriza a los lenguajes puros.

Entre los lenguajes funcionales puros, cabe destacar a Haskell y Miranda. Los lenguajes funcionales híbridos más conocidos son Lisp, Scheme, Ocaml y Standar ML (estos dos últimos, descendientes del lenguaje ML).

El paradigma de programación lógica es tal vez el que carece de más variedad entre sus lenguajes de programación representativos, Prolog es el más conocido y difundido para la creación de sistemas basados en este paradigma.

La programación orientada a objetos ha tomado auge en las últimas generaciones de los lenguajes de programación pues le permite al programador escribir código reutilizable y ampliable que opera imitando al mundo real, permitiendo por lo tanto que los programadores utilicen su intuición natural con respecto al mundo para comprender el comportamiento de un programa y construir código apropiado.

Entre los primeros lenguajes de programación orientados a objetos se encuentran el Simula 67 y Algol 60; en los años 80's, el más popular fue Smalltalk y actualmente lo son Java y C++.

BIBLIOGRAFÍA

- A. Tucker, R. Noonan, *"Lenguajes de programación, principios y paradigmas",* Mc Graw Hill, España 2003.
- Kenneth C. Louden, *"Lenguajes de programación, principios y práctica",* Ed. Thomson, 2ª Edición, México 2004.
- Levine Guillermo, *"Computación y programación moderna. Perspectiva integral de la informática"*, Ed. Addison Wesley, México 2001.
- Alonso, *"Metodología de la programación"*, Ed. Paraninfo, S.A., España 1992.
- Peter Van Roy y Seif Haridi, *"Concepts, techniques and models of computer programming"*, MIT (Massachusetts Institute of Technology), Inglaterra 2004.
- Robert. W. Sebesta, "*Concepts of programming languages"*, Ed. Addison Wesley, 6a edición, Estados Unidos 2003.
- Alonso, F.; Martínez, L.; Segovia, J. *"Introducción a la Ingeniería del Software. Modelos de Desarrollo de Programas"*. DELTA Publicaciones, 2005.
- Alonso, F.; Frutos, S.; Martinez, L.; Montes, C. *"Towards a Natural Agent Paradigm Development Methodology"*. Lecture Notes in Computer Science, Springer-Verlag, 2004.
- Watt, D.A. *"Programming Language, Concepts and Paradigms"*. Prentice Hall Int., London, 1990.

PÁGINAS WEB CONSULTADAS

http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n
http://www.um.es/docencia/barzana/IAGP/Iagp3.html
http://www.cristalab.com/tutoriales/101/fundamentos-de-la-programacion
http://www.frt.utn.edu.ar/sistemas/paradigmas/
http://www.rena.edu.ve/cuartaEtapa/Informatica/Tema13.html
http://www.infor.uva.es/~jvegas/cursos/prog/tema1.html
http://www.cibercalli.com/erick/hackingnews/historia-de-los-lenguajes-de-programacion
http://java.sun.com
http://www.afm.sbu.ac.uk/logic-prog/
http://www.visual-prolog.com
http://www.cs.nott.ac.uk/~gmh//faq.html
http://pauillac.inria.fr/caml/
http://wilucha.com.ar/Paradigma/A_Paradigma.html
http://www.dccia.ua.es/dccia/inf/asignaturas/LPP/2006-2007/tema-01.html
http://www.wikilearning.com/metodologias_usadas_en_ingenieria_del_software-wkc-3618.htm
http://www.rescomp.berkeley.edu/~hossman/cs263/paper.html
http://www.rena.edu.ve/cuartaEtapa/Informatica/Tema12.html
http://www.frt.utn.edu.ar/sistemas/paradigmas/index.html
http://www.frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm%20/%20_Lenguajes_de_Programación
http://www.frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm%20/%20_Generaciones_de_los
http://labsopa.dis.ulpgc.es/ada/
http://www.smalltalk.org/
http://www.java.org/
http://www.masternet.com.co/prod/delphi.htm
http://www.frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm%20/%20_C_Más_Más
http://www.frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm%20/%20_Pascal
http://www.lisp.org/
http://www.prolog.org/
http://www.perl.org/
http://www.php.org/
http://www.sql.org/
http://www.wilucha.com.ar/Paradigma/A_Paradigma.html