Paradigmas de Programación

Paradigmas

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 técnicas para corregir los datos recopilados.

Los paradigmas establecen límites, para resolver problemas dentro de estos, y de esta forma mejorar o proporcionar nuevas soluciones, ya que estos filtran experiencias, se ajustan a los límites, percepciones o creencias, lo que se denomina efecto paradigma.

Paradigmas de programación

Un paradigma es una forma de representar y manipular el conocimiento. Representan un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

El paradigma imperativo es considerado el más común y está representado, por ejemplo, por el C o por BASIC.

El paradigma funcional está representado por la familia de lenguajes LISP, en particular Scheme.

El paradigma lógico, un ejemplo es PROLOG.

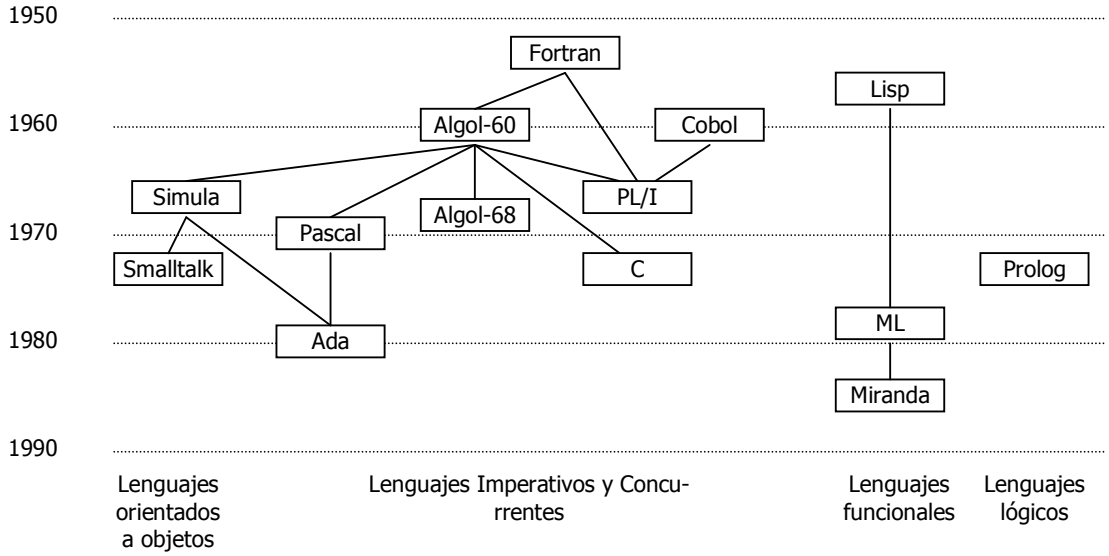
El paradigma orientado a objetos. Un lenguaje completamente orientado a objetos es Smalltalk.

Un poco de historia

Los lenguajes de programación de nuestros días fueron desarrollados a principios de los años 50. Numerosos conceptos fueron inventados, probados e incorporados a distintos lenguajes.

Con algunas pocas excepciones, el diseño de cada lenguaje tuvo una fuerte influencia de experiencias tomadas de lenguajes anteriores.

El gráfico siguiente nos muestra una cronología de cómo fueron desarrollándose los lenguajes actuales y nos demuestra que los mismos no son un producto terminado sino que nuevos y revolucionarios conceptos y paradigmas están siendo desarrollados.



Los primeros lenguajes de alto nivel fueron el Fortran y el COBOL. Fortran introdujo las expresiones simbólicas y subprogramas con parámetros y COBOL introdujo el concepto de descripción de datos.

El primer lenguaje diseñado para algoritmos de comunicación más que sólo para la programación de computadoras fue el Algol-60. Introdujo el concepto de estructura de bloques, variables, procedimientos, etc. que podían se declarados en cualquier lugar del programa donde fueran necesarios.

Influenció a numerosos lenguajes posteriores tan fuertemente que fueron llamados lenguajes estilo Algol.

Así como Fortran y Algol-60 fueron más usados para procesamiento numérico y el COBOL para procesamiento de datos comerciales. PL/I fue un intento de diseñar un lenguaje de propósito general mezclando características de estos tres lenguajes antecesores. El lenguaje resultado fue muy complejo, incoherente y difícil de implementar.

Fue el Pascal, el lenguajes estilo Algol que se hizo más popular porque es simple, sistemático e implementable eficientemente. Pascal y Algol-68 fueron los primeros lenguajes con una rica estructura de control, ricos tipos de datos y definiciones de tipos.

Un sucesor poderoso del Pascal: el Ada, introdujo los conceptos de paquetes y genéricos, diseñados para ayudar a la construcción de grandes programas modulares.

Ada también fue un intento de sus diseñadores de construir el lenguaje de propósito general estándar.

Ciertas tendencias pueden discerniese en la historia de los lenguajes de programación. Una de las tendencias es lograr cada vez un nivel de abstracción más alto.

Los nemónicos y etiquetas del lenguaje ensamblador es una abstracción de los códigos de operación y dirección de máquina.

- Las variables y la asignación es una abstracción del almacenar o recuperar un dato de un almacenamiento de memoria.
- Las estructuras de datos son una abstracción de las estructuras de almacenamiento.
- Las estructuras de control son una abstracción de los saltos (jumps).
- Los procedimientos son una abstracción de las subrutinas.
- Las estructuras de bloques y módulos son una forma de encapsulación que ayuda a desarrollar programas modulares.

Otra tendencia fue la proliferación de paradigmas de programación. Todos los lenguajes mencionados son lenguajes de programación imperativos caracterizados por el uso de comandos que actualizan variables.

El paradigma de programación imperativo es aún el dominante, aunque otros paradigmas están rápidamente ganando popularidad.

Smalltalk está basado en clases de objetos, un objeto viene a ser una variable que puede accederse sólo a través de operaciones asociadas con él. Smalltalk es un ejemplo de un lenguaje orientado a objetos, el cual es un lenguaje imperativo altamente estructurado donde el programa completo está constituido a partir de tales clases de objetos.

La programación orientada a objetos deriva de conceptos introducidos por Simula, otro lenguaje del estilo Algol.

Lisp en su forma pura, está basado enteramente en funciones sobre listas y árboles. Lisp fue el antecesor de los lenguajes de programación funcional. ML y Miranda son lenguajes funcionales modernos, tratan a las funciones como valores de primera clase e incorporan también un sistema de tipos avanzado.

Hasta ahora la notación matemática en su completa generalidad no está implementada. Sin embargo algunos diseñadores de lenguajes quisieron explotar un subconjunto de la notación matemática en los lenguajes de programación. Un lenguaje de programación lógica está basado en un subconjunto de la lógica matemática. La computadora es programada para inferir relaciones entre valores más que calcular valores de salida a partir de valores de entrada. Prolog hizo a la programación lógica popular y es más bien débil e ineficiente, sin embargo fue modificado para agregarle características no lógicas para hacerlo más usable como lenguaje de programación.

Programación Imperativa

La **programación imperativa**, en contraposición a la programación declarativa es un paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea. La implementación de hardware de la mayoría de computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar código de máquina, que es nativo al computador, escrito en una forma imperativa. Esto se debe a que el hardware de los computadores implementa el paradigma de las Máquinas de Turing. Desde esta perspectiva de bajo nivel, el estilo del programa está definido por los contenidos de la memoria, y las sentencias son instrucciones en el lenguaje de máquina nativo del computador (por ejemplo el lenguaje ensamblador). Los lenguajes imperativos de alto nivel usan variables y sentencias más complejas, pero aún siguen el mismo paradigma. Las recetas y las listas de revisión de procesos, a pesar de no ser programas de computadora, son también conceptos familiares similares en estilo a la programación imperativa; cada paso es una instrucción.

La programación imperativa recibe este nombre porque está basada en comandos que actualizan variables contenidas en almacenamientos. Este paradigma tienen una historia relativamente larga, porque los diseñadores de lenguajes de los principios de los años 50 se dieron cuenta que las variables y los comandos de asignación tienen una relación muy cercana con la arquitectura de máquinas por lo que los lenguajes imperativos pueden ser implementados eficientemente.

Sin embargo, hay una razón fundamental de la importancia del paradigma imperativo y tiene que ver con la estrecha relación que tiene para modelar el mundo real. Los programas se escriben para modelar procesos del mundo real que afectan a objetos del mundo real, y tales objetos a menudo poseen un estado que varía con el tiempo. Las variables modelan tales objetos y los programas imperativos tales procesos.

Durante los primeros años de la Informática, la programación se llevaba a cabo en lenguajes y mediante programas estrictamente procedimentales. Un programa que sigue un paradigma imperativo (como el BASIC, PASCAL, COBOL etc.) se caracteriza esencialmente porque las instrucciones que lo componen se ejecutan secuencialmente en un orden preestablecido.

En la programación procedimental, la unidad base de ejecución es el *programa* o conjunto de instrucciones ejecutables, que se divide en una serie de módulos o rutinas distribuidos de acuerdo con una organización jerárquica. En este tipo de programación los datos están desorganizados, son meros apéndices de los programas y tan solo proporcionan valores que sirven para realizar cálculos, o, en el mejor de los casos, para decidir la secuencia de ejecución de las instrucciones del programa.

La instrucción fundamental de la programación procedimental, la que permite construir la jerarquía de programas en que se basa este tipo de programación, es la instrucción CALL. Una aplicación determinada suele estar formada por una jerarquía más o menos implícita de programas, módulos o subrutinas, cada uno de los cuales es capaz de invocar a otros programas de la jerarquía. Uno de los programas situados en la raíz de la jerarquía recibe el nombre de *programa principal* y los demás se conocen con el nombre de *subprogramas, subrutinas, funciones o procedimientos*. En cuanto a los datos, dependiendo del lenguaje que se trate pueden ser *globales*, accesibles por todos los programas de la jerarquía, o *locales*, utilizables solo por el programa al que pertenece, y a veces por aquellos que se encuentran en niveles jerárquicos inferiores a este.

Dentro de este tipo de programación podemos encontrar dos divisiones, las cuales se conocen como *programación estructurada* o programación sin GOTO, que presenta bastantes ventajas en cuanto a la facilidad de modularización, documentación y mantenimiento de los programas; y por otro lado la *programación no estructurada* que permiten realizar saltos incondicionales en la ejecución del programa mediante instrucciones GOTO.

Este paradigma es el más viejo pero aún es el dominante.

Programación Funcional

Sus orígenes provienen del Cálculo Lambda (o λ-cálculo), una teoría matemática elaborada por Alonzo Church como apoyo a sus estudios sobre computabilidad. Un lenguaje funcional es, a grandes rasgos, un azúcar sintáctico del Cálculo Lambda.

El objetivo es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitando el concepto de estado del cómputo. La secuencia de computaciones llevadas a cabo por el programa se regiría única y exclusivamente por la reescritura de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan definiciones dirigidas.

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos laterales.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes procedimentales, como las secuencias de instrucciones o la asignación de variables. En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

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

Programación Lógica

La programación lógica es un paradigma de programación basado en la lógica.

Los programas escritos en un lenguaje lógico están construidos únicamente por expresiones lógicas, es decir, que son ciertas o falsas, en oposición a un expresión interrogativa (una pregunta) o expresiones imperativas (una orden). No obstante, a veces para obtener resultados de un programa lógico sí se usan consultas, de manera que el programa dirá cierto o falso según corresponda (en lógica bivalente).

La mayoría de estos lenguajes se basan en la lógica de predicados, de forma que los programas consisten en definir propiedades de los objetos del dominio de referencia y relaciones entre ellos a través de fórmulas, las cuales usan normalmente el conector implicación.

Programación Orientada a Eventos

La programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema o que ellos mismos provoquen.

Para entender la programación dirigida por eventos, podemos oponerla a lo que no es: mientras en la programación secuencial es el programador el que define cuál va a ser el flujo del programa, en la programación dirigida por eventos será el propio usuario -o lo que sea que esté accionando el programa- el que dirija el flujo del programa.

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. Cuando alguno de estos eventos tenga lugar, el programa pasará a ejecutar el código del correspondiente manejador de evento. 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.

Programación Orientada a Objetos

La Programación Orientada a Objetos (POO u OOP según siglas en inglés) es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto difiere de los lenguajes procedurales tradicionales, en los que los datos y los procedimientos están separados y sin relación. Estos métodos están pensados para hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

Otra manera en que esto es expresado a menudo, es que la programación orientada a objetos anima al programador a pensar en los programas principalmente en términos de tipos de datos, y en segundo lugar en las operaciones ("métodos") específicas a esos tipos de datos. Los lenguajes procedurales animan al programador a pensar sobre todo en términos de procedimientos, y en segundo lugar en los datos que esos procedimientos manejan.

Los programadores que emplean lenguajes procedurales, escriben funciones y después les pasan datos. Los programadores que emplean lenguajes orientados a objetos definen objetos con datos y métodos y después envían mensajes a los objetos diciendo que realicen esos métodos en sí mismos.

Algunas personas también diferencian la POO sin clases, la cual es llamada a veces programación basada en objetos.

Los conceptos de la programación orientada a objetos tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard del Centro de Cómputo Noruego en Oslo. Según se informa, la historia es que trabajaban en simulaciones de naves, y fueron confundidos por la explosión combinatoria de cómo las diversas cualidades de diversas naves podían afectar unas a las otras. La idea ocurrió para agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamiento. Fueron refinados más tarde en Smalltalk, que fue desarrollado en Simula en Xerox PARC pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "en marcha" en lugar de tener un sistema basado en programas estáticos.

La programación orientada a objetos tomó posición como la metodología de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de C++, una extensión del lenguaje de programación C. Su dominación fue consolidada gracias al auge de las Interfaces gráficas de usuario, para los cuales la programación orientada a objetos está particularmente bien adaptada.

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp, Pascal, y otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y a la capacidad de mantenimiento del código. Los lenguajes orientados a objetos "puros", por otra parte, carecían de las características de las cuales muchos programadores habían venido depender. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos

orientados a objetos, pero permitiendo algunas características procedurales de maneras "seguras". El Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos pero ahora ha sido esencialmente reemplazado por Java, en gran parte debido a la aparición de Internet, para la cual Java tiene características especiales.

Entre los lenguajes orientados a objetos destacan los siguientes:

Smalltalk
Objective-C
C++
Ada 95
Java
Ocaml
Python
Delphi
Lexico (en castellano)
C Sharp
Eiffel
Ruby
ActionScript
Visual Basic.NET
PHP
PowerBuilder

Estos lenguajes de programación son muy avanzados en orientación a objetos.

Al igual que C++ otros lenguajes, como OOCOBOL, OOLISP, OOPROLOG y OOREXX, han sido creados añadiendo extensiones orientadas a objetos a un lenguaje de programación clásico.

Programación Orientada a Aspectos

La Programación Orientada a Aspectos (POA) es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la POA se pueden capturar los diferentes conceptos que componen una aplicación en entidades bien definidas, de manera apropiada en cada uno de los casos y eliminado las dependencias inherentes entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término POA es usado para referirse a varias tecnologías relacionadas -como los métodos adaptativos, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias.

Muchas veces nos encontramos, a la hora de programar, con problemas que no podemos resolver de una manera adecuada con las técnicas habituales usadas en la pogramación procedural o en la orientada a objetos. Con éstas, nos vemos forzados a tomar decisiones de diseño que repercuten de manera importante en el desarrollo de la aplicación y que nos alejan con frecuencia de otras posibilidades. Por otro lado, la implementación de dichas decisiones a menudo implica escribir líneas de código que están distribuidas por toda, o gran parte, de la aplicación para definir la lógica de cierta propiedad o comportamiento del sistema, con las consecuentes dificultades de mantenimiento y desarrollo que ello implica. En inglés este problema se conoce como tangled code, que podríamos traducir como código enredado. El hecho es que hay ciertas decisiones de diseño que son difíciles de capturar con las técnicas antes citadas, debiéndose al hecho de que ciertos problemas no se dejan encapsular de igual forma que los que habitualmente se han resuelto con funciones u objetos. La resolución de éstos supone o bien la utilización de repetidas líneas de código por diferentes componentes del sistema, o bien la superposición dentro de un componente de funcionalidades dispares.

La programación orientada a aspectos, permite, de una manera comprensible y clara, definir nuestras aplicaciones considerando estos problemas. Por aspectos se entiende dichos problemas que afectan a la aplicación de manera horizontal y que la programación orientada a aspectos persigue poder

tenerlos de manera aislada de forma adecuada y comprensible, dándonos la posibilidad de poder construir el sistema componiéndolos junto con el resto de componentes.

La programación orientada a objetos (POO) supuso un gran paso en la ingeniería del software, ya que presentaba un modelo de objetos que parecía encajar de manera adecuada con los problemas reales. La cuestión era saber descomponer de la mejor manera el dominio del problema al que nos enfrentáramos, encapsulando cada concepto en lo que se dio en llamar objetos y haciéndoles interactuar entre ellos, habiéndoles dotado de una serie de propiedades. Surgieron así numerosas metodologías para ayudar en tal proceso de descomposición y aparecieron herramientas que incluso automatizaban parte del proceso. Esto no ha cambiado y se sigue haciendo en el proceso de desarrollo del software. Sin embargo, frecuentemente la relación entre la complejidad de la solución y el problema resuelto hace pensar en la necesidad de un nuevo cambio. Así pues, nos encontramos con muchos problemas donde la POO no es suficiente para capturar de una manera clara todas las propiedades y comportamientos de los que queremos dotar a nuestra aplicación. Así mismo, la programación procedural tampoco nos soluciona el problema.

Entre los objetivos que se ha propuesto la POA están, principalmente, el de separar conceptos y el de minimizar las dependencias entre ellos. Con el primer objetivo se persigue que cada decisión se tome en un lugar concreto y no diseminado por la aplicación. Con el segundo, se pretende desacoplar los distintos elementos que intervienen en un programa. Su consecución implicaría las siguientes ventajas:

Un código menos enmarañado, más natural y más reducido.

Mayor facilidad para razonar sobre los conceptos, ya que están separados y las dependencias entre ellos son mínimas.

Un código más fácil de depurar y más fácil de mantener.

Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.

Se tiene un código más reusable y que se puede acoplar y desacoplar cuando sea necesario.