Paradigma de programação

conjunto de conceitos ou padrões que suportam uma linguagem de programação

> Esta página cita fontes, mas que não cobrem todo o conteúdo.
>
> Saiba mais

Paradigma de programação é um meio de se classificar as linguagens de programação baseado em suas funcionalidades. As linguagens podem ser classificadas em vários paradigmas.

Um **paradigma de programação** fornece e determina a visão que o programador possui sobre a estruturação e execução do programa. Por exemplo, em programação orientada a objetos, os programadores podem abstrair um programa como uma coleção de objetos que interagem entre si, enquanto em programação funcional os programadores abstraem o programa como uma sequência de funções executadas de modo empilhado.

Alguns paradigmas estão preocupados principalmente com as implicações para o modelo de execução da linguagem, como permitir efeitos colaterais, ou se a sequência de operações está definida pelo modelo de execução. Outros paradigmas estão preocupados principalmente com o modo que o código está organizado, como o agrupamento de código em unidades junto com o estado que é modificado pelo código. Outros ainda estão preocupados principalmente com o estilo de sintaxe ou gramática.

Assim como diferentes grupos em engenharia de software propõem diferentes metodologias, diferentes linguagens de programação propõem diferentes *paradigmas de programação*. Algumas linguagens foram desenvolvidas para suportar um paradigma específico (Smalltalk e Java suportam o paradigma de orientação a objetos enquanto Haskell suportam o paradigma funcional), enquanto outras linguagens suportam múltiplos paradigmas (como o LISP, Perl, Python, C++ e Oz).

Os paradigmas de programação são muitas vezes diferenciados pelas técnicas de programação que *proíbem* ou *permitem*. Por exemplo, a programação estruturada não permite o uso de `goto`. Esse é um dos motivos pelo qual novos paradigmas são considerados mais rígidos que estilos tradicionais. Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de conceito de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.

O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem suportar mais de um paradigma.

História

Inicialmente, os computadores eram programados através de código binário, que representava as sequências de controle alimentadas à Unidade Central de Processamento (CPU). Tal processo era difícil e propício a erros; os programas estavam em código de máquina, que é um paradigma de programação de muito baixo nível.

Para facilitar a programação foram desenvolvidas linguagens de montagem ("assembly"). Elas substituíam as funções do código de máquina por mnemônicas, endereços de memória absolutos referenciados por identificadores. A linguagem de montagem ainda é considerada de baixo nível, ainda que seja um paradigma da "segunda geração" das linguagens. Mesmo linguagens de montagem da década de 1960 suportavam gerações condicionais de macros bastante sofisticadas. Também suportavam recursos de programação modular tais como `CALL` (para suportar sub-rotinas), variáveis externas e secções comuns (globais); isso permitia reutilizar código e o isolamento de características específicas do hardware, através do uso de operadores lógicos como `READ`, `WRITE`, `GET` e `PUT`. A linguagem de montagem foi e ainda é usada para sistemas críticos, e frequentemente usada em sistemas embarcados.

O próximo avanço foi o desenvolvimento das linguagens procedimentais. As primeiras a serem descritas como de alto nível, essas linguagens da terceira geração usam um vocabulário relativo ao problema sendo resolvido. Por exemplo, COBOL usa termos como `file` (para identificar arquivos), `move` (para mover arquivos) e `copy` (para copiar arquivos). Tanto FORTRAN quanto ALGOL usam terminologia matemática, tendo sido desenvolvidas principalmente para problemas comerciais ou científicos. Tais linguagens procedurais descrevem, passo a passo, o procedimento a ser seguido para resolver certo problema. A eficácia e a eficiência de cada solução é subjetiva e altamente dependente da experiência, habilidade e criatividade do programador.

Posteriormente, linguagens orientadas a objeto foram criadas. Nelas, os dados e as rotinas para manipulá-los são mantidos numa unidade chamada objeto. O utilizador só pode acessar os dados através das sub-rotinas disponíveis, chamadas *métodos*, o que permite alterar o funcionamento interno do objeto sem afetar o código que o consome. Ainda há controvérsia por programadores notáveis como Alexander Stepanov, Richard Stallman[1] entre outros, questionando a eficácia desse paradigma em comparação do paradigma procedural. A necessidade de cada objeto de ter métodos associados tornaria os programas muito maiores. O conceito de polimorfismo foi desenvolvido como tentativa de solucionar tal dilema. Tendo em vista que a orientação a objeto é um paradigma e não uma linguagem, é possível criar até mesmo uma linguagem de montagem orientada a objeto, como o High Level Assembly.

Independente do ramo das linguagens imperativas, baseadas nas linguagens procedurais, paradigmas de programação declarativa também foram desenvolvidos. Nessa linguagens se descreve o problema ao computador, não como resolvê-lo. O programa é estruturado como uma coleção de propriedades para encontrar o resultado esperado, e não um procedimento a se seguir. Dado um banco de dados ou um conjunto de regras, o computador tenta encontrar a solução ao casar todas as propriedades desejadas. Um exemplo é o SQL, assim como a família das linguagens funcionais e lógicas.

Programas escritos em programação funcional usam funções, blocos de código construídos para agir como funções matemáticas. Desencoraja-se a mudança do valor das variáveis através de atribuição, fazendo grande uso de recursividade para isso.

Na programação lógica, fatos sobre o domínio do problema são expressados como fórmulas lógicas, e os programas são executados ao se aplicar regras de inferência nas fórmulas até que uma resposta é encontrada, ou a coleção de fórmulas é provada inconsistente.

Exemplos

- Programação estruturada, em contraste a Programação orientada a objetos.
- Programação imperativa, em contraste de programação declarativa.
- Programação de passagem de mensagens, em contraste de programação imperativa.
- Programação procedural, em contraste de programação funcional.
- Programação orientada a fluxos, em contraste de programação orientada a eventos.
- Programação escalar, em contraste de programação vetorial.
- Programação restritiva, que complementa a programação lógica.
- Programação orientada a aspecto (como em AspectJ).
- Programação orientada a regras (como em Mathematica).
- Programação orientada a tabelas (como em Microsoft FoxPro).
- Programação orientada a fluxo de dados (como em diagramas).
- Programação orientada a políticas.
- Programação orientada a testes.
- Programação Genérica.
- Programação multiparadigma.

Referências

1. *«Mode inheritance, cloning, hooks & OOP (Google Groups Discussion)» (https://groups.google.com/group/comp.emacs.xemacs/browse_thread/thread/d0af257a2837640c/37f251537fafbb03?lnk=st&q=%22Richard+Stallman%22+oop&rnum=5&hl=en#37f251537fafbb03)*