Pensamento computacional: abstração, requisitos operacionais e paradigmas de programação

Ana Paula Lüdtke Ferreira
Universidade Federal do Pampa
Bagé, Rio Grande do Sul, Brasil
anaferreira@unipampa.edu.br

O termo "Pensamento Computacional", cunhado por Wing em seu artigo seminal [18], tem cerca de 2.500 citações registradas no site da ACM. Conceituado como um conjunto de habilidades e atitudes cujo foco é a solução de problemas, é uma competência necessária para todas as pessoas, independente de área de atuação. O termo é definido como "os processos mentais envolvidos na formulação e na expressão da solução de um problema, de forma que tanto pessoas como máquinas possam resolvê-lo" [19]. A abstração é considerada o meio para a mobilização desses processos, decorrente do reconhecimento e definição de padrões, generalização a partir de instâncias concretas e parametrização para adaptação de soluções a contextos específicos.

O desenvolvimento da capacidade de abstração é um desafio no ensino de programação. Contudo, não é suficiente declarar que um método suporta o desenvolvimento dessa capacidade: os processos envolvidos devem ser explicitados, com indicadores de avaliação claros. Conceitos educacionais sem o aporte de procedimentos operacionais para implementá-los são inócuos [3].

Na escola, a Matemática é o lugar do pensar abstrato, mas costuma ser ensinada de forma puramente operacional, como sinônimo de "fazer contas" e não de linguagem de descrição de modelos e fenômenos. A pouca relação com a vida faz com que o assunto seja considerado "difícil" e pouco interessante, afastando os alunos do tema. Na graduação, o ensino de algoritmos é similar, com com foco na sintaxe das construções de linguagem e não em solução de problemas.

Paradigmas de programação determinam como o programador estrutura a solução de um problema, a partir do modelo matemático que dá suporte às estruturas da linguagem. A facilidade da construção de uma solução deriva da consistência entre a forma de pensar a solução e as construções disponibilizadas [11]. A eficiência da solução também advém do projeto e implementação da linguagem. Por exemplo, a linguagem C [7] permite desenvolver um programa funcional (tipado), embora a ausência de mecanismos de *garbage collection* comprometa a efetividade da solução; um programa concorrente em Java [12] produz um resultado menos eficiente do que uma solução em Erlang [2], que contém construções de concorrência nativas e não dependentes dos mecanismos implementados pelo sistema operacional.

O modelo matemático subjacente a um paradigma estabelece quais são os elementos principais de uma linguagem e as operações daí decorrentes. No paradigma imperativo, a memória determina o estado e o fluxo de execução de um programa e a atribuição é a única operação capaz de alterar seu estado. O modelo de execução imperativo permite efeitos colaterais na execução de subprogramas por meio de atribuição de valores a variáveis globais ou a parâmetros passados por referência ou resultado [14]. Estruturas de encapsulamento de dados e código usualmente existem, mas não têm uso obrigatório, facilitando a escrita de programas pouco estruturados, com blocos de códigos que executam muitas operações diferentes e não relacionadas em sequência. Esse modelo não orienta o aprendiz para o exercício de reconhecimento de padrões e parametrização de soluções, elementos essenciais da abstração.

O paradigma de orientação a objetos favorece o exercício da abstração, visto que a construção de classes envolve pensar sobre padrões, separar dados de suas funcionalidades e evidenciar oportunidades de parametrização de objetos e de seus métodos. A abstração necessária para construção de um modelo de classes/objetos exige o desenvolvimento de habilidade de modelagem, antes da implementação dos métodos. Contudo, o uso de linguagens orientadas a objeto com fluxo de execução imperativo acaba produzindo o mesmo tipo de problema descrito no parágrafo anterior, especialmente quando o foco na codificação (e não na modelagem) é privilegiado [5, 8, 10].

A falta do exercício da abstração prejudica a reutilização de especificações, modelos e código já construídos. O reúso de software exige a composição de elementos conhecidos para a solução de um problema novo. Essa habilidade não pode ser desenvolvida quando os estudantes, a cada nova tarefa, são instados a construir sempre uma solução a partir do zero. A composição é a operação de destaque do paradigma funcional [13, 20]. Construir um programa funcional monolítico é mais difícil do que compor especificações menores. Contudo, linguagens funcionais tem sintaxes pouco adequadas para o ensino, pela variabilidade de seus elementos sintáticos fortemente baseadas em notação matemática [1, 6, 9, 15–17], que não favorecem o pensar sequencial associado às linguagens textuais imperativas, usadas na maior parte das disciplinas dos cursos de graduação e na indústria de software.

O objetivo deste trabalho é discutir as relações existentes entre as habilidades do pensamento computacional e dos modelos subjacentes aos paradigmas de programação, com o intuito de fomentar propostas de linguagens adequadas ao ensino que assegurem – e não somente possibilitem – o desenvolvimento das competências de solução de problemas dentro do arcabouço ao Pensamento Computacional.

EduComp'22, Abril 24–29, 2022, Feira de Santana, Bahia, Brasil (On-line)

REFERÊNCIAS

[1] Ulisses Almeida. 2018. *Learn Functional Programming with Elixir – New Foundations for a New World.* The Pragmatic Bookshelf. 198 pages.

[2] Joe Armstrong. 2013. *Programming ERLANG: Software for a Concurrent World* (2 ed.). Pragmatic Bookshelf. 548 pages.

[3] José Sérgio Carvalho. 2001. O discurso pedagógico das diretrizes curriculares nacionais: competência crítica e interdisciplinaridade. *Cadernos de Pesquisa* 12 (Março 2001), 155–165.

[4] Marina Silva da Silva and Ana Paula Lüdtke Ferreira. 2021. Sintaxe baseada em gramáticas de grafos para uma linguagem funcional visual voltada ao aprendizado de programação. In *VI Workshop-Escola de Informática Teórica (WEIT 2021).*

[5] Anabela Gomes and Antonio José Mendes. 2007. Learning to program - difficulties and solutions *(International Conference on Engineering Education – ICEE 2007).*

[6] Sonya E. Keene. 1989. *Object-Oriented Programming in Common Lisp.* Addison-Wesley. http://cl-cookbook.sourceforge.net/clos-tutorial/

[7] Brian W. Kernighan and Dennis Ritchie. 1988. *The C Programming Language* (2 ed.). Prentice Hall. 274 pages.

[8] Wanda M. Kunkle and Robert B. Allen. 2016. The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts. *ACM Transactions on Computing Education* 16 (February 2016), 3(1–26). Issue 1. https://doi.org/10.1145/2785807

[9] Martin Odersky, Lex Spoon, and Bill Venners. 2008. *Programming in Scala* (4th ed.). Artima. 776 pages.

[10] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin* 39 (December 2007), 204–223. Issue 4.

[11] Peter Van Roy, Joe Armstrong, Matthew Flatt, and Boris Magnusson. 2003. The Role of Language Paradigms in Teaching Programming. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education.* Reno, Nevada.

[12] Herbert Schildt. 2021. *Java: The Complete Reference* (12 ed.). McGraw Hill. 1280 pages.

[13] Robert W. Sebesta. 2018. *Conceitos de linguagens de programação* (11 ed.). Bookman, Porto Alegre. 765 pages.

[14] Ravi Sethi. 1996. *Programming languages: concepts and constructs* (2 ed.). Addison-Wesley Publishing, Reading, MA. 640 pages.

[15] Simon Thompson. 1999. *Haskell: The Craft of Functional Programming* (2 ed.). Addison-Wesley. 507 pages. https://www.haskell.org/

[16] David S. Touretzky. 1990. *COMMON LISP: A Gentle Introduction to Symbolic Computation.* The Benjamin/Cummings Publishing Company, Inc., Redwood City. 587 pages. http://www.inf.ufsc.br/~aldo.vw/func/touretzky/touretzky.pdf

[17] Jeffrey D. Ullman. 1998. *Elements of ML Programming.* Prentice Hall, Englewood Cliffs, New Jersey. 383 pages.

[18] Jeannette M. Wing. 2008. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society* (2008).

[19] Jeannette M. Wing. 2014. Computational thinking benefits society.

[20] Weixin Zhang, Yaozhu Sun, and Bruno C. D. S. Oliveira. 2021. Compositional Programming. *ACM Trans. Program. Lang. Syst.* 43, 3, Article 9 (Sept. 2021), 61 pages. https://doi.org/10.1145/3460228