Além da Corretude: Investigando os Limites da Correção Automática ao Analisar Códigos Corretos

Eryck Pedro da Silva[1], **Ricardo Caceffo**[2], **Rodolfo Azevedo**[1]

[1]Universidade Estadual de Campinas (UNICAMP)

[2]Universidade Virtual do Estado de São Paulo (Univesp)

{eryck.silva,rodolfo}@ic.unicamp.br, ricardo.caceffo@univesp.br

A integração de componentes curriculares de Ciência da Computação em diferentes cursos de ensino superior tem sido tradicionalmente justificada pela necessidade de formar uma nova geração de profissionais com habilidades voltadas não somente à programação, como também à aplicação da computação em campos interdisciplinares [Blikstein and Moghadam 2019, Guzdial and du Boulay 2019]. Esse contexto tem motivado a inclusão de disciplinas de Ciência da Computação em diversos níveis educacionais, estendendo-se para alunos com históricos e interesses não necessariamente alinhados com áreas de Ciências, Tecnologia, Engenharia e Matemática (STEM).

Dentro do contexto dos cursos de graduação, a disciplina de Introdução à Programação (*Computer Science* 1 – CS1) é, em geral, uma das primeiras com as quais os estudantes têm contato [Austing et al. 1979, Hertz 2010]. Embora não haja consenso completo sobre os tópicos que devem compor seu conteúdo programático [Becker and Fitzpatrick 2019, Hertz 2010, Silva et al. 2023b], seu principal objetivo é desenvolver o raciocínio lógico e sistemático dos discentes. No entanto, disciplinas de CS1 estão frequentemente associadas a desafios [Bosse and Gerosa 2015, Walker 2017], entre os quais pode-se listar a necessidade de aplicar avaliações concisas e coerentes para garantir um aprendizado efetivo [Bloom et al. 1971]. Essa questão é particularmente relevante, uma vez que as atividades avaliativas desempenham um papel central no alcance dos objetivos de aprendizagem [Black and William 1998, Lancaster et al. 2019].

Uma das abordagens tradicionalmente adotadas para promover o aprendizado tem sido o aumento do número de avaliações, com o intuito de proporcionar maior prática aos discentes. Com base nisso, sistemas de correção automática (também conhecidos como *autograders* ou Juízes Online) [Hollingsworth 1960] foram desenvolvidos para reduzir o tempo e o esforço despendidos pelos docentes. Atualmente, em CS1, *autograders* são utilizados no envio e na correção de tarefas submetidas pelos estudantes, baseando-se predominantemente na verificação da saída gerada por um programa em relação a um conjunto predefinido de entradas [Prather et al. 2018], denominado *casos de teste*.

À primeira vista, avaliar um programa com base em sua passagem nos casos de teste pode parecer suficiente. No entanto, estudantes de CS1 frequentemente elaboram códigos que apresentam características normalmente evitadas por programadores experientes, mesmo quando o programa gera a saída correta [De Ruvo et al. 2018, Keuning et al. 2021]. Exemplos incluem aumentos redundantes na complexidade devido ao uso inadequado de estruturas de decisão ou laços de repetição [Ihantola and Petersen 2019, Silva et al. 2021, Silva et al. 2023a]. Dessa forma, o uso exclusivo de *autograders* com foco na corretude gera o risco de que discentes sejam apro-

vados em CS1 sem uma compreensão adequada dos conceitos ensinados.

À luz do exposto, este estudo identificou e classificou Problemas de Compreensão em Códigos Corretos (PC³) [Silva 2024], que são padrões de codificação presentes em códigos considerados corretos por um *autograder*, que potencialmente indicam que o discente não está compreendendo totalmente algum conceito abordado em CS1. A nomenclatura foi baseada nos estudos de problemas de compreensão (*misconceptions*) [Qian and Lehman 2017] que, em CS1, têm como objetivo identificar e esclarecer erros comuns em níveis sintático, semântico ou lógico [Caceffo et al. 2016, Kaczmarczyk et al. 2010], mas não necessariamente se limitam a códigos corretos.

O processo de identificação e catalogação dos PC³ foi realizado por meio da análise inicial de 2.441 códigos em Python submetidos por estudantes das turmas da disciplina Algoritmos e Programação de Computadores (MC102) da Universidade Estadual de Campinas (UNICAMP). No total, foram identificados 45 PC³, os quais foram organizados em oito categorias: A) Variáveis, identificadores e escopo (8); B) Expressões booleanas (12); C) Iteração (8); D) Parâmetros de função e escopo (4); E) Raciocínio (2); F) Casos de teste (2); G) Organização de código (6) e H) Outros (3). Cada PC³ foi representado por um identificador composto pela letra de sua respectiva categoria seguida de um número. O Código 1 exemplifica um caso em que o estudante, acreditando que a cláusula `else` era obrigatória, incluiu-a com um bloco de código redundante.

```
1 var = int(input())
2 if var >= 0:
3     print("positivo␣ou␣0")
4 elif var < 0:
5     print("negativo")
6 else:
7     (...)
```

Código 1. Exemplo de `else` desnecessário.

```
1 var = int(input())
2 if var >= 0:
3     print("positivo␣ou␣0")
4 else:
5     print("negativo")
```

Código 2. Código 1 sem o `else` desnecessário.

Os passos da metodologia que compôs este estudo estão representados, em ordem cronológica, na Figura 1. O estudo foi estruturado a partir de métodos que buscaram validar os PC³ tanto de forma externa (considerando outras disciplinas de CS1) quanto interna (no contexto específico de MC102). A validação externa foi conduzida por meio da análise de 225 ementas de disciplinas de CS1 em nível nacional, as quais foram comparadas com ementas internacionais. Essa etapa permitiu verificar que os tópicos mais abordados estavam alinhados às categorias de PC³. Além disso, 32 docentes de CS1 responderam um questionário e participaram de entrevistas para avaliar os 45 PC³ iniciais e indicar quais seriam os mais graves, isto é, aqueles que demandariam maior atenção na correção, resultando em uma lista de 15 PC³. Os docentes entrevistados também destacaram que os PC³ poderiam ter origens distintas, sendo decorrentes tanto da desatenção (e.g., PC³ A4 – Redefinição de *built-in*) quanto de concepções equivocadas dos conceitos (e.g., PC³ C1 – Condição `while` testada novamente em seu interior).

A validação interna, por sua vez, complementou a análise das razões de ocorrência dos PC³ por meio da observação em sala de aula, envolvendo 20 discentes e um docente de MC102. Os resultados indicaram que os PC³ mais graves das categorias A, D, G e H estavam relacionados à desatenção ou, em alguns casos, foram deliberadamente cometi-

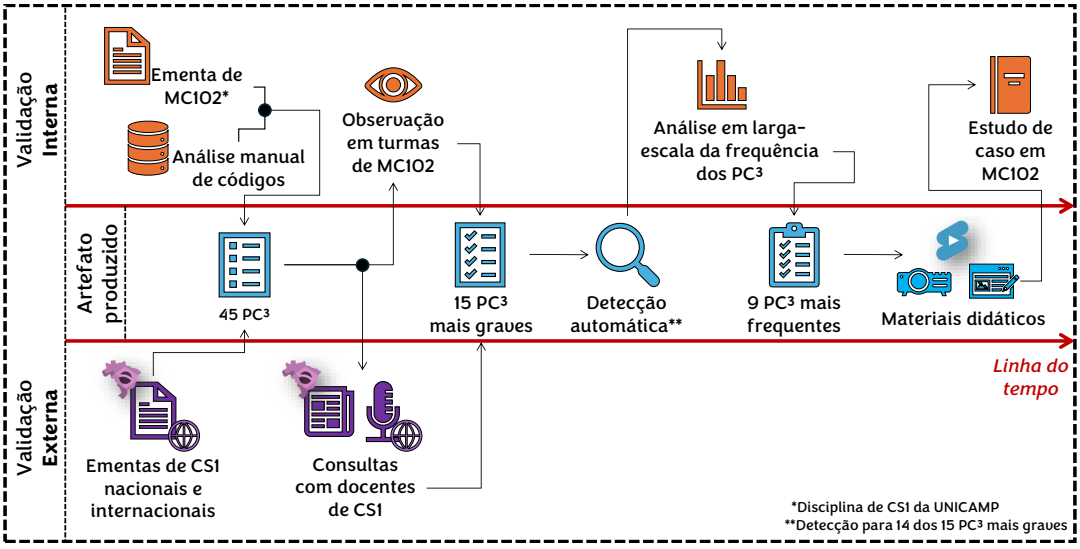

Figura 1. Metodologia empregada na pesquisa em ordem cronológica.

dos por estudantes que buscavam diferenciar seus códigos, receosos de que o *autograder* os classificasse como plágio. Em contraste, os PC³ das categorias B e C tiveram suas ocorrências associadas a compreensões erradas. Esse aspecto representa um desafio significativo, dado que expressões booleanas e iteração são conceitos fundamentais em CS1. Para complementar, o desenvolvimento de *script*[1] para a detecção automática de 14 PC³ mais graves viabilizou uma análise em larga escala de mais de 40 mil códigos submetidos ao longo de oito semestres de MC102. A aplicação do teste de Kruskal-Wallis revelou que a frequência de ocorrência dos PC³ entre a primeira e a segunda metade da disciplina não apresentou uma redução estatisticamente significativa, reforçando a hipótese de que os discentes podem concluir a disciplina ainda mantendo compreensões inadequadas.

Por fim, materiais didáticos[1] específicos foram desenvolvidos e avaliados por meio de um estudo de caso em uma turma de MC102. Ao todo, participaram 23 discentes, divididos em dois grupos distintos, A e B, que foram expostos a esses materiais antes da realização das respectivas atividades práticas. Ao término do estudo, a incidência de PC³ foi quantificada e comparada entre os alunos que alegaram ter visto os materiais com aqueles que não viram. A aplicação do teste Chi-Quadrado revelou que os códigos dos discentes do grupo A apresentaram uma redução na ocorrência de PC³ quando expostos aos materiais didáticos. Em contrapartida, os códigos dos alunos do grupo B que acessaram os materiais apresentaram mais PC³ em comparação àqueles que não os viram. No entanto, somente foi possível rejeitar a hipótese nula para o grupo A.

Os resultados sugerem que, embora determinados PC³ possam ser atenuados por meio de materiais didáticos, estratégias complementares devem ser empregadas em sala de aula para garantir um aprendizado compreensivo. A ocorrência dos PC³ pode ser uma visão de esforço extra desnecessário [Wigfield and Eccles 2000] para programar ou ainda resultante de uma linha de raciocínio rápido e intuitivo [Robins 2022]. Dessa forma, docentes e instrutores deveriam incentivar metodologias que promovam a importância dos discentes se preocuparem com demais características de seus códigos além da corretude.

[1] `https://linktr.ee/materiaispc3`

Agradecimentos

Gostaríamos de agradecer aos 32 docentes e 56 discentes que se voluntariaram para os estudos conduzidos. A pesquisa foi financiada pelo Fundo de Apoio ao Ensino, Pesquisa e Extensão (FAEPEX) sob os números 38813-20 e 69086-24 e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) sob o número 142476/2020-0.

Referências

Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L., and Stokes, G. (1979). Curriculum '78: Recommendations for the Undergraduate Program in Computer Science—a Report of the ACM Curriculum Committee on Computer Science. *Commun. ACM*, 22(3):147–166.

Becker, B. A. and Fitzpatrick, T. (2019). What do cs1 syllabi reveal about our expectations of introductory programming students? In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 1011–1017, New York, NY, USA. Association for Computing Machinery.

Black, P. and William, D. (1998). Assessment and classroom learning. *Assessment in Education: Principles, Policy & Practice*, 5(1):7–74.

Blikstein, P. and Moghadam, S. H. (2019). *Computing Education Literature Review and Voices from the Field*, page 56–78. Cambridge Handbooks in Psychology. Cambridge University Press.

Bloom, B., Hastings, J., and Madaus, G. (1971). *Handbook on Formative and Summative Evaluation of Student Learning*. McGraw-Hill.

Bosse, Y. and Gerosa, M. (2015). Reprovações e Trancamentos nas Disciplinas de Introdução à Programação da Universidade de São Paulo: Um Estudo Preliminar. In *Anais do XXIII Workshop sobre Educação em Computação*, pages 426–435, Porto Alegre, RS, Brasil. SBC.

Caceffo, R., Wolfman, S., Booth, K. S., and Azevedo, R. (2016). Developing a Computer Science Concept Inventory for Introductory Programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, page 364–369, New York, NY, USA. Association for Computing Machinery.

De Ruvo, G., Tempero, E., Luxton-Reilly, A., Rowe, G. B., and Giacaman, N. (2018). Understanding semantic style by analysing student code. In *Proceedings of the 20th Australasian Computing Education Conference*, ACE '18, page 73–82, New York, NY, USA. Association for Computing Machinery.

Guzdial, M. and du Boulay, B. (2019). *The History of Computing Education Research*, page 11–39. Cambridge Handbooks in Psychology. Cambridge University Press.

Hertz, M. (2010). What Do "CS1"and "CS2"Mean? Investigating Differences in the Early Courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, page 199–203, New York, NY, USA. Association for Computing Machinery.

Hollingsworth, J. (1960). Automatic graders for programming classes. *Commun. ACM*, 3(10):528–529.

Ihantola, P. and Petersen, A. (2019). Code complexity in introductory programming courses. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, pages 7662–7670.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., and Herman, G. L. (2010). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, page 107–111, New York, NY, USA. Association for Computing Machinery.

Keuning, H., Heeren, B., and Jeuring, J. (2021). A tutoring system to learn code refactoring. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, page 562–568, New York, NY, USA. Association for Computing Machinery.

Lancaster, T., Robins, A. V., and Fincher, S. A. (2019). *Assessment and Plagiarism*, page 414–444. Cambridge Handbooks in Psychology. Cambridge University Press.

Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., and Cohen, M. (2018). Metacognitive difficulties faced by novice programmers in automated assessment tools. *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 41–50.

Qian, Y. and Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE).*, 18(1).

Robins, A. V. (2022). Dual process theories: Computing cognition in context. *ACM Trans. Comput. Educ.*, 22(4).

Silva, E. (2024). *Misconceptions in Correct Code: Assisting Instructors and Students by Shedding Light on What Is Potentially Overshadowed by Automated Correction*. Tese (doutorado), Universidade Estadual de Campinas (UNICAMP), Instituto de Computação, Campinas, SP, Brazil. Disponível em: `https://hdl.handle.net/20.500.12733/23209`. Acesso em: 21 mar. 2025.

Silva, E., Caceffo, R., and Azevedo, R. (2021). Análise estática de código em conjunto com autograders. In *Anais Estendidos do I Simpósio Brasileiro de Educação em Computação*, pages 25–26, Porto Alegre, RS, Brasil. SBC.

Silva, E., Caceffo, R., and Azevedo, R. (2023a). Misconceptions in Correct Code: rating the severity of undesirable programming behaviors in Python CS1 courses. Technical Report IC-23-01, Institute of Computing, University of Campinas.

Silva, E., Caceffo, R., and Azevedo, R. (2023b). A syllabi analysis of cs1 courses from brazilian public universities. *Brazilian Journal of Computers in Education*, 31(1):407–436.

Walker, H. M. (2017). ACM RETENTION COMMITTEE Retention of Students in Introductory Computing Courses: Curricular Issues and Approaches. *ACM Inroads*, 8(4):14–16.

Wigfield, A. and Eccles, J. S. (2000). Expectancy–Value Theory of Achievement Motivation. *Contemporary Educational Psychology*, 25(1):68–81.