Análise Estática de Código em Conjunto com Autograders

Eryck Silva
eryck.silva@ic.unicamp.br
Universidade Estadual de Campinas
Campinas, São Paulo, Brasil

Ricardo Caceffo
caceffo@ic.unicamp.br
Universidade Estadual de Campinas
Campinas, São Paulo, Brasil

Rodolfo Azevedo
rodolfo@ic.unicamp.br
Universidade Estadual de Campinas
Campinas, São Paulo, Brasil

A disseminação da computação como ferramenta para as mais diversas áreas do conhecimento soma novos desafios de aprendizagem aos já existentes nas disciplinas iniciais de programação (CS1) das universidades. Em especial, três fatores são relevantes a este trabalho: 1) índices de evasão e retenção [3, 17]; 2) Aprendizagem Ativa (*Active Learning*) [2]; e 3) *Autograders* [14].

Fatores como o grande número de alunos por turma, expectativas equivocadas com o conteúdo da disciplina e o contexto educacional prévio desses discentes contribuem para os índices de evasão e retenção das disciplinas de CS1 [17]. Além disso, Anwar [1] menciona que a presença de características tradicionais de ensino que as disciplinas utilizam, permeando uma posição passiva dos alunos no processo de aprendizado, também influenciam nessas taxas.

Como metodologia alternativa à abordagem tradicional, a Aprendizagem Ativa promove atividades que estimulem a discussão, análise, síntese e avaliação, elevando o engajamento dos alunos. Segundo Bonwell e Eison [2], estudos colaborativos, debates, simulações e tutoria em pares [5] são exemplos do uso dessa metodologia.

Autograders são sistemas de avaliação automática de programas normalmente empregados para auxiliar a correção de exercícios [14], reduzindo o tempo que o docente normalmente levaria para fazê-lo [7, 8]. No entanto, seu uso também está sujeito a falhas: um mal funcionamento dessa ferramenta pode gerar desconforto nos estudantes, que estão dependentes da mesma para enviar trabalhos acadêmicos [11]. Uma outra questão associada ao uso desses sistemas é o fato do docente querer avaliar se o aluno aprendeu os conceitos ensinados, verificando diretamente seus usos nas atividades práticas. Considerando que dois programas podem ter a mesma saída sendo escritos de formas diferentes, eles estariam *corretos* na avaliação de um *autograder* comum, mas se o discente não estiver aplicando tudo que está aprendendo na resolução dos exercícios, pode significar alguma deficiência no aprendizado.

O estudo de problemas de compreensão (*misconceptions*) permite montar um Inventário de Conceitos (*Concept Inventory*) [6], listando os erros comuns que os discentes cometem [4, 15]. Neste trabalho é proposto que, com a verificação desses erros em códigos, *feedback* além do resultado de saída, já realizado por *autograders*, das atividades práticas possa ser informado para ambos docentes e discentes. Também é esperando que, como consequência, o professor possa dedicar mais tempo auxiliando os alunos com dificuldade.

EduComp'21, Abril 26–30, 2021, Jataí, Goiás, Brasil (On-line)

Uma análise inicial está sendo feita identificando problemas de compreensão simples em códigos desenvolvidos por alunos da disciplina MC102[1] da Universidade Estadual de Campinas. Um grande número de alunos por semestre (aproximadamente 600) realizam uma série de laboratórios sobre os diversos tópicos abordados, submetendo suas soluções via sistema *online* de submissão, o SuSy[2].

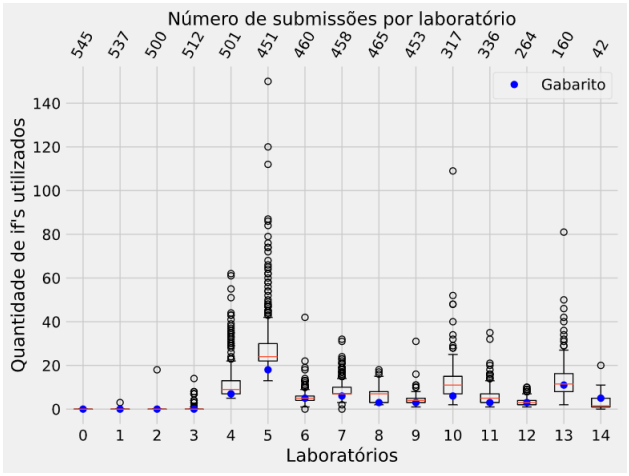

Figura 1: Boxplot da quantidade de *if's* utilizados a cada laboratório das turmas de MC102 em 1s2020. *Outliers* inclusos.

A Figura 1 representa a quantidade de *if's* utilizados em cada programa dos 15 laboratórios do semestre, tendo como objetivo comparar a variação entre cada programa e também com o gabarito. Um exemplo de antipadrão documentado que pode ser analisado é o uso de *if's* aninhados quando se poderia usar operadores lógicos pra obter o mesmo resultado [4]. Os laboratórios 4, 5 e 10 mostram situações de alguns programas com quase sete vezes mais que o gabarito, nesses casos, por mais que a saída possa estar correta, o *autograder* pode informar que não é um padrão incentivado, pois pode gerar códigos difíceis de manter futuramente, por exemplo.

Os próximos passos desta pesquisa incluem a análise e investigação de mais métricas comuns na literatura, como a Complexidade de McCabe [13] e Halstead [9], que possuem formas de avaliar complexidade, dificuldade de escrita e compreensão e até *bugs* estimados [10, 12, 16]. Com a utilização dessas, espera-se identificar programas que precisem de atenção, mesmo que considerados corretos pelo *autograder*, gerando um conjunto de *feedback* a ser empregado com as metodologias de Aprendizagem Ativa.

[1] https://www.ic.unicamp.br/~mc102/
[2] https://www.ic.unicamp.br/~susy/

AGRADECIMENTOS

Agradecemos ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo processo 142476/2020-0.

REFERÊNCIAS

[1] Saira Anwar. 2019. Impact of Educational Technology-Based Learning Environment on Students' Achievement Goals, Motivational Constructs, and Engagement. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 321–322. https://doi.org/10.1145/3291279.3339441

[2] C.C. Bonwell and J.A. Eison. 1991. *Active Learning: Creating Excitement in the Classroom.* Wiley. https://books.google.com.br/books?id=AW7uAAAAMAAJ

[3] Yorah Bosse. 2020. *Padrões de Dificuldades Relacionadas com o Aprendizado de Programação.* Ph.D. Dissertation. Universidade de São Paulo.

[4] Ricardo Caceffo, Breno de França, Guilherme Gama, Raysa Benatti, Tales Aparecida, Tania Caldas, and Rodolfo Azevedo. 2017. An Antipattern Documentation about Misconceptions related to an Introductory Programming Course in C. In *Technical Report 17-15.* Institute of Computing, University of Campinas, 42.

[5] Ricardo Caceffo, Guilherme Gama, and Rodolfo Azevedo. 2018. Exploring Active Learning Approaches to Computer Science Classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 922–927. https://doi.org/10.1145/3159450.3159585

[6] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. 2016. Developing a Computer Science Concept Inventory for Introductory Programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 364–369. https://doi.org/10.1145/2839509.2844559

[7] Margaret Ellis, Clifford A. Shaffer, and Stephen H. Edwards. 2019. Approaches for coordinating etextbooks, online programming practice, automated grading, and more into one course. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (2019), 126–132. https://doi.org/10.1145/3287324.3287487

[8] Leandro Galvão, David Fernandes, and Bruno Gadelha. 2016. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)* 27, 1 (2016), 140. https://doi.org/10.5753/cbie.sbie.2016.140

[9] Maurice H Halstead. 1972. Natural laws controlling algorithm structure? *ACM Sigplan Notices* 7, 2 (1972), 19–26.

[10] Petri Ihantola and Andrew Petersen. 2019. Code complexity in introductory programming courses. In *Proceedings of the 52nd Hawaii International Conference on System Sciences.*

[11] Inside Higher Ed. 2018. Autograder issues upset students at Berkeley. https://www.insidehighered.com/news/2018/11/30/autograder-issues-upset-students-berkeley. Acesso em 18/01/2021.

[12] Aaron Keen and Kurt Mammen. 2015. Program decomposition and complexity in CS1. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education.* 48–53.

[13] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[14] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive difficulties faced by novice programmers in automated assessment tools. *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research* (2018), 41–50. https://doi.org/10.1145/3230977.3230981

[15] Juha Sorva. 2012. *Visual program simulation in introductory programming education.* Aalto University.

[16] Nghi Truong, Paul Roe, and Peter Bancroft. 2004. Static analysis of students' Java programs. In *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30.* Citeseer, 317–325.

[17] Henry M. Walker. 2017. ACM RETENTION COMMITTEE Retention of Students in Introductory Computing Courses: Curricular Issues and Approaches. *ACM Inroads* 8, 4 (Oct. 2017), 14–16. https://doi.org/10.1145/3151936