Mapeamento sistemático sobre resolução de problemas em disciplina introdutória de programação com teste de software

Yuri Rafael Grajefe Feitosa[1], **Marco Aurélio Graciotto Silva**[1,2], **José Augusto Fabri**[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Programa de Pós-Graduação em Informática (PPGI)
Cornélio Procópio – PR – Brasil

[2]Universidade Tecnológica Federal do Paraná (UTFPR)
Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Campo Mourão – PR – Brasil

yurirafael@alunos.utfpr.edu.br, {magsilva, fabri}@utfpr.edu.br

Abstract. *Software testing is a way of dealing with the difficulties faced in the programming learning process for newer students in Computing undergraduate programs. The objective of this research is to identify studies on this topic with relation to problem solving in introductory programming disciplines. For this, a systematic mapping was done based on the resolution of problems in introductory programming courses with the application of software testing techniques and TDD (Test-Driven Development) as a way to better structure the student's logical reasoning. Several approaches were identified, with positive results regarding problem solving in introductory programming courses using software testing.*

Resumo. *Teste de software é um forma de tratar as dificuldades enfrentadas no processo de aprendizagem de programação para alunos iniciantes nos cursos de Computação. O objetivo desta pesquisa é identificar estudos neste tema com relação à resolução de problemas em disciplinas introdutórias de programação. Para isto, foi realizado um mapeamento sistemático com base na resolução de problemas em disciplinas introdutórias de programação com a aplicação de técnicas de teste de software e TDD (Test-Driven Development) como forma de estruturar melhor o raciocino lógico do aluno. Diversas abordagens foram identificadas, com resultados positivos quanto à resolução de problemas em disciplinas introdutórias de programação com auxílio de teste de software.*

1. Introdução

O ensino introdutório de Computação possui uma taxa de reprovação elevada [Watson e Li 2014, Bennedsen e Caspersen 2019] e é responsável pela maioria das desistências em cursos de Computação [Beaubouef e Mason 2005]. Dentre os fatores relacionados a isto, observa-se a baixa habilidade de resolução de problemas.

A resolução de problemas em Computação envolve: entendimento, caracterização e decomposição do problema; coleta, armazenamento e interpretação de dados; reflexão sobre o problema e suas suposições; reflexão sobre o conhecimento necessário, estratégias de solução e sobre a solução propriamente dita, incluindo atividades de validação e verificação [Salehi et al. 2020, Brennan e Resnick 2012].

Ao ensinar explicitamente técnicas para resolução de problemas, observa-se melhor desempenho de estudantes em disciplinas introdutórias de programação [Koulouri et al. 2015], contribuindo para o tratamento deste cenário.

Desta forma, é pertinente identificar estudos em educação em Computação e o efeito de atividades relativas ao desenvolvimento da competência de resolução de problema, medindo-se a diferença dos resultados durante a vida acadêmica. Evidenciando-se essa diferença, pode-se analisar a viabilidade de transferir as atividades em questão para períodos iniciais do curso, estabelecendo-se subsídios para melhorar o ensino de resolução de problemas.

Neste trabalho, o foco reside no emprego de técnicas e conceitos de teste de software em atividades de aprendizagem relacionadas à resolução de problemas. Diversos trabalhos em educação em programação abordam a questão de teste de software [Utting et al. 2013] e sua integração com ensino de programação, seja pela avaliação com auxílio de critérios de teste ou pela integração de atividades de teste no desenvolvimento, em variações da abordagem de desenvolvimento baseado em testes (TDD) [Edwards 2004, Janzen et al. 2013, Pachulski Camara e Graciotto Silva 2016].

Outra perspectiva encontrada durante a pesquisa foi a aplicação de Dojo de programação, em que grupos de desenvolvedores se reúnem para fazer *networking* e compartilhar conhecimentos, favorecendo assim o aprendizado voluntário e colaborativo [Mourão 2017]. De modo geral, tais abordagens proporcionam um efeito positivo na aprendizagem ao oferecer um direcionamento para o desenvolvimento e um retorno rápido e contínuo sobre a correção das atividades feitas pelos estudantes. Alunos que foram explicitamente solicitados a refletir sobre a declaração do problema via caso de teste são mais propensos a criar um modelo mental da tarefa correto [Denny et al. 2019]

Considerando esse contexto, o objetivo deste trabalho foi identificar os estudos sobre aprendizagem introdutória de programação em cursos de graduação em Computação que aplicam conceitos e técnicas de teste de software para o desenvolvimento de resolução de problemas. O método de pesquisa escolhido foi o mapeamento sistemático, propiciando uma forma abrangente e consistente para recuperar e analisar estudos primários relevantes de forma eficiente para investigar uma área de pesquisa [Kitchenham et al. 2010]. O protocolo do mapeamento é descrito na Seção 2. Dos estudos selecionados, foram extraídos dados sobre técnicas de teste de software e TDD utilizadas e como elas contribuem para a resolução de problemas, conforme apresentado na Seção 3 e discutido na Seção 4. As considerações finais e trabalhos futuros são apresentados na Seção 5.

2. Método

O objetivo deste trabalho foi caracterizar o uso de conceitos e técnicas de teste de software ou de TDD na resolução de problemas de programação com foco no desempenho de aprendizagem de estudantes dos períodos iniciais de cursos de graduação da área de Ciência da Computação. Assim, organizamos o objetivo em três questões de pesquisa:

- Quais elementos de teste de software são considerados em disciplinas introdutórias de programação?
- Quais elementos de TDD são considerados em disciplinas introdutórias de programação?

- Como a resolução de problemas é tratada com conceitos, atividades e técnicas de teste em disciplinas introdutórias de programação?

O método utilizado neste estudo foi o mapeamento sistemático [Kitchenham et al. 2010, Petersen et al. 2008]. Embora existam estudos sobre o ensino de teste de software [Valle et al. 2015, Garousi et al. 2020] e ensino de programação integrado ao teste de software [Scatalon et al. 2017, Scatalon et al. 2019], não pudemos identificar trabalhos que contemplem TDD ou teste de software especificamente quanto à resolução de problemas. Desta forma, o mapeamento sistemático foi realizado, sendo organizado em duas iterações. A primeira iteração foi conduzida em julho de 2020 e contemplou estudos em inglês indexados em veículos internacionais. A segunda iteração foi realizada em fevereiro de 2021, contemplando veículos nacionais da área de Educação em Computação.

A busca foi realizada com em bibliotecas digitais, utilizando expressões de busca construídas a partir do objetivo deste trabalho, conforme a estratégia PICO. Mais especificamente, foram observadas a população (disciplinas introdutórias de programação), a intervenção (teste de software e TDD para resolução de problemas) e resultados esperados (resolução de problema), omitindo-se o controle da expressão de busca. Foram consideradas as bibliotecas digitais ACM Digital Library, IEEExplore e Scopus, contemplando assim as principais revistas e eventos de publicação de estudos em educação em Computação. Além disso, foram realizadas buscas na biblioteca digital da SBC (SOL), no BDBComp e no site da RBIE, de modo a contemplar os principais veículos nacionais de educação em Computação.

(“CS0” OR “CS1” OR “introductory programming” OR “introduction to programming” OR “introduction to computer programming” OR “programming fundamentals” OR “computing education” OR “novice programming”)
AND
(“test-driven” OR “TDD” OR “test-first” OR “software testing” OR “testing” OR “test case” OR “dojo”)
AND
(“problem resolution” OR “problem-solving” OR “problem solving” OR “computational thinking” OR “logic” OR “programming skill” OR “programming plan”)

Para a busca de artigos em âmbito nacional, foi utilizada uma expressão de busca simplificada, com termos mais abrangentes quanto à população e intervenção e sem incluir termos referentes ao resultado esperado, conforme exposta a seguir. Essa simplificação deu-se pela limitação dos mecanismos de busca disponíveis naquelas bibliotecas digitais. Após a realização da busca, foi manualmente verificada a presença de termos equivalentes ao utilizado na busca anterior.

(teste OR dojo) AND (algoritmo OR programação OR computação)

A busca retornou 179 artigos, sendo 11 pela biblioteca digital da ACM, 10 pelo IEEExplore, 30 pelo Scopus, e 128 dos veículos nacionais (RBIE). Foram excluídos 11 artigos duplicados dos resultados do Scopus, restando 19 artigos. Em relação aos veículos nacionais, após a verificação manual pelos termos previstos, apenas um artigo permaneceu. Desta forma, foram recuperados 41 artigos das bibliotecas digitais.

Os artigos recuperados nas bibliotecas digitais com as expressões de busca foram submetidos a critérios de seleção. Para inclusão, todo artigo deve: (1) ser um estudo sobre educação em Computação; (2) tratar de conceitos introdutórios de programação e algoritmos; (3) utilizar teste de software para o desenvolvimento de programas, considerando, por exemplo, a implementação de casos de teste ou por planos manuais de teste de programas desenvolvidos pelos estudantes; e (4) considerar a questão de resolução de problemas com auxílio de atividades de teste de software.

Além disso, foram considerados critérios de exclusão, os quais cada artigo não devem satisfazer: (1) educação em Computação para outras áreas que não a Computação; (2) tratar apenas do ensino de conceitos relacionados à estruturas de dados; (3) utilizar teste de software apenas para avaliar programas desenvolvidos pelos estudantes, sem exigir que o estudante desenvolva casos de teste ou planos de teste; (4) tratar de resolução de problemas, mas sem associar a resolução de problemas com teste de software; (5) usar teste de software, mas sem associar o teste de software com a resolução de problema, utilizando, por exemplo, critérios de teste para avaliar o conjunto de casos de teste ou o programa, mas sem relacionar isto com a melhoria nas técnicas de resolução de problemas; (6) trabalho curto, sem apresentação de resultados; (7) estudo que não foi avaliado pelos pares, ou seja, publicado em veículo sem mecanismos de revisão pela comunidade acadêmica; (8) estudos em idioma distinto de português ou inglês; (9) estudo deve ser primário (não pode ser revisão ou mapeamento sistemático).

No processo de seleção, de forma independente, cada autor montou sua classificação, considerando os critérios de inclusão e exclusão e a leitura de cada artigo. A concordância foi medida pelo kappa de Cohen [Cohen 1960], obtendo-se um kappa de 0,72, o que representa um grau de concordância moderado. Ainda assim, posterior à classificação individual, ocorreu uma discussão entre os autores sobre a relevância de cada artigo, buscando o consenso da seleção. Em suma, foram selecionados 9 artigos.

A extração de dados dos artigos selecionados compreendeu o ano e local de publicação, tipo de estudo, população, intervenção, forma de avaliação da intervenção e resultados obtidos. Esses dados, devidamente relacionados às questões de pesquisa, são apresentados na próxima seção.

3. Resultados

A relação completo dos artigos selecionados, com seus respectivos indicadores utilizados neste artigo, consta na Tabela 1. Os identificadores serão indicados, entre colchetes, para relacionar os artigos selecionados com os resultados apresentados.

Conforme apresentado na Figura 1, a maioria dos estudos foi publicada em eventos (sete estudos): dois artigos no IEEE Frontiers in Education Conference (FIE 2009 [5] e 2017 [6]), um no ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2016 [2]), um no Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006 [7]), um no ACM Conference on Global Computing Education (CompEd 2019 [3]), um no Koli Calling International Conference on Computing Education Research (Koli-Calling 2019 [1]) e um no Workshop de Informática na Escola (WIE 2017 [9]). Os dois artigos restantes foram publicados nos periódicos IEEE Transactions on Software Engineering (TSE, 1980 [4]) e Routledge Interactive Learning Environments (ILE, 1994 [8]). Dos artigos selecionados,

Tabela 1. Relação de artigos selecionados.

ID	Título
1	A Closer Look at Metacognitive Scaffolding: Solving Test Cases Before Programming [Denny et al. 2019]
2	A Strategy to Combine Test-Driven Development and Test Criteria to Improve Learning of Programming Skills [Pachulski Camara e Graciotto Silva 2016]
3	Translation from Problem to Code in Seven Steps [Hilton et al. 2019]
4	A Framework for Discipline in Programming [Hsia e Petry 1980]
5	An approach for problem specification and its application in an Introductory Programming Course [Mendonça et al. 2009]
6	Most common fixes students use to improve the correctness of their programs [De Souza et al. 2017]
7	A novice's process of object-oriented programming [Caspersen e Kölling 2006]
8	Knowledge Integration in Introductory Programming: CodeProbe and Interactive Case Studies [Bell et al. 1994]
9	Uma proposta da eficiência do uso da Metodologia Ativa Baseada em Problemas, utilizando Dojo de Programação, aplicada na disciplina de Lógica de Programação [Mourão 2017]

com exceção daquele publicado no WIE [9], todos os demais foram escritos em inglês. Observamos-se que quatro artigos possuem autores brasileiros [2, 5, 6, 9].

Embora sejam poucos os trabalhos selecionados no mapeamento, cabe destacar que eles estão inseridos em meios de publicação de reconhecida qualidade na área de Educação em Computação e de Engenharia de Software. Metade dos trabalhos foram publicados nos últimos cinco anos, sendo os demais abrangendo desde 1980 até 2009, de forma esparsa. Embora a primeira publicação seja em veículo específico da área de Engenharia de Software, as demais foram em Educação em Computação. De certa forma, isso era esperado: a maioria dos autores atua em pesquisa e ensino, sendo comum a publicação de relatos de experiência ou resultados de novas técnicas de ensino em uma área específica da Computação. Sendo o tópico deste artigo relacionado a tópicos de Engenharia de Software, é natural que ocorra uma transposição das técnicas daquela área para o domínio de Educação.

Estudos que alteram ou definem uma abordagem de programação predominam entre os trabalhos selecionados: seis definem uma nova abordagem que emprega teste de software [3, 4, 5, 7, 8, 9]; dois alteram uma abordagem [1, 2], acrescentando atividades de teste. Um trabalho trata exclusivamente da análise e diagnóstico sobre dificuldades de ensino, entre as quais a resolução de problemas, considerando atividades de teste de software [6]. Em relação às linguagens de programação consideradas no ensino introdutório,

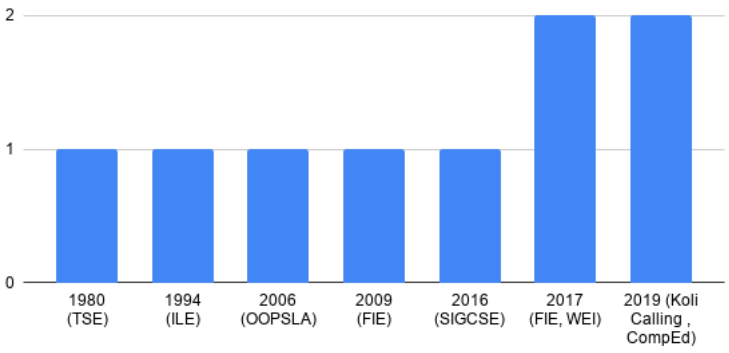

Figura 1. Quantidade de Artigos X Ano de publicação

encontram-se C [1], Fortran [4], Java [2, 7], LISP [8], Python [3, 5] e VisualAlg [9].

Um elemento em comum nas abordagens novas para ensino de programação é a inserção de atividades de teste nos passos iniciais, antes da implementação. Na maioria dos estudos os casos de teste criados nesse momento inicial não são automatizados [1, 3, 4, 8, 9], sendo utilizados para estimular a reflexão do estudante quanto ao problema e resolvê-lo mentalmente. Além disto, algumas abordagens estabelecem a criação de casos de teste considerando critérios de teste funcionais [8] e estruturais [2, 4, 8], seja antes da implementação da resolução do problema [4, 8] ou após [2, 4, 8]. A criação de casos de teste automatizados foi observada apenas em dois estudos [2, 5, 7]. No caso do estudo que analisa resultados, todos os casos de teste eram automatizados [6], sendo os resultados dos casos de teste utilizados para identificar os erros e acertos dos estudantes e, após análise quanto aos comandos utilizados, identificar os erros e a relação entre eles.

Além da atividade de teste em si, diversas abordagens novas agregam elementos para melhorar a estruturação da proposta de solução [4, 7], revisão pelos pares [4, 9] e estudos guiados [8]. Por exemplo, temos fluxogramas podem ser utilizados para o projeto da solução [4] e atividades de programação pareada, com intensa revisão pelo par, na forma de Dojos de Programação [9].

Todos os estudos que estabelecem uma alteração em uma abordagem existente de ensino consideram a criação de casos de teste antes da implementação. Em um estudo [1], a intervenção consiste na inclusão de uma atividade de teste nos passos iniciais, de forma similar aos demais trabalhos. O outro estudo [2] considera uma abordagem existente baseada em testes (TDD) e adiciona, após a implementação da solução, atividades para melhoria do conjunto de casos de teste segundo critérios estruturais.

Como método para avaliar as abordagens, os estudos utilizaram experimentos ou quasi-experimentos [1, 2, 4, 8] e estudos de caso [3, 5, 6, 7, 8, 9]. Ao analisar o efeito das abordagens, foram considerados: compreensão do problema [1], submissão da implementação [1, 4, 6, 8], detecção de erros [2, 4, 5, 6], cobertura de requisitos de teste [2], esforço [4], dificuldades encontradas [5, 6], nota [1, 8, 9] e opinião geral dos estudantes e instrutores [2, 3, 5, 7, 9].

De modo geral, o efeitos do teste de software em disciplinas introdutórias à programação foram positivos. A reflexão possibilitada pela criação dos casos de teste e execução, mental, dos casos de teste foram benéficas para a compreensão e resolução de problemas [1, 3]. Além disso, observa-se que os estudantes ficaram mais confiantes na resolução de problemas [3, 9]. O uso de critérios de teste contribuem para a criação de casos de teste, guiando o estudante a criar conjuntos de mais qualidade [2] e a explorar melhor o problema e as particularidades a serem tratadas nas implementações [8]. Embora o esforço ao utilizar teste possa ser ligeiramente maior (e sem diferença estatisticamente significativa) do que em uma abordagem tradicional, os ganhos em qualidade são evidentes e significativos [4]. Estratégias que permitem a adoção incremental de teste de software, inclusive com adoção de diversos critérios de teste [8], contribuem para o melhor desempenho do estudante, permitindo não apenas notas melhores, mas a realização de atividades de programação mais complexas do que com a abordagem tradicional [8]. Destaca-se também que a utilização de ferramentas integradas para auxiliar e conduzir a execução da abordagem é considerada importante pelos estudantes, contribuindo para que estudantes com mais dificuldades percebam e tenham benefícios maiores no aprendizado

[8]. Entretanto, cabe observar que, mesmo com atividades de teste de software, observam-se dificuldades nas atividade de programação referentes à resolução de problemas, com indícios percebidos pela evolução da estruturação do código durante as atividades [6].

4. Discussão

Os estudos selecionados apresentam resultados que reforçam os benefícios de considerar atividades de teste de software na educação em Computação quanto ao ensino de programação, dedicando atenção especial a questão de resolução de problemas.

Quanto à primeira questão de pesquisa, o principal elemento de teste de software considerado é o caso de teste. Com exceção do estudo que analisa programas e casos de teste [6], todos os demais empregam o conceito de casos de teste antes da implementação da solução. Muitos estudos em ensino de programação, ao integrar teste de software, consideram apenas casos de teste automatizados, ou seja, a implementação de um caso de teste que pode ser automaticamente executado e avaliado. Embora alguns dos trabalhos selecionados façam uso disso, a maioria dos trabalhos utiliza apenas o caso de teste puro, sem exigir a implementação dos casos de teste pelo estudante. Ao requerer que o estudante defina um caso de teste, estabelecendo o valor esperado para um dado de entrada sugerido, especificando os dados de entrada e resultados esperados ou mesmo executar mentalmente o algoritmo que solucionaria o problema considerando um caso de teste fornecido, observa-se o desenvolvimento de modelos mentais adequados para a resolução do problema proposto e da implementação da solução.

Além de casos de teste, outro conceito de teste de software utilizado é relativo à técnicas e critérios de teste. O uso de técnicas funcionais e estruturais colaboraram para organizar a criação de um conjunto diversificado de casos de teste. Entretanto, os estudos não apresentam evidências fortes de que os conjuntos de teste contribuíram direta e positivamente para o ensino de programação e para auxiliar a resolução de problemas.

Em relação à segunda questão de pesquisa, alguns trabalhos abordam o TDD e outros consideram ao menos a criação de casos de teste antes da implementação. Em um dos estudos com TDD, o passo de refatoração é utilizado para melhorar o conjunto de casos de teste, considerando critérios estruturais. A investigação sobre a consideração dos critérios para a criação de casos de teste no início do processo (antes da implementação) não é abordada, com exceção do trabalho que requer a criação de um fluxograma e emprega critérios estruturais para definir requisitos de teste a serem satisfeitos.

Quanto à terceira questão de pesquisa, a resolução de problemas é tratada principalmente com a criação de casos de teste antes da implementação da solução. Também existem atividades, não relacionadas a teste, que foram empregadas e que colaboram para a resolução de problemas: revisão pelos pares, estudos de caso dirigidos e dojos de programação. Tais atividades, ao serem associadas ao teste de software, complementam as abordagens de ensino de programação propostas, tratando de aspectos de colaboração relevantes à resolução de problemas e que não podem ser tratadas isoladamente com teste.

Atendo-se à perspectiva isolada de TDD ou técnicas de teste de *software*, existe uma dificuldade em avaliar a conformidade com essas técnicas pelo estudante. Por exemplo, quanto a técnicas e critérios, embora existam medidas de cobertura, como facilitar a interpretação delas para os estudantes ou assegurar que ele entende um requisito de teste específico não é trivial. Da mesma forma, é difícil avaliar se o estudante segue

exatamente uma estratégia similar ao TDD. Desta forma, é importante avaliar se os alunos estão aderindo aos métodos, bem como entender suas atitudes [Buffardi e Edwards 2012] e entender como as atividades de teste contribuem efetivamente para a resolução de problemas no ensino introdutório de programação.

Considerando a atividade mais ampla de ensino de programação, deve-se ter especial consideração com o fato de ser uma disciplina introdutória de um conteúdo essencial em Computação. Embora alguns estudantes tenham formação prévia em Computação, esse não é o cenário da maioria. Além disso, existem estudantes que enfrentam mais dificuldades que outros. Definir uma abordagem que permite que todos os estudantes tenham sucesso é um desafio e requer não apenas o estabelecimento de sequência adequada de intervenções pedagógicas, mas também o emprego de ferramentas que permitam fornecer uma resposta (feedback) apropriada para cada estudante. Por exemplo, embora para alguns estudantes pareça desnecessário uma mensagem de confirmação de que a interpretação de um determinado caso de teste esteja correto, para outros estudantes, com mais dificuldades, essa mensagem traz mais confiança de que ele progrediu no tarefa [8].

Um fator que deveria ser considerado na aprimoração da resolução de problemas computacionais é o professor. Desenvolver tarefas e coordenador atividades de programação eficazes para os estudantes é difícil. Os estudos encontrados não abordam essa questão. Incentivar mecanismos para documentação e publicação desses recursos, para uso e adaptação por outros educadores, é pertinente [Edwards et al. 2008].

5. Conclusões

O emprego de teste de software no ensino de programação é considerado benéfico para a aprendizagem. No entanto, poucos estudos tratam da questão de resolução de problemas com o auxílio de técnicas de teste. Neste artigo, foram apresentados os resultados de um mapeamento sistemático que investiga essa questão. Comparando-se com a ampla literatura sobre ensino de programação e os estudos sobre integração de teste de software neste ensino introdutório, observa-se que ainda são poucos os trabalhos que tratam das contribuições de teste no desenvolvimento de habilidades para resolução de problemas. Ao mesmo tempo, foi possível identificar abordagens que integram diversos conceitos de teste de software ao ensino de programação, com contribuições que melhoraram a qualidade da educação nesta disciplina introdutória.

Entretanto, cabe a análise de estratégias que conciliem mais técnicas de teste e que permitam ao estudante explorar os problemas a serem resolvidos quanto a cenários mais restritos. Por exemplo, poderiam ser consideradas: classes inválidas considerando critérios funcionais; a particularidades do espaço da solução, relacionados a critérios estruturais; e erros típicos cometidos por estudantes, com auxílio de teste de mutação. No entanto, isso deve ser feito sem exigir o domínio dessas técnicas pelos estudantes, o que sugere a utilização de geração de dados de teste, para interpretação das consequências pelos estudantes, e o fornecimento de informação (feedback) que permite a ele entender os princípios relacionados aos critérios envolvidos. Desta forma, será possível abordar a questão de resolução de problema de forma mais completa, contribuindo para uma formação mais sólida e facilitando a ligação com outras disciplinas de Computação e, especialmente, de Engenharia de Software.

Referências

Beaubouef, T. e Mason, J. (2005). Why the high attrition rate for computer science students: Some thoughts and observations. *SIGCSE Bulletin*, 37(2):103–106.

Bell, J., Linn, M., e Clancy, M. (1994). Knowledge Integration in Introductory Programming: CodeProbe and Interactive Case Studies. *Interactive Learning Environments*, 4(1):75–95.

Bennedsen, J. e Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *Inroads*, 10(2):30–36.

Brennan, K. e Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Annual Meetting 2012 – American Educational Research Association*, p. 1–25.

Buffardi, K. e Edwards, S. H. (2012). Exploring influences on student adherence to test-driven development. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, p. 105–110, Haifa, Israel. ACM.

Caspersen, M. e Kölling, M. (2006). A novice's process of object-oriented programming. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, volume 2006, p. 892–900, Portland, OR.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

De Souza, D., Kölling, M., e Barbosa, E. (2017). Most common fixes students use to improve the correctness of their programs. In *Proceedings - Frontiers in Education Conference, FIE*, volume 2017-October, p. 1–9. IEEE. ISSN: 15394565.

Denny, P., Prather, J., Becker, B. A., Albrecht, Z., Loksa, D., e Pettit, R. (2019). A Closer Look at Metacognitive Scaffolding: Solving Test Cases Before Programming. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, p. 1–10, Koli, Finland. ACM.

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *35th SIGCSE Technical Symposium on Computer Science Education*, p. 26–30, New York, NY, EUA. ACM.

Edwards, S. H., Börstler, J., Cassel, L. N., Hall, M. S., e Hollingsworth, J. (2008). Developing a common format for sharing programming assignments. *ACM SIGCSE Bulletin*, 40(4):167–182.

Garousi, V., Rainer, A., Lauvås, P., e Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, 165.

Hilton, A. D., Lipp, G. M., e Rodger, S. H. (2019). Translation from Problem to Code in Seven Steps. In *Proceedings of the ACM Conference on Global Computing Education*, p. 78–84, Chengdu,Sichuan, China. ACM.

Hsia, P. e Petry, F. (1980). A Framework for Discipline in Programming. *IEEE Transactions on Software Engineering*, SE-6(2):226–232.

Janzen, D. S., Clements, J., e Hilton, M. (2013). An evaluation of interactive test-driven labs with WebIDE in CS0. In *35th International Conference on Software Engineering*, p. 1090–1098, Piscataway, NJ, EUA. IEEE.

Kitchenham, B. A., Budgen, D., e Brereton, O. P. (2010). The value of mapping studies – a participant-observer case study. In *14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, p. 1–9, Swinton, Reino Unido. British Computer Society.

Koulouri, T., Lauria, S., e Macredie, R. D. (2015). Teaching introductory programming: A quantitative evaluation of different approaches. *Transactions on Computing Education*, 14(4):26:1–26:28.

Mendonça, A., Guerrero, D., e Costa, E. (2009). An approach for problem specification and its application in an Introductory Programming Course. In *2009 39th IEEE Frontiers in Education Conference*, p. 1–6. ISSN: 2377-634X.

Mourão, A. B. (2017). Uma proposta da eficiência do uso da metodologia ativa baseada em problemas, utilizando dojo de programação, aplicada na disciplina de lógica de programação. (10):667–676.

Pachulski Camara, B. H. e Graciotto Silva, M. A. (2016). A Strategy to Combine Test-Driven Development and Test Criteria to Improve Learning of Programming Skills. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, p. 443–448, Memphis, Tennessee, USA. ACM.

Petersen, K., Feldt, R., Mujtaba, S., e Mattsson, M. (2008). Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008)*, p. 68–77, Swindon, Reino Unido. BCS Learning & Development.

Salehi, S., Wang, K. D., Toorawa, R., e Wieman, C. (2020). Can majoring in computer science improve general problem-solving skills? In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, p. 156–161, New York, NY, EUA. ACM.

Scatalon, L. P., Barbosa, E. F., e Garcia, R. E. (2017). Challenges to integrate software testing into introductory programming courses. In *2017 IEEE Frontiers in Education Conference (FIE)*, p. 1–9, Piscaway, NY, EUA. IEEE.

Scatalon, L. P., Carver, J. C., Garcia, R. E., e Barbosa, E. F. (2019). Software testing in introductory programming courses: A systematic mapping study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, p. 421–427, New York, NY, EUA. ACM.

Utting, I., Tew, A. E., McCracken, M., Thomas, L., Bouvier, D., Frye, R., Paterson, J., Caspersen, M., Kolikant, Y. B.-D., Sorva, J., e Wilusz, T. (2013). A fresh look at novice programmers' performance and their teachers' expectations. In *18th Annual Conference on Innovation and Technology in Computer Science Education*, p. 15–32, New York, NY, EUA. ACM.

Valle, P., Barbosa, E. F., e Maldonado, J. (2015). Um mapeamento sistemático sobre ensino de teste de software. In SBC, editor, *26th Brazilian Symposium on Informatics in Education*, Porto Alegre, RS,. SBC.

Watson, C. e Li, F. W. B. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, p. 39–44, New York, NY, EUA. ACM.