V Congresso sobre Tecnologias na Educação (Ctrl+E 2020)
Educação do Futuro: Tecnologias e Pessoas para Transformar o Mundo
João Pessoa – PB | Brasil | Online
25 a 28 de Agosto de 2020

Feedback no Ensino de Lógica de Programação com o Auxílio de Ferramentas para Apoiar o Ensino e a Aprendizagem: Uma Abordagem Empírica

Dirson Santos de Campos[1], Deller James Ferreira[1]

[1]Universidade Federal de Goiás (UFG)

{dirson, deller}@inf.ufg.br

Abstract. *Feedback is one of the ways to inform students about their academic performance. Feedback can improve and reinforce student learning once detecting the error is the first step to correct it. In this way, it is an essential component in the learning cycle, providing reflection and cognitive development on the part of the student. Teaching in Computing, particularly in algorithms and programming, requires the use of tools to be able to correct a large amount of source code produced by students, which makes mass production of feedback impossible manually. In this paper, automatic exercise correction tools were analyzed and an empirical study based on the results of this analysis was performed, generating ipsatives feedback.*

Resumo. *Feedbacks é uma das maneiras de informar aos estudantes sobre seu desempenho acadêmico. Ele pode melhorar e reforçar o aprendizado dos estudantes, uma vez que detectar o erro é o primeiro passo para corrigi-lo. Dessa forma, é um componente essencial no ciclo de aprendizado, proporcionando reflexão e desenvolvimento cognitivo por parte do estudante. O ensino de computação, particularmente em algoritmos e programação, requer o uso de ferramentas para poder corrigir uma grande quantidade de código fonte produzido pelos alunos, o que torna impossível a produção em massa de feedback manualmente. Neste artigo, foram analisadas ferramentas de correção automática de exercícios e um estudo empírico com base nos resultados dessa análise foi feito, gerando feedbacks ipsativos.*

1. Introdução

O ensino de conceitos de algoritmos e programação ocorre tanto em nível superior quanto no Ensino Básico, particularmente no Ensino Médio [Mattos e Pinto 2016], e também em OBIs (Olimpíadas Brasileira de Informática). A OBI, na sua modalidade programação que é aplicada a estudante do Ensino Médio, aplica provas com correção automática de exercícios de programação usando ferramenta do tipo *Online Judge* [Santos et al. 2015]. Em ambos tipos de grau do ensino, básico ou superior, é necessário adotar estratégias pedagógicas para informar o desempenho discente, entre elas, o fornecimento de *feedbacks. Feedbacks manuais são inviáveis para os docentes, em termos práticos,* por exemplo, somente na correção em duas listas de exercícios descrito na Figura 1 seriam necessários 549 *feedbacks*.

Ferramentas para o apoiar o ensino e a aprendizagem de lógica de programação que gerem *feedbacks* automáticos, tais como, ferramentas *Online Judge* viabilizam a comunicação dos resultados aos discentes de imediato [Francisco et al. 2016].

Nenhuma tecnologia, por si mesma, pode equacionar problemas educativos [Silva 2018]. Logo é necessário avaliar a ferramenta tecnológica em sala de aula. Foi feito um estudo empírico, neste artigo, sobre a usabilidade da ferramenta *Sharif-Judge* [Narderi 2020] que se mostrou produtiva na correção automática de exercícios de programação, mas também apresentou limitações em relação ao conteúdo do *feedback*.

Feedbacks são importantes em qualquer atividade humana porque explicitam informações sobre o padrão de conduta e/ou desempenho esperado ou desejado. Eles contêm informações que reorientam e estimulam as pessoas de modo que elas possam se adequar e melhorar o rendimento e/ou comportamento na execução de uma tarefa.

A detecção sistemática de erros lógicos é fundamental para um bom desempenho dos estudantes em diversas disciplinas que envolvem programação e terá um alto impacto no desempenho acadêmico dos estudantes e, no caso de estudantes de Ensino Básico influenciará na escolha deles a uma carreira superior uma vez que a informatização é onipresente e vários *softwares* possuem módulos de programação.

A correção automática de exercícios de programação é o principal atrativo do uso deste tipo ferramenta em disciplinas de programação. Do ponto de vista docente, a correção automática é a ideal devido ao grande número de estudantes que produzem milhares de códigos fontes diferentes por disciplina. Da perspectiva do discente, oferece um *feedback* imediato, embora semanticamente pobre para este público-alvo, porém útil. O *feedback é* destinado a ajudar os alunos a melhorar seu trabalho e é indiscutivelmente um fator importante na aprendizagem de programação [Keuning et al. 2016] e na aprendizagem de qualquer disciplina em nível superior [McConlogue 2020].

É comum professores terem dúvidas e dificuldades a respeito de como explorar diversos tipos de ferramentas com finalidades diferentes em sala de aula [Keuning et al. 2016]. A dificuldade de adequar as ferramentas a necessidades discentes com *backgrounds* diversificados, levando a ritmos e curva de aprendizagem diferenciados.

As principais contribuições deste artigo, além da resposta as questões de pesquisa, referem-se tanto na forma de conduzir o estudo empírico em sala de aula quanto na métrica usada para determinar a eficácia do *feedback* automatizado fornecido aos discente. A métrica é quantitativa (erros lógicos detectáveis em testes automatizados) extraída diretamente do código fonte. As análises dos resultados permitem *feedbacks* ipsativos. A contribuição secundária é a análise da ferramenta usada no estudo empírico em seus aspectos positivos e negativos.

O *feedback* ipsativo (*ipsative feedback*) avalia o progresso do estudante ao desempenhar uma tarefa com o objetivo de aprimorar a si mesmo, pois comparar resultados atuais do discente com anteriores na mesma tarefa [McConlogue 2020].

2. Metodologia e Questões de Pesquisa

Sinteticamente a metodologia de pesquisa foi dividida em duas partes. Inicialmente foi feita uma revisão da literatura com trabalhos correlatos para responder as questões de pesquisa tende em conta o estado-da-arte (seção 3 e 4). Os resultados desta análise permitiram verificar *gaps* que permitiu o desenvolvimento de um estudo empírico, em sala de aula que foi descrito na seção 6 deste artigo.

As questões de pesquisa são: QP1: Quais os pontos fortes e fracos do uso das ferramentas que apoiam o ensino de programação em sala de aula? QP2: Quais são as

dificuldades do uso de ferramentas de apoio no ensino da programação do ponto de vista docente? QP3: Quais são as vantagens de ser produzir *feedbacks* aos discentes?

3. Trabalhos correlatos

Nesta seção, foi feita uma meta-análise das Revisões Sistemáticas (RS) e de outros artigos [Francisco et al. 2016, Li et al. 2020, Marcolino et al. 2015, Muuli et al. 2020, Ossenberg et al. 2019, Santos et al. 2015, Silva et al. 2015a, Silva et al. 2015b, Souza e Favero 2015] que usam ferramentas pedagógicas em sala de aula, em particular, as que envolvem correção automática de exercícios e, dentre elas, quais são baseadas em testes automatizados e também quais são as técnicas de testes utilizadas pelos pesquisadores e/ou idealizadores destas ferramentas.

Ao todo as RS supracitadas descrevem 271 estudos secundários, sendo que em 201 deles envolve o uso de ferramentas em sala de aula. Dentre eles, foi encontrado 26 (12,94%) dos 201 artigos que usam ferramentas de correção automática de exercícios baseada em testes. As ferramentas possuem estratégias tanto caixa-preta quanto caixa-branca. Em 23 dos artigos (88,23%) dos casos utiliza-se teste de caixa-preta que são mais populares pois a correção dos códigos fontes é feita sem o conhecimento do código fonte discente pelo professor.

Li et al. (2020) afirma que os *feedbacks* são muito importantes para promover a participação dos estudantes e envolve aspectos, entre outros, referentes a motivação dos alunos, atenção, relevância, confiança e satisfação. Estes aspectos são difíceis de mensurar efetivamente. Eles fizeram um *survey* que é uma pesquisa executava em retrospectiva onde os meios principais para coletar as informações quantitativas e qualitativas são os questionários uma métrica indireta de coleta de dados, na seção 5, foi utilizada uma outra abordagem da Engenharia de *Software* Experimental onde optamos em fazer um experimento ou estudo empírico baseado na métrica que pode ser mensurada diretamente por testes do *dataset* (Figura 1) que é a detecção de erro lógico.

Apesar da dificuldade de fornecem *feedbacks* sistemáticos que possam acompanhar o desempenho do estudante no ensino superior existem também outros aspectos como as percepções diferentes no processo de *feedback* [Carless 2006], tais como, a qualidade do *feedback* pode estar, portanto, influenciando a qualidade das soluções discentes segundo Muuli et al. (2020). No caso da pesquisa desenvolvida neste artigo, não conseguem avaliar todos os códigos-fontes submetidos pelos discentes e fornecer *feedback* imediatos sem o uso de uma ferramenta. Logo foi adotada uma ferramenta para este fim que exige pouco treinamento, por parte do discente, para usá-la e receber o *feedback* dos testes automatizados para a detecção de erros lógicos.

Ossenberg et al. (2019) analisou 61 pesquisas que tratam sobre *feedbacks*, pesquisas não empíricas formam 90% dos casos e estudos empíricos 10%. Deste modo, estudos empíricos sobre *feedbacks* são mais raros, por envolver aspectos tantos pedagógicos quantos tecnológicos, no caso deste artigo, os aspectos tecnológicos envolvem a análise dos resultados dos testes automatizados.

O *feedback* é um processo de dialógico professor-aluno, porém os estudantes parecem silenciosos em seu envolvimento com o *feedback*, no estudo empírico da seção 5 foi observado *in loco* este fato. O silêncio do aluno em relação a qualidade do *feedback* pode ser atribuído a experiências anteriores dos estudantes com ferramentas

ou, até mesmo, nas relações de poder desiguais percebidas entre o aluno e o professor, que podem ameaçar a autoestima do estudante e o medo de expor suas fraquezas.

4. Discussões sobre as questões de pesquisa

Por razões didáticas, as respostas da questão de pesquisa seguem a metodologia de pesquisa, ou seja, primeiramente será analisado os resultados provenientes da literatura.

4.1. Resposta à questão de pesquisa QP1

A QP1 se refere aos pontos fracos e fortes das ferramentas no contexto deste artigo. Os maiores contributos didático do uso de ferramentas de correção automática de exercícios é a própria correção do exercício feita pela ferramenta e o seu *feedback* automático [Marcolino et al. 2015, Francisco et al. 2016, Carvalho et al. 2016]. A correção é automática e os *feedbacks* automáticos proporciona uma redução da carga de trabalho docente.

O ponto fraco das ferramentas, detectados em experimentos pedagógicos, refere-se a qualidade e expressividade do *feedback* automático que se resume, na maioria das ferramentas a uma frase que não oferece informações semânticas sobe o erro lógico, mas pelo menos o estudante sabe que o código-fonte não passou nos testes lógicos automatizados, ou seja, possui algum tipo de erro lógico não detectado pelo estudante. Este fato também foi observado no experimento descrito neste trabalho.

4.2. Resposta à questão de pesquisa QP2

A QP2 se refere as dificuldades do uso de ferramentas de apoio no ensino da programação do ponto de vista docente. Os professores usam as ferramentas em sala de aula como auxílio para o ensino e a correção de exercícios de programação. As ferramentas detectam erros lógicos com precisão, mas não fornecem informações aos docentes sobre o porquê dos erros lógicos e, do ponto de vista do discente, as ferramentas não fornecem auxílio expressivo ao estudante para um posterior tratamento dos erros lógicos de programação depois de detectá-los. A mensagem enviada como *feedback* automático é a resposta em forma de uma frase pré-definida. É apropriada aos objetivos didáticos de uma maratona de programação, onde o objetivo didático principal é eliminar o candidato fraco e elevar o forte. Novatos necessitam de mais informações.

Uma das limitações do uso destas ferramentas, do ponto de vista docente, é a falta de algum tipo de análise qualitativa das soluções discentes o que obriga o professor analisar os códigos fontes para fornecer uma nota didaticamente justa. Problemas e dificuldades do ensino de lógica de programação do ponto de vista docente correspondem a somente 2% dos trabalhos analisados por Souza et al. (2016).

Foram encontradas poucas pesquisas qualitativas envolvendo ferramentas didático-pedagógicas do ponto de vista docente. Foi feita uma pesquisa qualitativa com professores dos de cursos tecnológicos em Institutos Federais onde a computação não é atividade fim [Mattos e Pinto 2016]. Perguntou-se aos professores se eles já haviam usados ferramentas ou recursos de programação em disciplina que leciona que os auxiliassem na correção de tarefas discentes, 69% dos professores disseram que não. Limitavam a adoção de ferramentas essenciais como o compilador e ao ambiente de programação. Entretanto, está não é uma rejeição a outros tipos de ferramenta, pois em outra pergunta, nesta mesma pesquisa, professores responderam que se tivessem apoio

institucional para o desenvolvimento de alguma atividade ou projeto, mesmo de modo interdisciplinar na disciplina que lecionam, os professores responderam, em 95% dos casos que aceitariam participar pois isto seria potencialmente benéfico.

Desta forma, o motivo de não se adotar ferramentas de correção automação, bem como outras ferramentas didáticas inovadoras, não é apenas pela resistência a inovação em si e a conformidade com o *status quo*, envolvem outras questões, tais como, suporte institucional e o tempo docente aplicado ao aprendizado da ferramenta, afinal toda ferramenta possui uma curva de aprendizagem. Entretanto o problema de evasão dos cursos de programação é conhecido e estudos indicam que os usos de ferramentas didáticas podem motivar os estudantes [Santos et al. 2015].

4.3. Resposta à questão de pesquisa QP3

A QP3 se refere as vantagens de ser fornecer *feedbacks* aos discentes.

Neste artigo a resposta da QP3 é feita tanto pela análise de boas pesquisas sobre o assunto [Ossenberg et al. 2019, Muuli et al. 2020] quanto pelas conclusões do estudo empírico baseado na análise dos seus resultados.

Do ponto de vista discentes os principais benefícios de *feedbacks* escritos ou orais é que eles permitem identificar possíveis equívocos e deficiências no aprendizado do estudante, pois *feedbacks* são uma das formas mais eficientes de comunicação do professor-aluno de modo a informar com precisão que desempenho que está tendo um estudante em um determinado tópico, tarefa ou exercício [Ossenberg et al. 2019].

No caso de disciplinas básicas de programação, os conteúdos de *feedbacks* ideais devem ter informações sobre os erros e falhas lógicas, bem como dicas de como superá-las. Eles deveriam ser construídos de acordo com a natureza dos erros lógicos cometidos. Infelizmente as ferramentas estado-da-arte não fornecem *feedbacks* pedagógicos ricos em informações semânticas do porquê dos erros lógicos.

5. Estudo empírico em Sala de Aula com ferramenta baseado em testes de software automatizado do tipo caixa-preta

Os resultados obtidos por Valle et al. (2015) afirmam que 52% dos pesquisadores usam estudos empíricos para validar seus trabalhos. Esta foi uma grande motivação para também propormos um estudo empírico para contribuir com um estudo primário próprio envolvendo as questões de pesquisa QP1 e QP2 e também uma nova estratégia experimental baseada na mensuração dos resultados dos *feedbacks*, não por variáveis qualitativas coletadas em um questionário sobre a avaliação da qualidade dos *feedbacks* por parte dos discentes, mas sim por uma métrica quantitativa direta, retirada dos resultados dos testes lógicos automatizados intermediários, que é ignorada pela maioria dos experimentos que se limitam a três resultados básicos: satisfaz todos os casos de testes, não satisfaz nenhum caso de teste e satisfaz alguns casos de teste.

Foi utilizado neste estudo empírico uma ferramenta, não usada em maratonas e olimpíadas de programação porque o *ranking* estudantil de acertos pode levar a estímulos negativos para estudante de baixo rendimento que tem seu desempenho explicitado. A ferramenta escolhida foi o *Sharif-Judge* [Narderi 2020].

5.1. Descrição do Estudo empírico

O Estudo empírico envolveu duas turmas de estudantes, uma de 28 estudantes, outra de 29 estudantes. Não há necessidade do grupo de controle pois as amostras são emparelhadas por estudantes. Todos os estudantes são do grupo experimental e recebem o *feedback* ipsativo que não faz uma comparação com os demais estudantes e sim comparar resultados atuais do discente com os seus resultados anteriores na mesma tarefa [McConlogue 2020]. Todo são de disciplinas de Introdução a Computação da Universidade Federal de Goiás (UFG) no paradigma imperativo, linguagem C, do primeiro semestre de 2019. Os estudantes submeteram os mesmos exercícios, disponibilizados nas mesmas listas, usando a mesma ferramenta *Sharif-Judge*. Além disso, eles foram submetidos as mesmas avaliações. Para cada exercício deve-se incluir no enunciado um *input* e *output* para evitar erros de apresentação nas submissões.

Neste estudo empírico foi adotado a seguinte estratégia para o Teste de Hipótese Paramétrico para os dados quantitativos coletados da ferramenta. A variável quantitativa se refere o percentual de acertos nos casos de testes a um determinado código fonte discente. Por exemplo, se foram feitos 4 casos de teste no *suite case* para detecção de erros lógicos em um determinado exercício de uma lista qualquer. O estudante submete a sua resposta caso o seu código satisfaz um dos casos de testes testado automaticamente pela ferramenta, qualquer um deles e falha nos demais. Neste caso o percentual é de 25% de acerto. Para cada submissão de código fonte o resultado obtido pelo estudante é automático, sendo assim, o resultado dos testes lógicos brutos é, em síntese, o resultado do *feedback* automático quando o programa compila.

O estudo empírico analisa os resultados da execução dos casos de testes para cada exercício e estes resultados são armazenados no PostgreSql. Os resultados parciais da execução dos casos de testes não são analisados por ferramentas estado-da-arte *On-line Judge* ao criar o texto do *feedback*. Em nosso experimento estes resultados parciais fazem parte do conteúdo ipsativo do *feedback*. Este tipo de *feedback* permite comparar se houve ganho na qualidade do código-fonte discente ao comparar as amostras emparelhadas produzidas pelo mesmo estudante, ou seja, se em uma nova versão o código fonte fazer parte do satisfez mais ou menos casos de testes o professor pode analisar o erro. O *feedback* ipsativo permite ao docente comparar o aprendizado do estudante tendo como métrica a diminuição ou erradicação dos erros lógicos em fontes.

O teste paramétrico aplica-se quando os dois conjuntos de observações do mesmo indivíduo, no caso o mesmo exercício com códigos fontes submetido n vezes por um mesmo estudante durante o tempo em que a submissão estiver aberta. No caso, a primeira amostra analisa os códigos fontes submetidos nas primeiras 24 horas para obter uma versão inicial dos erros lógicos dos estudantes sem erros de sintaxe, se existirem. A segunda amostra é a versão final do estudante quando se encerra a submissão das tarefas. O estudante pode escolher qual é a versão final dele para avaliação docente que não precisar ser necessariamente a última versão submetida.

5.2. Execução do Experimento

A Figura 1 representa os resultados das submissões do estudo empírico que foram propostos aos estudantes com exercícios diferentes e com assuntos típicos de disciplinas introdutórias de programação no paradigma imperativo usando a linguagem C.

Figura 1 – Resultados das submissões da Lista 3 e 4 usando o *Sharif-Judge*

Foi escolhido como amostra os 38 exercícios de duas listas (Lista L3, sobre *Strings* e Lista 4 sobre matrizes). As listas são do conteúdo programático mais ao final da disciplina. Analisando a Figura 1 observa-se que neste extrato amostral do *dataset* foram submetidos 549 códigos-fontes. Como são 57 estudantes ao todo, a média de submissões, por estudante, para cada exercício foi de 9,63 códigos-fontes.

A ferramenta possui uma opção *Final Submissions* que permite coletar os resultados dos testes lógicos aplicados a versão final escolhida pelo próprio estudante para avaliação. Esta versão é a usada para avaliação final do exercício, desta forma é a segunda amostra emparelhada. A qualquer momento pode-se fazer a avaliação do código. No caso deste estudo, conforme foi dito, foi escolhido 24 horas depois da abertura para submissão usando a opção *All Submission* onde é possível selecionar uma das listas, por exemplo, a lista L4 referente a matrizes e pode-se coletar os dados da submissão em formato da planilha excel (opção all_excel). Estes dados coletados inicialmente formam a primeira amostra do código-fonte emparelhada.

5.3. Análise das submissões discentes de códigos-fontes no Estudo Empírico

Os resultados empíricos proveram evidências de que as ferramentas não oferecem *feedback* automático, com mensagens do porquê do erro, visto que, se as mensagens fossem claramente entendidas não haveriam tantos reenvios espaçados no tempo. A coleta dos resultados para esta inferência referente ao número de submissões média, conforme foi dito, para cada exercício foi de 9,63 submissões de códigos-fontes.

Ao analisar os resultados dos testes lógicos dos exercícios realizados pelos discentes. O conteúdo dos resultados obtidos para cada caso de teste (passou ou falhou) e incorporados ao *feedback* ipsativo. Observa-se claramente que, muitas vezes, o estudante, principalmente de baixo rendimento, não sabe como dar o próximo passo na correção de erros lógicos pois envia novas soluções que falham no mesmo teste. Apesar disso, existe um ganho discente, empiricamente comprovado, ao fazer a análise qualitativa da evolução dos códigos fontes que ocorreu em 78% dos casos na análise das amostras emparelhadas dos discente que foram até o final da disciplina, ou seja, entre a primeira versão em 24 horas e a versão avaliada houve um progresso com a diminuição e/ou eliminação do erro lógico. Lembrando que, neste estudo empírico, as listas ficaram disponíveis por um longo tempo, até o final do semestre.

6. Discussões e Conclusões

As questões de pesquisas QP1, QP2 e QP3 do ponto de vista da literatura foram analisadas na seção 4 pois, na metodologia adotada, a análise dos resultados destas

questões influenciara o estudo experimental desenvolvido na seção 5. Na meta-análise feita neste artigo, na seção 3, verificou-se que em 26 artigos (12,94% das 201 pesquisas analisadas) usam ferramentas de correção automática de exercícios baseada em testes. Sendo que as baseadas em automações de testes caixa-preta correspondem a 88,23%. O uso por 12,94% dos pesquisadores é devido a diversidades de ferramentas disponíveis para o ensino de programação, em abordagens pedagógicas relevantes, mas não incluídos no escopo deste artigo, tais como, trabalhos que relatam oficinas de jogos, robótica, ferramenta de programação visual, ferramenta de depuração de código e aspectos estritamente motivacionais do ensino.

No Estudo empírico da seção 5 foi abordado também as questões QP2 e QP3 e foi observado que o *feedback* automático fornecido pelo *Sharif-Judge* permite ao menos que o estudante saiba em quais casos de testes o código fonte submetido por ele foi satisfatório e quais não. Isto tem uma relevância na motivação dos estudantes que procuraram ao longo do tempo corrigir seus erros lógicos e verificá-los por testes. Embora nem todos os estudantes submetam os exercícios e a ferramenta não teve um resultado significativo nas evasões discentes nas duas disciplinas, Entretanto, ao fazer a análise qualitativa do código, verifica-se que é estatisticamente significativo da evolução dos códigos fontes, analisando as amostras emparelhadas, feita pelo mesmo discente como resposta a um mesmo exercício, comparando a primeira versão do código-fonte submetido por ele em 24 h (sem erro de sintaxe) e a melhor versão, quase sempre a última submetida pelo estudante, observa-se em 78% dos casos uma melhora dos estudantes que foram até o final das disciplinas. Este resultado é devido a coleta de versões iniciais em 24 h e devido a submissão ser possível por um longo prazo. Em outro contexto de estudo empírico com prazos ou regras de submissões diferentes pode-se encontrar percentuais diferentes de melhoria do código fonte discente.

O ideal é que a automação por testes e a qualidade do conteúdo do *feedback* atendesse a resposta da questão de pesquisa QP3, mas a tecnologia do atual estado-da-arte não está suficiente madura em tempo de escrita deste artigo. Os *feedbacks* idealmente, deveriam explicar o porquê dos erros lógicos, porém no estado atual apresentar problemas, por exemplo, informações incompletas (sem as informações suficiente e necessárias para descrever os erros lógicos para diferentes estudantes em estágios de aprendizagem também diferentes). Alguns estudantes, por constatação *in loco* em sala de aula, consideraram o *feedback* desnecessário ou difíceis de decifrar ou compreender, em último caso, o excesso de falhas pode ser desmotivante ao discente.

Os *feedbacks* ipsativos nos permitiram transpor algumas das limitações sobre *feedbacks* automatizados com mais conteúdo pedagógicos, no caso deste artigo, sobre erros lógicos. A ferramenta *Sharif-Judge* armazena a informações dos resultados da execução dos casos de testes de maneira estruturada em tabelas do SGBD PostgreSQL e, alternativamente em planilhas do Excel usando a opção all_excel da referida ferramenta. A construção de *feedbacks* ipsativos, em nossa abordagem experimental, foi feita baseada nos resultados obtidos na detecção de erros lógicos que pode ser feita por consultas SQL nas tabelas da ferramenta.

Além disso, no estudo empírico foi constatado que o *Sharif-Judge* permite exercitar uma diversidade de habilidades ligadas à programação, tais como, melhoria na compreensão semântica dos enunciados, na percepção e correção dos erros lógicos através da análise dos *inputs* e *outputs* dos testes caixa-preta. Como as versões dos

códigos anteriores são reenviados e são revisadas após os testes lógicos. Caso se ultrapasse a curva de aprendizagem e se faça uma adaptação da ferramenta de correções automáticas aos objetivos didáticos da disciplina elas podem auxiliaram tanto o docente quanto o discente. Do ponto de vista docente, empiricamente foi verificado que a ferramenta apresenta uma curva de aprendizagem mais amena, em comparação a outras ferramentas similares já utilizadas na Universidade Federal de Goiás. A ferramenta possui uma boa documentação para um *software* livre e funcionalidades que proporciona um ganho de produtividade no manuseio pelo docente e *feedback*s automatizados para os estudantes.

As desvantagens é que a ferramenta (versão 1.4.1) não é homologada para OBI e Maratonas de Programação que são bem populares. Fazer *feedbacks* explicando o porquê dos erros lógicos esbarra nas dificuldades inerentes ao processo de submissão às cegas que ainda é um problema em aberto. Testes caixa-preta automatizados têm limitações e sua análise não faz parte do escopo deste artigo. Embora existam várias ferramentas de automação é improvável que uma única ferramenta seja capaz de automatizar todas as tarefas de teste e ainda fornecer bons *feedback* sobre os erros por elas detectados, mas saber os resultados dos testes é importante para os estudantes.

Agradecimentos

Os autores agradecem o apoio dado a pesquisa pela Universidade Federal de Goiás (UFG), Instituto de Informática (INF) da UFG e a Pós-Graduação em Ciências da Computação do INF-UFG.

Referências

Carless, D. (2006). Differing perceptions in the feedback process. Studies in Higher Education, Volume 31, No. 2, April 2006, pp. 219–233. DOI: https://doi.org/10.1080/03075070600572132.

Carvalho, L. S. G., Oliveira, D. B. F, Gadelha, B. F. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE), p. 140-149. DOI: http://dx.doi.org/10.5753/cbie.sbie.2016.140 .

Francisco, R. E., Pereira, Jr. C. X. P. e Ambrósio, A. P. (2016). Juiz Online no ensino de Programação Introdutória – Uma Revisão Sistemática da Literatura. Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE, p. 11-20. DOI: http://dx.doi.org/10.5753/cbie.sbie.2016.11.

Keuning, H., Jeuring, J., Heeren, B. (2016). Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In the 21th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '16), July 11–13, 2016, Arequipa, Peru, p. 41-46. DOI: https://doi.org/10.1145/2899415.2899422.

Li, J., Wong, S.C., Yang, X. et al. (2020) Using feedback to promote student participation in online learning programs: evidence from a quasi-experimental study. Education Tech Research Dev 68, p. 485–510 (2020). DOI: https://doi.org/10.1007/s11423-019-09709-9

Marcolino, A. S. and Barbosa, E. F. (2015). Softwares Educacionais para o Ensino de Programação: Um Mapeamento Sistemático. Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE), p. 190-199. DOI: http://dx.doi.org/10.5753/cbie.sbie.2015.190

Mattos, M. and Pinto, S. C. C. S. (2016). Uma Visão do Ensino de Computação nos Cursos Técnicos Integrados ao Ensino Médio em campi do Instituto Federal do Rio de Janeiro: Uso de Programação no Apoio ao Aprendizado. Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE), p. 688-697. DOI: http://dx.doi.org/10.5753/cbie.wie.2016.688

McConlogue, T. (2020). Assessment and Feedback in Higher Education: A Guide for Teachers. London: UCL Press. ISBN: 9781787353640.

Muuli, E., Tõnisson, E., Lepp, M. et al. (2020) Using image recognition to automatically assess programming tasks with graphical output. Educ Inf Technol (2020). DOI: https://doi.org/10.1007/s10639-020-10218-z.

Naderi, M.: *Sharif-Judge* version 1.4 Documentation. (2020) https://github.com/mjnaderi/Sharif-Judge/tree/docs/v1.4. Acessado em 13/01/2020.

Ossenberg, C., Henderson, A. & Mitchell, M. (2019).What attributes guide best practice for effective feedback? A scoping review. Adv in Health Sci Educ 24, p. 383–401. DOI: https://doi.org/10.1007/s10459-018-9854-x.

Santos, A. C. T., Monteiro, J. A., Machado, K. C. T., Lins, P. R. B., Ramos , T. A. R., Batista, L. V. (2015). Ensino de programação para Olimpíada Brasileira de Informática. Anais do XXI Workshop de Informática na Escola (WIE), p. 122-126. DOI: http://dx.doi.org/10.5753/cbie.wie.2015.122.

Silva, P., Tenório, M. C., Fechine, J., Costa, E. (2015). Um Mapeamento Sistemático sobre Iniciativas Brasileiras em Ambientes de Ensino de Programação. Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE), p. 367-375. DOI: http://dx.doi.org/10.5753/cbie.sbie.2015.367.

Silva, T. R., Medeiros, T. J., Medeiros, H., Lopes, R. and Aranha, E. (2015). Ensino-aprendizagem de programação: Uma Revisão Sistemática da Literatura. Revista Brasileira de Informática na Educação (RBIE), vol. 23, No. 01. DOI: http://dx.doi.org/10.5753/rbie.2015.23.01.182.

Silva, L. T. G. Pensar a Educação Mediada por Tecnologias Digitais (2018). Disponível em http://www.ce.ufpb.br/leppi/contents/documentos/publicacoes/. Acessado em 17/02/2020.

Souza, D. M., Batista, M. H. S., Barbosa, E. F. (2016). Problemas e Dificuldades no Ensino e na Aprendizagem de Programação: Um Mapeamento Sistemático. Revista Brasileira de Informática na Educação (RBIE), vol. 24, No. 01. DOI: http://dx.doi.org/10.5753/rbie.2016.24.1.39.

Valle, P. H. D., Barbosa, E. F., Maldonado, J. C. (2015). Um Mapeamento Sistemático sobre Ensino de Teste de Software. Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE), p. 71-80. DOI: http://dx.doi.org/10.5753/cbie.sbie.2015.71.