Revista Tecnologias na Educação-ISSN: 1984-4751 – Ano 12 -Vol.33- Dezembro/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[2]

RESUMO

Prover *feedback* é uma das maneiras de informar aos estudantes sobre seu desempenho acadêmico. Fornecer *feedback* pode melhorar e reforçar o aprendizado dos estudantes, uma vez que detectar seus erros é o primeiro passo para corrigi-los, sendo 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 a correção de uma grande quantidade de códigos-fonte produzidos pelos alunos, dada a inviabilidade de uma produção em massa de *feedback* manual. Neste artigo, foram analisadas ferramentas de correção automática de exercícios e foi realizado um estudo empírico, com base nos resultados dessa análise, gerando *feedbacks* ipsativos que permitem acompanhar e mensurar o desempenho dos estudantes, examinando-se os códigos-fonte escolhidos especificamente para este fim.

Palavras-chave: *Feedback*. Aprendizagem de Programação. Ferramenta de correção automática, *Online Judge*.

1. Introdução

Feedbacks são importantes em qualquer atividade humana porque explicitam informações sobre o padrão de conduta e 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 (MCCONLOGUE 2020). Em relação à aprendizagem de programação, 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.

Existem ferramentas para auxiliar a detecção de erros lógicos com *feedback* imediato. Algumas destas ferramentas são utilizadas para correção automática de exercícios de programação e as mais populares delas são do tipo *Online Judge* (FRANCISCO et. al. 2018). Ocorre que nenhuma tecnologia, por si mesma, pode equacionar problemas educativos (SILVA 2018), sendo necessário planejar o seu uso em sala de aula e adequá-lo aos objetivos

[1] Mestre em Ciência da Computação pela Universidade de Brasília. Professor do Instituto de Informática da UFG.
[2] Doutora em Educação pela Universidade de Brasília. Professora do Instituto de Informática da UFG.

de aprendizagem propostos na disciplina. Desta forma, é necessário avaliar a ferramenta tecnológica em sala de aula.

A correção automática de exercícios de programação é o principal atrativo do uso de ferramenta *Online Judge* 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-fonte diferentes por disciplina. Da perspectiva do discente, a ferramenta oferece um *feedback* imediato, embora semanticamente pobre para estudantes novatos descobrirem o porquê do erro, porém útil para os estudantes saberem, ao menos, se o código fonte submetido passou ou não nos testes lógicos desenvolvidos pelo professor.

Para entender como estas ferramentas estão sendo usadas no Brasil, foi feita uma Revisão Sistemática da Literatura (RSL) em três importantes bases brasileiras. Foram propostas questões de pesquisa sobre o assunto. Neste artigo, além de entender como os pesquisadores brasileiros estão usando efetivamente estas ferramentas ou como estão usando e fornecendo feedbacks, nós fizemos um estudo empírico que também nos ajudará a responder as questões de pesquisa e avaliar e dar dicas de uso sobre a usabilidade da ferramenta Sharif-Judge que se mostrou produtiva na correção automática de exercícios de programação, mas também apresentou limitações em relação ao conteúdo do *feedback*.

As principais contribuições deste artigo, além das respostas às questões de pesquisa usando o método da RSL, são tanto na forma de conduzir o estudo empírico em sala de aula quanto na métrica usada para analisar os resultados experimentais que permitem *feedbacks ipsativos*. A contribuição secundária é a análise da ferramenta usada no estudo empírico em seus aspectos positivos e negativos.

2. Embasamento Teórico

O embasamento teórico deste artigo visa relatar pesquisas desenvolvidas sobre *feedbacks* educacionais usados como instrumentos pedagógicos e também sobre ferramentas para apoiar o ensino e a aprendizagem de linguagens de programação em sala de aula que possam fornecer ao estudante algum tipo de feedback educacional.

2.1. *Feedbacks* educacionais

A parte referente a *feedbacks* educacionais que estão contidas na RSL serão discutidas na seção 4 pois desta forma nos permite analisar como pesquisados brasileiros aplicam *feedbacks* educacionais para seus estudantes em disciplinas de programação com o auxílio de

ferramentas. Nesta seção discutiremos a construção de *feedback* como instrumento pedagógico em sala de aula, independentemente de estar atrelado a uma ferramenta de correção automática ou semi-automática que gerem dados para a construção do *feedback* fornecido ao discente. Lembramos que a qualidade do *feedback* não está necessariamente ligada à automação e que a qualidade do *feedback* do professor pode estar influenciando a qualidade das soluções discentes, segundo Muuli et al. (2020).

Li *et al.* (2020) afirma que os *feedbacks* são muito importantes para promover a participação dos estudantes e envolvem aspectos, entre outros, referentes à motivação dos alunos, atenção, relevância, confiança e satisfação. Estes aspectos são difíceis de mensurar efetivamente. Esses autores fizeram um *survey*, que é uma pesquisa executada em retrospectiva onde os meios principais para coletar as informações quantitativas e qualitativas são os questionários, uma métrica indireta de coleta de dados.

Apesar da dificuldade de fornecer *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). No caso da pesquisa desenvolvida neste artigo, um estudo empírico foi usado para conduzir o processo de *feedback*. Carless (2006) afirma que o *feedback* é um processo de diálogo 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 à 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.

O *feedback* ipsativo é baseado em uma comparação com o desempenho anterior dos estudantes e vinculado ao progresso de longo prazo (MCCONLOGUE, 2020). Esse tipo de *feedback* pode ter um impacto maior na motivação, pois pode enfatizar as melhorias apresentadas pelo estudante em determinado assunto ao longo do tempo. Isso é particularmente interessante na aprendizagem de programação, pois os estudantes, muitas vezes, ficam desanimados quando não reconhecem o progresso em sua aprendizagem (HUGHES 2011).

2.2. Ferramentas de apoio ao ensino e aprendizagem de programação

A parte referente a ferramentas de apoio ao ensino e aprendizagem de programação, por razões de fluxo do desenvolvimento pedagógico deste artigo, está contida nas análises dentro da RSL na seção 4.

3. Metodologia

A princípio foi feito um embasamento teórico na seção 2 e sinteticamente a metodologia de pesquisa foi dividida em duas partes. Inicialmente foi feita uma RSL (seção 4). Os resultados desta análise permitiram verificar *gaps* que levaram ao desenvolvimento de um estudo empírico, em sala de aula e que foi descrito na seção 5 deste artigo.

4. Revisão Sistemática da Literatura (RSL)

4.1. Questões de Pesquisa

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 se produzir *feedbacks* aos discentes?

4.2 - Processo de busca

O processo de pesquisa escolhido foi a busca usando as opções de buscas disponibilizadas pelas próprias bibliotecas digitais dos anais das conferências e revistas relevantes da área de Informática e Educação em língua portuguesa, publicadas entre 2015 e 2019 (5 anos). As revistas e conferências selecionadas são: Revista Brasileira de Informática na Educação (RBIE), Simpósio Brasileiro de Informática na Educação (SBIE), Workshop de Educação em Computação (WIE). A *string* de busca utilizada foi ensino de programação.

4.2.1 Critérios de Inclusão e Exclusão

Foram examinados documentos de 1 de janeiro de 2015 a 31 de dezembro de 2019 nas bases nacionais supracitadas usando o próprio mecanismo de busca (pesquisa) de cada uma das bases. Os documentos semelhantes que apresentavam a evolução de uma mesma pesquisa

e que continham os mesmos autores tiveram suas versões mais antigas removidas para evitar duplicações. Após isto, os dados extraídos dos documentos selecionados foram sintetizados para análise e foi feita a apresentação dos resultados.

O protocolo de pesquisa inclui *string* de busca, critérios de inclusão e exclusão de artigos. Os critérios de inclusão foram aplicados para selecionar os artigos. Inicialmente, ao executar a *string* de busca em cada um dos mecanismos de busca avançada nas bases de dados, o resultado foi 166 artigos selecionados. Os critérios de inclusão foram: documentos em bases brasileiras pesquisadas, envolvendo aspectos do ensino, da programação e de *feedbacks*. A *string* de busca utilizada é ensino de programação, sem filtros específicos e ainda sem aplicar os critérios de exclusão. Foi considerado todo o texto do artigo em todas as bases de busca nacionais do período proposto usando o mecanismo de busca avançada disponível no próprio site da revista. Como os estudos empíricos foram lidos, foi possível verificar se os pesquisadores usaram algum tipo de ferramenta utilizado em ambiente escolar com finalidade didático-pedagógica. Após esta etapa, foram aplicados os critérios de exclusão. Estes artigos formam a seleção preliminar.

Os critérios de exclusão foram aplicados à seleção preliminar, além dos estudos fora dos critérios de inclusão e dos excluídos pelo filtro, foram retirados artigos derivados da mesma pesquisa (estudos duplicados). Neste caso, o último trabalho da pesquisa foi considerado. Excluiu-se também artigos que relatam ferramenta de Teste de Software não automatizado, artigos que relatam oficinas de jogos, robótica, ferramenta de programação visual, ferramenta de depuração de código, ensino de programação no ensino médio e aspectos estritamente motivacionais do ensino.

Os artigos descartados da seleção final foram analisados pelos revisores de acordo com o julgamento da qualidade do artigo descrito na subseção 4.2.2. Os resultados da seleção final juntamente com outros números da RS estão na subseção 4.2.3 e na seção 4.3.

4.2.2 Critérios de Avaliação da Qualidade usados na Coleta e Análise de Dados

Os critérios referentes à análise de qualidade do artigo (QA) foram: (QA1) Foi discutida no artigo alguma metodologia baseada em testes de softwares automatizados para auxiliar a abordagem de ensino apresentada? (QA2) Foi analisado algum estudo empírico em sala de aula envolvendo avaliação automática ou correção dos exercícios por testes automatizados? (QA3) Foram discutidos aspectos referentes a *feedbacks* automatizados ou semi-

automatizados para os estudantes? (QA4). Foram discutidos problemas e dificuldades do ensino ou aprendizagem de programação?

4.2.3 Resultados da RS

Esta seção resume os resultados obtidos na RS. Aplicando os critérios de inclusão e exclusão, obteve-se na base RBIE o total de 20 artigos, após seleção preliminar restaram 6, foram descartados 3 e na seleção final ficaram 3. Na base SBIE o total foi de 73 artigos, após seleção preliminar restaram 13, foram descartados 6 destes e na seleção final ficaram 7. Na base WIE o total foi 68, após seleção preliminar restaram 12, foram então descartados 5 e na seleção final ficaram 7. O total foi então de 166 artigos, após seleção preliminar 31, foram descartados em seguir 14 e na seleção final foram considerados 17.

As avaliações da qualidade dos estudos reportados foram feitas analisando as 4 questões sobre a qualidade dos artigos da subseção 4.2.2. A análise da qualidade dos 17 artigos escolhidos na seleção final da RS deste artigo pode ser vista na Tabela 1.

Tabela 1 – Mapeamento da Avaliação da Qualidade dos estudos reportados

Estudos	QA1	QA2	QA3	QA4
(MARCOLINO et. al. 2015; SILVA et. al.2015a; SANTOS et. al. 2015)	N	S	N	S
(SOUZA et. al. 2016; MATTOS e PINTO 2016; ARIMOTO e OLIVEIRA 2019; LIMA e COUTINHO, 2019)	N	N	N	S
(SILVA et. al. 2015b)	N	N	S	S
(SOUZA e FAVERO 2015)	S	N	N	N
(FRANCISCO et. al. 2015; CARVALHO et. al. 2016; FRANCISCO et. al. 2018; BRITO e FORTES, 2019)	S	S	S	S
(VALLE et. al. 2015)	S	S	N	N
(SALES et. al. 2016)	N	S	N	N
(BARBOSA et. al. 2017)	N	N	S	N
(SILVA JUNIOR e FRANÇA 2017)	N	S	S	S

Fonte: Autores

4.3. Análise e Discussão sobre as Questões de Pesquisa

4.3.1 Resposta à questão de pesquisa QP1

A QP1 se refere aos pontos fracos e fortes das ferramentas no contexto deste artigo. Os maiores contributos didáticos 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: FRANCISCO et. al. 2018; BRITO e FORTES, 2019) disponibilizado ao discente. A correção é automática e os *feedbacks* imediatos proporcionam uma redução da carga de trabalho docente pois seria humanamente impossível ficar 24h por dia analisando códigos-fonte como a ferramenta Sharif Judge faz.

O ponto fraco das ferramentas, detectados em experimentos pedagógicos, refere-se à qualidade e expressividade do *feedback* automático que se resume, na maioria das ferramentas a uma frase que não oferece informações semânticas sobre 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.

4.3.2 Resposta à questão de pesquisa QP2

A QP2 se refere às 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 a analisar os códigos-fonte 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 cursos tecnológicos em Institutos Federais onde a computação não é atividade fim (MATTOS e PINTO 2016). Perguntou-se aos professores se eles já haviam usado ferramentas ou recursos de programação em disciplina que leciona que os auxiliassem na correção de tarefas discentes, e 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 automática, bem como outras ferramentas didáticas inovadoras, não é apenas pela resistência à inovação em si e a conformidade com o status quo. Há outras questões envolvidas, tais como, suporte institucional e o tempo docente aplicado ao aprendizado da ferramenta, já que toda ferramenta possui uma curva de aprendizagem. Entretanto, o problema de evasão dos cursos de programação é conhecido e estudos indicam que o uso de ferramentas didáticas pode motivar os estudantes (SANTOS *et al.* 2015).

SILVA Jr e FRANÇA (2017) fizeram uma análise comparativa de ferramentas utilizadas no ensino de programação categorizando pela forma de interação textual ou visual e textual ao mesmo tempo. Algumas ferramentas descritas por eles não trabalham com linguagens de programação tradicionais, por exemplo, Portugol (pseudolinguagem que permite desenvolver algoritmos estruturados em português) e Blocos, outra com Java (ferramenta Feeper), com Python (TSTView) e C (SOAP).

4.3.3 Resposta à questão de pesquisa QP3

A QP3 se refere às vantagens de se fornecer *feedbacks* aos discentes. Não foram encontradas discussões avançadas nas bases de dados. A resposta da QP3 é feita tanto pela análise de boas pesquisas sobre o assunto quanto pelas conclusões do estudo empírico baseado na análise dos seus resultados.

Do ponto de vista discente, 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.

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 baseada em testes de software automatizado do tipo caixa-preta

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.

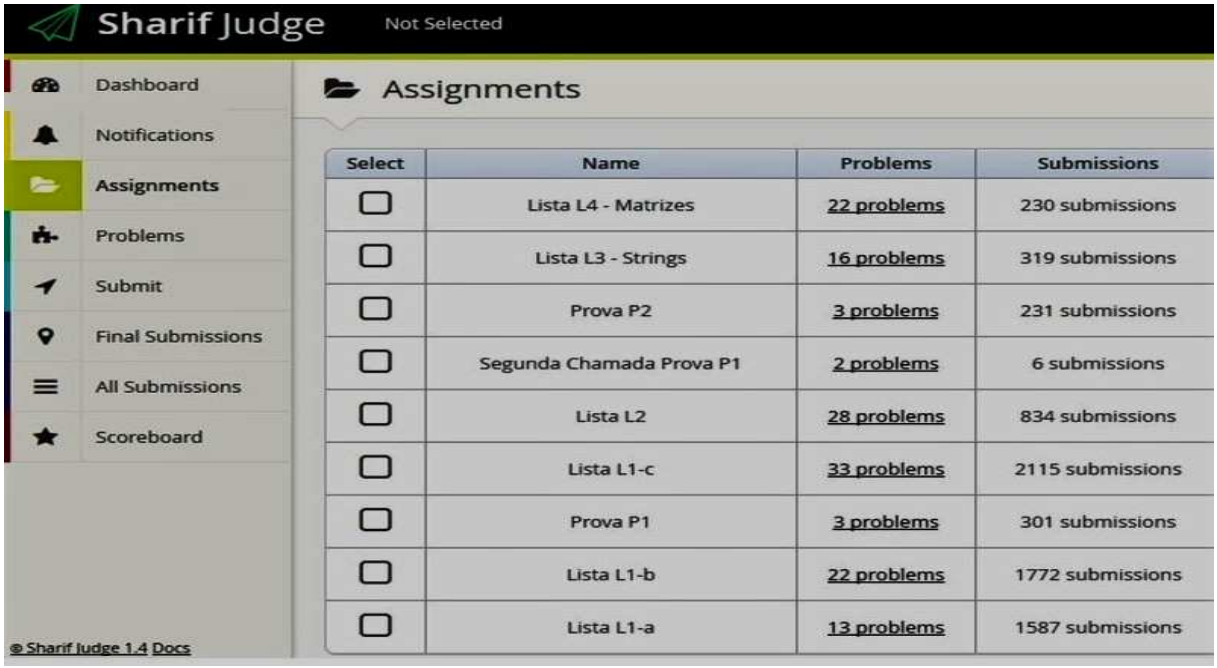
5.1. Descrição do Estudo empírico

O Estudo empírico envolveu estudantes de cursos introdutórios de programação na Universidade Federal de Goiás (UFG) em 2019, um curso com 28 estudantes, o outro com 29 estudantes, logo inicialmente foram 57 estudantes. Para fins do estudo empírico foram consideradas as 6 listas de exercícios com duas avaliações, sendo que a primeira delas houve segunda chamada conforme Figura 1. Foram retirados os estudantes considerados estatisticamente *outliers* que se encaixavam nas condições de trancamentos, desistência sem trancamentos, abandonos da disciplina ou mesmo abandonos após o recebimento dos resultados da primeira avaliação ou a qualquer tempo no decorrer da disciplina, sem qualquer tipo de comunicação formal. Além disso, estudantes com submissão de tarefas sistematicamente incompletas, ou seja, abaixo do mínimo estipulado para que o estudo empírico tivesse estudantes considerados provenientes de uma população normal do ponto de vista estatístico. Este grupo específico de *outliers* é composto de estudantes que não submeteram os códigos-fonte em 50% dos casos para os problemas das listas, ou submeteram os códigos-fonte praticamente em branco ou ainda sem nenhum construto lógico relevante para resolução do problema proposto.

Aplicando os critérios de eliminação de estudantes *outliers* tivemos 37 estudantes participando efetivamente do estudo empírico até a primeira prova e 30 estudantes que foram até o final do estudo empírico.

No estudo empírico desenvolvido neste artigo, tratar os *outliers* é crucial na análise de dados por pelo menos dois motivos: (i) *Outliers* podem contaminar negativamente todo o resultado de uma análise de dados empírica; (ii) o comportamento estatístico dos *outliers* em uma análise pode ser justamente o que está se tentando demonstrar para uma população normal. Exemplificando: se o estudante deixa uma questão na prova em branco, qualquer submissão que ele fizer à posteriori acrescentando algum código e passar em apenas um dos casos de teste do *test suite* será considerado uma melhora no *feedback* ipsativos com amostras emparelhadas do mesmo estudante. *Test suite* ou conjunto de casos de testes é formado por um conjunto especializado de casos de teste agrupados capazes de testar todo um programa logicamente (MALDONADO *et. al.* 2018). No caso deste *paper*, significa testar logicamente um exercício proposto pelo docente aos discentes no contexto da disciplina de programação. Em nosso estudo empírico não houve necessidade do grupo de controle pois as amostras são emparelhadas por estudantes. Todos os estudantes que não são outliers são do grupo experimental pois o feedback ipsativo compara resultados atuais do estudante com os seus resultados anteriores obtidos por ele mesmo na mesma tarefa (MCCONLOGUE 2020).

Figura 1 – Resultados das submissões de problemas no *Sharif-Judge*



Fonte:Autores

5.2. Execução do Experimento

A Figura 1 representa os resultados das submissões do estudo empírico em que foram propostos aos estudantes 134 exercícios diferentes e com assuntos típicos de disciplinas introdutórias de programação no paradigma imperativo usando a linguagem C. Além dos exercícios, foram feitas duas provas e uma prova substitutiva da primeira avaliação perfazendo 8 problemas. Desta forma, foram considerados ao final 142 problemas.

Analisando a Figura 1, observa-se que foram submetidos 6.857 códigos-fonte para resolução de 134 exercícios em 6 listas por todos os estudantes, incluindo os estudantes *outliers* que em algum momento, enquanto estavam efetivamente participando da disciplina, submeteram alguns exercícios das listas. Nas duas avaliações (Prova P1 e Prova P2) foram propostos 6 problemas com 532 submissões.

Em nossa abordagem do estudo empírico, as provas não foram *online*, foram presenciais. A submissão foi feita após as correções das avaliações. O primeiro código submetido na ferramenta Sharif-Judge era o código fonte escrito na respectiva prova, sem erros de sintaxe. Se na prova a questão foi deixada em branco pelo discente, a submissão foi feita apenas com o corpo do método *main* da linguagem C para evitar que o compilador detecte erros de sintaxe.

É importante para verificar os resultados dos testes automatizados que os códigos-fonte não tenham erros de sintaxe, pois nesses casos a ferramenta *Sharif-Judge* ou qualquer ferramenta *online Judge* não pode executar os testes lógicos previamente definidos no *test suite*. O *test suite* e uma solução padrão docente que obrigatoriamente passe pelos casos de testes lógicos do referido *suíte* são feitos pelo *staff* da disciplina antes de abrir as listas de exercícios ou de provas para que os estudantes façam a submissão das suas respostas aos respectivos exercícios. Todos os exercícios submetidos pelos estudantes que passarem em todos os casos de testes do *test suite* são considerados corretos e equivalentes lógicos da solução dos professores, portanto, logicamente corretos.

A primeira amostra submetida de um exercício por um estudante no *Sharif Judge,* incluído no estudo empírico, é considerada a primeira amostra emparelhada na lista. Na Figura 1, a opção "*Final Submissions*" permite coletar os resultados dos testes lógicos aplicados à versão final escolhida pelo próprio estudante para ser avaliado. Esta versão escolhida pelo próprio estudante é a usada para avaliação final do exercício que corresponde à segunda amostra emparelhada.

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

A análise das submissões discentes de códigos-fontes foi feita após a remoção dos estudantes *outliers* de acordo com o descrito na seção 5.1.

5.3.1 Análise das submissões discentes de códigos-fontes antes da primeira avaliação

Os resultados das submissões discentes antes da primeira avaliação, desconsiderando os estudantes *outliers,* incluem as três primeiras listas da Figura 1 (Lista L1-a, L1-b, L1-c). Os estudantes *outliers* submeteram ao todo 876 códigos-fonte nas três listas iniciais. Ao total, os 37 estudantes incluídos no estudo empírico submeteram nas três listas fizeram 4598 (5474-876) submissões de códigos-fonte efetivas para efeito estatístico.

Os 37 estudantes incluídos no estudo empírico até a primeira avaliação não precisavam fazer todos os 68 exercícios das três listas. Eles poderiam descartar 20% dos exercícios, a escolha deles, ou seja, os estudantes poderiam indicar os 14 exercícios que foram descartados por eles. Logo 54 exercícios foram para avaliação no estudo empírico. Embora, em tese, os estudantes devessem submeter apenas códigos-fonte para estes 54 exercícios, na prática, alguns estudantes submeteram códigos-fonte para mais de 54 exercícios. Estes códigos-fonte foram descartados pois estavam em branco. O total de códigos-fonte descartados nestes 14 exercícios não avaliados foram de 570. Logo, para fins estatísticos, restaram 54 dos exercícios que foram efetivamente avaliados nestas três listas iniciais que tiveram 4028 submissões discentes. Desta forma, temos a média de 74,60 (4028 submissões /54 exercícios) códigos-fonte submetidos por exercícios. Evidentemente dos 54 exercícios para avaliação, nem todos os 37 estudantes que submeteram as respostas aos exercícios obtiveram êxito em algum caso de testes lógico do *test suite* de um determinado exercício. No processo de automação dos casos de testes cada caso de teste passa parâmetros de entrada para o programa em teste (*input* do teste lógico) e, também, define qual é o resultado que se espera se o programa estiver logicamente correto (*output* esperado) considerando as linhas afetadas, em tempo de execução, por este mesmo teste, passando em todos os *test suite* . Se ocorrer para cada caso de teste do *output* obtido ao rodar o programa ser o mesmo que o *output* esperado, o programa testado é considerado logicamente correto segundo os os resultados dos testes (MALDONADO *et. al.* 2018).

Os 37 estudantes não submeteram respostas aos 54 exercícios e alguns dos exercícios foram deixados em branco. Foram 29 exercícios diferentes submetidos, em média, por estudante, logo temos um total de 53,70%.

5.3.2 Análise das submissões discentes de códigos-fontes após a primeira avaliação

Os resultados das submissões discentes depois da primeira avaliação referem-se a 37 estudantes, desconsiderando os estudantes *outliers* pós primeira avaliação. Restaram então 30 estudantes que chegaram até o final do estudo empírico aplicando os critérios usados para definir *outliers* na seção 5.1, logo, mais 7 estudantes foram excluídos. Foram submetidos no total 1383 códigos-fonte em três listas após a primeira avaliação conforme Figura 1 (Lista L2, Lista L3 - *Strings*, L4 - Matrizes). O número de exercícios propostos das três listas após a primeira prova foi de 66 exercícios.

Os 30 estudantes restantes não precisavam fazer todos os 66 exercícios das três listas. Eles poderiam descartar, nesta fase, 30% dos exercícios pois os exercícios eram mais complexos e demorados que no início da disciplina. Os estudantes escolhiam os que iriam submeter, ou seja, os estudantes poderiam indicar os 20 exercícios que foram descartados por eles. Logo, 46 exercícios foram para avaliação nesta fase do estudo empírico.

Os 7 estudantes *outliers* entre a primeira e segunda prova submeteram 80 códigos-fonte, logo o total de submissões de códigos-fonte dos 30 estudantes que foram até o final da disciplina após a primeira avaliação foi de 1303 códigos-fonte. Desta forma, temos a média de 28,32 (1303 submissões /46 exercícios) códigos-fonte submetidos por exercícios.

Os 30 estudantes também não submeteram respostas a todos os 46 exercícios avaliados nesta fase. Foram 33 exercícios diferentes submetidos, em média, por estudante, num total de 71,74%.

5.3.3 Análise das submissões discentes de códigos-fontes referentes às provas

Os *outliers* são os estudantes com o comportamento no estudo empírico descrito nas duas seções anteriores (5.3.1 e 5.3.2) acrescido de outras características discutidas nesta seção.

Na Figura 1 observa-se que houve duas avaliações: ProvaP1 (3 problemas) e ProvaP2 (3 problemas) além de uma segunda chamada da primeira prova (2 problemas). Todas as avaliações foram feitas presencialmente em 2019.

A primeira avaliação foi feita por 37 estudantes incluindo 1 estudante que fez a segunda chamada da prova 1. Conforme a Figura 1, a primeira prova tinha 3 problemas. A primeira e a segunda questão foram submetidas por 35 estudantes, a terceira questão por 34 estudantes. Na segunda chamada da prova 1 o estudante submeteu as duas questões. Se todos os estudantes resolvessem todos exercícios, teríamos 108 resoluções (36 estudantes x 3 questões) na primeira avaliação. Na segunda chamada da prova 1 o único estudante que fez submeteu a resposta das duas questões, faltando uma. Se fossem as 3, seriam 101 submissões possíveis, mas como foram mais duas, temos um total de 110 resoluções. Considerando que na primeira chamada 4 questões não foram submetidas e na segunda chamada 1, temos um total de 5 resoluções a menos que o esperado. Logo, a porcentagem média de submissões foi de 95,45%. Observando a Figura 1, temos 301 submissões para a prova 1, com 3 questões por prova, num total de 100,34 submissões, em média, por questão. Na segunda chamada da primeira avaliação houve 6 submissões para 2 problemas, resultando em um total, em média, de 3 submissões por questão. A média de submissões de questões por estudante foi 8,36 questões (301 submissões por 36 estudantes) na primeira avaliação e 3,0 questões na segunda chamada da prova 1.

A segunda avaliação foi feita por 30 estudantes e tinha 3 problemas. O primeiro problema foi submetido por todos os estudantes, o segundo por 27 deles e o terceiro por 6 estudantes. Se todos os estudantes resolvessem todos os exercícios teríamos 90 resoluções (30 estudantes x 3 questões) na segunda avaliação. Como 3 estudantes não submeteram a segunda questão e 24 não submeteram o terceiro problema, temos um total de 27 resoluções a menos que o esperado. Logo, a porcentagem média de submissões foi de 70%. Observando a Figura 1 temos 231 submissões para a segunda prova, com 3 questões por prova, num total de 77 submissões, em média, por problema. A média de submissões de questões por estudante foi 7,7 questões (231 submissões por 30 estudantes) na segunda avaliação.

Em nossa abordagem para o estudo empírico com relação às provas foi que a primeira submissão do exercício pelo estudante fosse a digitação do código fonte escrito na prova. Esta é a primeira amostra. Depois os estudantes teriam que corrigi-lo até passar em todos os testes lógicos ou o melhor que pudessem, mas não poderiam desprezar a primeira versão submetida e iniciar a solução do zero. Durante o prazo em que estivesse aberta a submissão, o estudante poderia submeter tantas versões quanto achasse necessário até produzir o melhor código logicamente possível, passando em todos os casos de teste do *test suite*.

Desta forma, para efeito de avaliação do estudo empírico, na parte relativa às provas, nós obtemos a primeira versão do código fonte original que está na prova e um ou mais códigos-fonte foram submetidos com objetivo de melhorar a acuracidade lógica da solução. Como neste artigo adotamos o *feedback* ipsativo, precisa-se no mínimo de duas ou mais amostras emparelhadas dos mesmos estudantes.

5.3.4 Análise dos resultados das submissões de códigos-fontes discentes

As análises dos resultados das submissões de códigos-fontes discentes estão resumidas nas Tabela 2 e 3. A Tabela 2 representa os resultados obtidos na análise das 6 listas propostas aos estudantes na Figura 1, sendo que 3 delas são antes da primeira avaliação e 3 depois dela.

Tabela 2 – Resultados obtidos das submissões de antes e depois da primeira avaliação

Etapa do Estudos Empírico	**Estudantes sem *outliers***	**(1)**	**(2)**	(3)
Antes Primeira avaliação	37	74,60	54	53,70%
Depois Primeira avaliação	30	28,32	46	71,74%

Legenda da Tabela 2: (1) Média de submissões, sem outliers, por exercício. (2) Número de exercícios examinados nas 3 listas. (3) Percentagem de exercícios submetidos por estudante.

Observe que na Tabela 2 a percentagem de exercícios submetidos por estudantes foi bem menor (53,70%) antes da primeira prova do que depois da primeira avaliação (71,74%). No entanto, a média de submissões, sem estudantes *outliers*, por exercício é bem maior (74,60) antes da primeira prova do que depois (28,32). Isto significa dizer que os estudantes submeteram mais códigos-fonte, em média, por exercício antes na primeira avaliação pois tinham mais dificuldades de obter o resultado correto do ponto de vista lógico, ou seja, passar nos testes lógicos dos exercícios e por isto foram necessárias mais submissões. Os estudantes antes da primeira prova também tentaram resolver uma quantidade percentualmente menor de exercícios próximo à metade (53,70%) enquanto depois da prova, a taxa de submissão de exercícios passou de 70% de submissão (71,74%).

No caso das listas de exercícios, os estudantes, eles mesmos, verificavam o *feedback* ipsativo, analisando o progresso, em detectar e corrigir erros lógicos à medida que foram submetendo soluções discentes analisando os próprios resultados dos testes lógicos automatizados da ferramenta *Sharif Judge*. Em nosso estudo empírico 18,92% dos estudantes desistiram ou abandonaram entre a primeira e a segunda avaliação.

Os resultados referentes às duas avaliações formais e uma prova de segunda chamada é diferente pois incluíram a etapa de correção das provas que foram presenciais por parte do professor. A avaliação dos professores da prova é feita, às claras, analisando a estrutura de dados e a lógica de programação de forma rigorosa e completa, detectando todos os erros lógicos. A submissão automatizada da ferramenta Sharif Judge não fornece automaticamente uma nota e sim os resultados dos testes lógicos. No caso de se ter uma solução discente que passe em todos os testes lógicos de um exercício, a solução é considerada logicamente correta. Não é possível avaliar somente os extremos da nota, por exemplo, atribuir a um exercício nota 10 às soluções discentes que passaram em todos os testes lógicos de um exercício e zero nas demais que falharam em alguns deles. A Tabela 3 representa os resultados obtidos nas avaliações considerando três parâmetros explicados na legenda.

Tabela 3 – Resultados obtidos das submissões de antes e depois da primeira avaliação

Avaliação	Estudantes (sem *outliers)*	(1)	(2)	(3)
P1	36	41,67% (1h 50 min)	8,36	(75%) 3h 03min
2º chamada P1	1	0 % (1h 50 min)	3,00	(75%) 1h 48min
P2	30	63,34% (1h 50 min)	7,70	(90%) 2h 19min

Legenda da Tabela 3: (1) Percentual de Submissões da Primeira Amostra (prova) acima da média (>= 6,0) com correção manual pelo professor. Tempo destinado à Prova. (2) Média de tentativas de submissões da segunda amostra no *Sharif Judge*. (3) Média de percentual de acertos nos casos de testes lógicos. Média do tempo entre a amostra 1 e amostra final.

A Tabela 3 nos mostra para amostra 1 do código fonte que foi feita em sala de aula presencial o percentual de estudantes que tirou acima da média, que é 6 no caso da UFG, foi de 41,67% na prova 1, zero por cento na segunda chamada porque o único estudante que fez obteve uma pontuação abaixo da média. Na prova P2 o percentual acima da média foi de 63,34%. A partir do código fonte da prova escrita, o estudante tinha um prazo para submissão da sua tarefa que deveria ser feita presencialmente no laboratório à tarde (máximo 5h, entre 13h e 18h). O estudante receberia a avaliação no próprio laboratório e teria no máximo 5 horas para tentar melhorar a qualidade lógica do seu código fonte. Em média, os tempos que os estudantes ficaram no laboratório foram na primeira avaliação 3h e 3 min, na substitutiva

da primeira avaliação 1h 48 min e na segunda avaliação 2 h e 19 min. É possível observar uma melhoria tanto na qualidade do código fonte discente quanto no tempo gasto na prova P2.

6. Discussões e Conclusões

As questões de pesquisas QP1, QP2 e QP3 do ponto de vista da RSL foram analisadas na seção 4 pois, na metodologia adotada, a análise dos resultados destas questões influenciara o estudo empírico desenvolvido na seção 5. Na RSL verificou-se que em 12,94% 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 foram abordadas 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 todos os exercícios propostos e também a ferramenta não teve tenha tido um resultado significativo na diminuição do número evasões discentes em relação à evasão média de turmas da universidade, ao fazer a análise qualitativa do código, verifica-se que é estatisticamente significativa a evolução dos códigos-fonte, analisando as Tabelas 2 e 3.

Observa-se que na Tabela 2 o percentual de submissão nas 3 primeiras listas antes da primeira avaliação foi de 53,70% e depois de 71,74%. Isto se explica porque alguns estudantes que estavam submetendo com uma frequência menor que a média abandonaram a disciplina. O percentual de abandono foi de 18,92% entre a primeira e a segunda avaliação.

Na Tabela 3 pode-se comparar o desempenho médio do estudante na primeira avaliação com os resultados obtidos por ele mesmo na segunda avaliação. É possível observar uma melhoria tanto na qualidade do código fonte discente quanto no tempo gasto para resolvê-la, pois na segunda avaliação houve um acerto na correção dos códigos de 90% em 2h 19 min e na primeira avaliação o acerto foi de 75% em 3h 03min. Considerando que os

tópicos explorados ao final da disciplina são mais complexos que no seu início, o ganho é empiricamente significativo.

O ideal é que a automação por testes e a qualidade do conteúdo do *feedback* respondesse as indagações da questão de pesquisa QP3 de forma definitiva. Existem sim, vantagens de ser produzir *feedbacks* aos discentes usando automação de testes, mas a tecnologia atual não está suficientemente madura para uma automação perfeita. Os *feedbacks* para estudante novatos, idealmente, deveriam dar dicas mais fortes do porquê dos erros lógicos, porém ferramentas de correção automática do tipo *Online Judge* focam em usuários avançados. Por exemplo, novatos podem considerar o texto do *feedback* imediato com informações incompletas (sem as informações suficientes 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 que o *feedback* informa que foram detectados erros lógicos, mas não o porquê destes erros. O excesso de falhas ao submeter um código fonte sem o estudante entender o porquê está errado pode ser desestimulante ao discente de baixo desempenho e, ao mesmo tempo, motivante para estudantes que apreciam desafios.

Os *feedbacks* ipsativos nos permitiram transpor algumas das limitações sobre *feedbacks* automatizados com mais conteúdos pedagógicos, no caso deste artigo, sobre erros lógicos. A ferramenta Sharif-Judge armazena 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 partir da planilha do Excel gerada pela ferramenta foi feita a análise da última coluna da Tabela 3.

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* obtidos ao reenviarem novos códigos fonte, melhorando assim, o desempenho discente.

Do ponto de vista docente, o Sharif-Judge é mais fácil de usar do que outras ferramentas *Online Judge* pois os docentes não têm que lidar com regras específicas implementadas em ferramentas homologadas, como por exemplo as usadas na Olímpiada Brasileira de Informática (OBI) ou Maratona SBC (Sociedade Brasileira de Computação) de Programação. A ferramenta possui uma boa documentação para um *software* livre e funcionalidades que proporcionam um ganho de produtividade no manuseio pelo docente.

Alguns exemplos são os resultados obtidos que podem ser exportados no formato da planilha Excel, além de *feedbacks* automatizados para os estudantes mais expressivos por apresentarem os resultados de cada um dos casos de testes lógicos executados pela ferramenta.

Uma das desvantagens é que a ferramenta Sharif-Judge (versão 1.4.1) não é homologada para olimpíadas ou maratonas de programação. É comum universidades terem times que participam desses tipos de eventos. Neste caso, deve-se manter uma ferramenta *Online Judge* homologada e a ferramenta Sharif-Judge. É importante destacar que fazer *feedbacks* explicando o porquê dos erros lógicos esbarra nas dificuldades inerentes ao processo de submissão às cegas de códigos-fontes que é baseado em resultados de testes automatizados do tipo caixa-preta. A automação de testes *software* é um problema em aberto e as futuras pesquisas, nesta área, podem trazer novas funcionalidades às ferramentas atuais.

7. Referências Bibliográficas

ARIMOTO, M. M.; OLIVEIRA, W. T. Dificuldades no Processo de Aprendizagem de Programação de Computadores: Um Survey com Estudantes de Cursos da Área de Computação. Anais do XXVII Workshop Sobre Educação em Computação (WIE), p. 244-254, 2019. ISSN 2595-6175. Disponível em: DOI: https://doi.org/10.5753/wei.2019.6633.

BARBOSA, A. A.; COSTA, E. B; PATRICK H. B. Uma abordagem adaptativa para gerar agrupamento de códigos em disciplinas de programação introdutória. Anais do XXVIII Simpósio Brasileiro de Informática na Educação (SBIE 2017), p. 1427-1436, 2017.

BRITO, P. S. S. e FORTES, R. S. O uso de corretores automáticos para o ensino de programação de computadores para alunos de engenharia. Anais do XXX Simpósio Brasileiro de Informática na Educação (SBIE 2019), p. 449-458, 2019. Disponível em: DOI: 10.5753/cbie.sbie.2019.449.

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

CARVALHO, L. S. G.; OLIVEIRA, D. B. F; GADELHA, B. F. 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, 2016. Disponível em: DOI: <http://dx.doi.org/10.5753/cbie.sbie.2016.140>.

FRANCISCO, R. E.; PEREIRA, Jr., C. X. P. e AMBRÓSIO, A. P. 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, 2016. DOI: http://dx.doi.org/10.5753/cbie.sbie.2016.11.

FRANCISCO, R. E., AMBRÓSIO, A. P. L., PEREIRA Júnior, C. X. e FERNANDES, M. A. (2018). Online judge in CS1 education - Lessons learned and proposal of a tool (Juiz online no ensino de CS1 - lições aprendidas e proposta de uma ferramenta). Brazilian Journal of Computers in Education (Revista Brasileira de Informática na Educação - RBIE), 26(3), 163-179. Disponível em: DOI: <10.5753/RBIE.2018.26.03.163>

HUGHES, G. Towards a personal best: a case for introducing ipsative assessment in higher education. In Studies in Higher Education. Volume 36, Issue 3, April 2011, pp. 353 – 367. Disponível em: DOI < 10.1080/03075079.2010.486859.>

LI, J.; WONG, S.C.; YANG, X. *et al.* Using feedback to promote student participation in online learning programs: evidence from a quasi-experimental study. Education Tech Research Dev 68, p. 485–510, 202). Disponível em: DOI:< https://doi.org/10.1007/s11423-019-09709-9 >

LIMA, E.; COUTINHO E. F. Uma Análise sobre o Desempenho de Alunos de Graduação em Disciplinas Iniciais de Programação. Anais do XXX Simpósio Brasileiro de Informática na Educação (SBIE 2019), p. 1876-1885, 2019. Disponível em:DOI: <10.5753/cbie.sbie.2019.1876>.

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. Disponível em: 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. Disponível em: DOI: <http://dx.doi.org/10.5753/cbie.wie.2016.688>

MALDONADO, J.; DELAMARO, M.; VINCENZI, A. M. R. [et. al.] Automatização de teste de software com ferramentas de software livre. 1. ed. - Rio de Janeiro: Elsevier, 2018. ISBN 978-85-352-8728-8.

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

MUULI, E.; TÕNISSON, E.; LEPP, M. [*et. al*]. Using image recognition to automatically assess programming tasks with graphical output. Educ Inf Technol (2020). Disponível em: DOI: <https://doi.org/10.1007/s10639-020-10218-z.>

SANTOS, A. C. T.; MONTEIRO, J. A.; MACHADO, K. C. T.; LINS, P. R. B.; RAMOS, T. A. R.; BATISTA, L. V. Ensino de programação para Olimpíada Brasileira de Informática. Anais do XXI Workshop de Informática na Escola (WIE), p. 122-126., 2015. Disponível em: DOI: <http://dx.doi.org/10.5753/cbie.wie.2015.122>

SILVA JUNIOR, S. M.; FRANÇA, S. V. A. Programação para todos: Análise Comparativa de Ferramentas Utilizadas no Ensino de Programação. 25º WEI - Workshop sobre Educação em Computação (WEI), p. 2199-2108, 2017, São Paulo. ISSN 2595-6175. Disponível em: DOI: <https://doi.org/10.5753/wei.2017.3535>

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

SILVA, P.; Tenório, M. C.; FECHINE, J.; COSTA, E. 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, 2015. Disponível em: DOI: <http://dx.doi.org/10.5753/cbie.sbie.2015.367>.

SILVA, T. R.; MEDEIROS, T. J.; MEDEIROS, H.; LOPES, R. e ARANHA, E. Ensino-aprendizagem de programação: Uma Revisão Sistemática da Literatura. Revista Brasileira de Informática na Educação (RBIE), vol. 23, No. 01, 2015. Disponível em: DOI: <http://dx.doi.org/10.5753/rbie.2015.23.01.182>.

SOUZA, D. M.; BATISTA, M. H. S.; BARBOSA, E. F. Problemas e Dificuldades no Ensino e na Aprendizagem de Programação: Um Mapeamento Sistemático. Revista Brasileira de Informática na Educação (RBIE), vol. 24, No. 01., 2016. Disponível em: DOI: <http://dx.doi.org/10.5753/rbie.2016.24.1.39>.

VALLE, P. H. D.; BARBOSA, E. F.; MALDONADO, J. C. Um Mapeamento Sistemático sobre Ensino de Teste de Software. Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE), p. 71-80, 2015. Disponível em :

DOI: <http://dx.doi.org/10.5753/cbie.sbie.2015.71>

Recebido em Dezembro 2020

Aprovado em Dezembro 2020