Qual tipo de feedback ajuda os estudantes novatos em programação? Um estudo exploratório com múltiplos feedbacks em um curso introdutório de Python

Luis Gustavo J. Araujo
IFBA – Instituto Federal de Educação Ciência e Tecnologia da Bahia
Jacobina, Bahia, Brasil
luis_araujo@ifba.edu.br

Christina von Flach
UFBA – Universidade Federal da Bahia
Salvador, Bahia, Brasil
flach@ufba.br

Roberto A. Bittencourt
University of Victoria
Victoria, BC, Canada
PGCC/UEFS
Feira de Santana, Bahia, Brasil
rbittencourt@uvic.ca

RESUMO

Compreender as mensagens de erro emitidas por compiladores e interpretadores costuma ser uma das barreiras encontradas por estudantes novatos em programação. Em geral, mensagens de erro são padronizadas e agnósticas ao seu público-alvo, sendo pouco adequadas para programadores menos experientes. Além disso, estudantes costumam ter dificuldades quanto a reconhecer que a solução proposta está correta. Este artigo apresenta um estudo de caso exploratório realizado em um curso de programação em Python com estudantes novatos fazendo uso do ambiente de aprendizagem PEEF com o objetivo de analisar e avaliar o uso e eficácia de seus feedbacks. Durante o curso, os estudantes tiveram acesso a múltiplos tipos de feedback: mensagens de erro melhoradas pré-cadastradas, testes unitários e chat. Os resultados demonstram que os estudantes assimilaram o conteúdo, se sentiram mais confiantes e familiarizados em resolver problemas, utilizar testes unitários e ler mensagens de erro. O estudo demonstrou ainda uma predileção dos estudantes por mensagens melhoradas de erro e testes unitários. Os testes unitários foram utilizados em 83,6% dos projetos. Destes, 90,5% dos projetos obtiveram sucesso nos testes. Quanto ao uso de mensagens melhoradas pré-cadastradas, o estudo demonstrou uma cobertura de 81,1% dos erros emitidos. Percebeu-se que as mensagens melhoradas contribuíram para a solução das atividades, porém 54% das mensagens não colaboraram para a solução do problema. Percebeu-se ainda que o chat é pouco explorado pelos estudantes. Por fim, observou-se que o uso de testes e mensagens melhoradas colaboram positivamente para a aprendizagem, no entanto o método de mensagens pré-cadastradas não se mostra flexível às diversas variações de erros, levando a uma assertividade baixa.

CCS CONCEPTS

• **Social and professional topics** → Computing education.

PALAVRAS-CHAVE

Mensagem de erro melhorada, Python, ensino de programação.

EduComp'24, Abril 22-27, 2024, São Paulo, São Paulo, Brasil (On-line)

1 INTRODUÇÃO

Apreender programação pode ser um desafio para estudantes novatos. Dentre as dificuldades encontradas, podemos citar as relacionadas à compreensão dos erros reportados e entendimento sobre a conclusão de tarefas. No primeiro caso, estudantes costumam enfrentar problemas para entender as mensagens de erro emitidas por compiladores e interpretadores [11] tornando-se barreiras para os estudantes novatos [5]. No segundo caso, os estudantes devem aguardar o feedback do professor que, em turmas grandes, torna-se um desafio [12]. Dado que feedback é um elemento efetivo para aprendizagem de programação [12], diversos trabalhos têm abordado temas relacionados a feedback para estudantes.

Os trabalhos de Becker et al. [5], Kohn and Manaris [11] e Charles and Gwilliam [6] tratam da emissão de mensagens melhoradas e resultados positivos em relação à quantidade de execuções e erros. Quanto ao uso de testes, o trabalho de Manzoor et al. [13] apresenta uma abordagem que utiliza testes unitários em uma disciplina introdutória. Os resultados mostram que os estudantes concordam que o uso de testes ajuda a resolver os problemas. No entanto, além dos resultados dos testes, outros aspectos em relação ao uso de conceitos devem ser considerados [7]. Por fim, outros pesquisadores utilizam *chatbots* como tutores online para emitir informações sobre as tarefas e como resolvê-las.

O estudo de Essel et al. [10] aponta para melhoria do escore e desempenho acadêmico dos estudantes que utilizam um assistente online. Os autores apresentam uma avaliação positiva para o uso do *chatbot* KNUSTbot, apontando mudanças significativas quanto ao escore e desempenho acadêmico. Por outro lado, Verleger and Pembridge [16] apresentam o EduBot em um curso introdutório de computação e indicam que a ferramenta não obteve um bom desempenho e, por isso, teve uma baixa adoção durante o curso.

Embora existam estudos sobre o fornecimento de feedback de modo isolado, este tema ainda carece de mais estudos que abordem questões sobre múltiplos tipos de feedback, a saber: qual o comportamento dos estudantes quando se oferece múltiplos tipos de feedback? Qual o tipo de feedback preferido pelos estudantes? Qual o nível de eficácia desses feedbacks? Qual a percepção dos estudantes sobre como esses feedbacks colaboram para a aprendizagem?

Diante disso, idealizou-se um estudo de caso [18] exploratório com o objetivo de compreender o uso e eficácia de múltiplos tipos de feedback em um curso introdutório em Python. Os tipos de feedback utilizados durante o curso são: testes unitários, mensagens melhoradas e *chatbot*. O objetivo deste estudo é analisar

o comportamento e a percepção dos estudantes diante de múltiplos feedbacks. Adicionalmente, visa-se avaliar a eficácia desses feedbacks quanto à reação dos estudantes, durante a realização das tarefas de programação.

Este trabalho está organizado em seis seções. A primeira seção apresenta uma introdução sobre o tema e os objetivos do trabalho; a segunda apresenta os trabalhos relacionados; a terceira apresenta a metodologia utilizada. O estudo de caso é apresentado na quarta seção. Na quinta seção são apresentados os resultados e discussões. Por fim, na sexta seção são abordadas as considerações finais.

2 TRABALHOS RELACIONADOS

Becker et al. [5] apresentam um estudo controlado realizado com dois grupos de estudantes utilizando uma ferramenta própria (Decaf). Decaf é um editor Java que possui dois modos de uso, o *pass-through* e o *enhanced*. O primeiro emite apenas a mensagem de erro original e o segundo a mensagem original acompanhada da mensagem melhorada. O Decaf possui 30 mensagens construídas com base nos princípios apresentados em [15]. A lista de erros foi construída com base em trabalhos anteriores [3]. Aproximadamente 100 estudantes participaram do estudo controlado conduzido pelos autores, durante a disciplina CS1. O primeiro grupo teve acesso ao ambiente no modo *pass-through*, o segundo grupo teve acesso ao modo *enhanced* do Decaf. Como resultado, os autores sinalizam que foram gerados 48.800 erros de 74 tipos distintos. Os resultados demonstraram que o grupo de intervenção obteve 35% menos erros que o grupo de controle e as 30 mensagens melhoradas representaram uma cobertura de 78,9% dos erros durante o curso. Constatou-se uma diferença significativa entre o número total de erros e o número de erros por estudantes entre os dois grupos.

Kohn and Manaris [11] apresentam o TigerJython, um ambiente focado em mensagens de erro. TigerJython conta com um *parser* para reconhecer um conjunto de erros, um *debugger* e bibliotecas multimídias com visualização de objetos gráficos. Para os autores, a baixa qualidade das mensagens de erro é frequentemente notada por outros trabalhos e embora a sintaxe Python seja adequada aos novatos, muitas vezes podem ser vagas e errôneas. Foi questionado a 81 estudantes, que utilizaram o ambiente, como as mensagens do TigerJython ajudaram no seu processo de aprendizagem utilizando uma escala Likert de 1 a 5. As respostas tiveram uma média de 3.30 e desvio padrão 1.2. Os autores fizeram o mesmo questionamento para 53 estudantes não nativos e a média das respostas foi 4.04 com desvio padrão 1.1, demonstrando uma adequação a este público. Adicionalmente, foram avaliados ainda 4.000 programas com erros sintáticos e cerca de um terço foram classificados como erro superficial. 95% desses erros foram resolvidos pelos estudantes.

Charles and Gwilliam [6] apresentam a ferramenta Error Explaner que visa melhorar mensagens de erro em Python por meio da transcrição das mensagens em texto plano no idioma inglês. Error Explaner foi idealizada para ser utilizada no Jupyter notebooks e sua mensagem apresenta a descrição do erro, informações de como proceder e uma lista de causas comuns. A limitação desta ferramenta está na lista de apenas 12 erros cadastrados. Os autores realizaram um estudo como 105 estudantes, em um curso de 12 semanas. Os autores apontam como resultados que 91% dos estudantes sabiam que a mensagem indicava o local do erro, 81% do estudantes perceberam que a mensagem continha o tipo do erro, 35% entenderam que a mensagem apresentava os passos para correção. Aproximadamente dois terços dos estudantes indicaram que o Error Explaner ajudou a resolver problemas em 25% das vezes. Um terço dos estudantes indicou que a ferramenta os ajudou em 50% das vezes. Adicionalmente, o estudo revelou que o uso da ferramenta reduziu significantemente a ansiedade e frustração dos estudantes.

No entanto, alguns pesquisadores apontam para a ineficácia do uso de mensagens de erro melhoradas, como é o caso do estudo apresentado por Denny et al. [9]. Os autores implementaram um sistema de mensagens melhoradas para a linguagem Java na plataforma CodeWrite utilizando análise estática com uso de expressões regulares. Este sistema foi capaz de reconhecer os tipos de erro de 92% das submissões. O estudo controlado foi realizado com dois grupos, não sendo identificadas diferenças significativas para o número de erros obtidos de modo consecutivos, o número de submissões que não compilam e o número de tentativas necessárias para solucionar o problema.

Manzoor et al. [13] apresentam outra solução para integrar Jupyter Notebooks à autoavaliação, visando promover feedback imediato aos estudantes com uso de testes unitários. Os autores utilizaram o ambiente Web-CAT para fornecer feedback aos estudantes por meio da criação de uma extensão do navegador (Auto-Gradind Jupyter) que submete a solução criada no Jupyter Notebook ao Web-CAT, além de abrir uma página web para exibir o escore e feedback relacionado. Foram realizadas duas sessões do curso de Análise de Dados e Visualização, cada sessão contou com 55 estudantes. As sessões foram avaliadas por meio de duas surveys, uma para os estudantes e outra para os instrutores, obtendo 94 respostas completas.

Os resultados demonstram que 88% (sessão 1) e 89,7% (sessão 2) dos estudantes concordam que o feedback imediato melhora a sua performance. A média de submissões por estudantes foi 11,4 e 14% dos estudantes fizeram mais de 20 submissões para melhorar seus escores. 75% e 91,17% dos estudantes, nas sessões 1 e 2 respectivamente, afirmam que recomendaria o uso da ferramenta para outros instrutores A maioria dos estudantes afirma que os feedbacks contribuíram para melhorar o seu escore, outros afirmam que o feedback deveria ser mais específico. 5 de 4 instrutores afirmam que o uso do Auto-Gradind Jupyter diminuiu sua carga de trabalho. Todos os instrutores afirmam que a solução apresentada é melhor que nbgrader (um *autograding* criado pelos autores do Jupyter Notebook). No entanto, os instrutores indicam que a criação das atividades é complicada e que o uso de interface de usuário poderia melhorar este aspecto.

Latih et al. [12] apresentam um estudo utilizando o sistema PC2 em um curso introdutório em programação. PC2 é um juiz online que permite ao estudante submeter soluções de modo concorrente dentro de uma rede de participantes. O juiz permite recompilar, executar, visualizar o código e o resultado, além de enviar feedback para o estudante. A plataforma possui 5 tipos de feedback: Sim (Aceito), Não (Resposta Incorreta), Não (Erro de Compilação), Não (Erro em tempo de execução) e Não (Erro no formato da saída). Os autores desenvolveram um estudo com 162 estudantes novatos e realizou coleta de dados por meio de questionário para capturar a percepção dos estudantes.

Os resultados demonstram que 90.8% dos estudantes entenderam o objetivo do uso do PC2. 54,9% afirmam que o uso da ferramenta

torna a submissão da atividade mais fácil. 58.1% dos estudantes entenderam os feedbacks dados e apenas 13,6% afirmaram que não, sendo o erro em tempo de execução (*RunTime Error*) o tipo do erro com feedback mais confuso. 58% dos estudantes afirmaram que os feedbacks dados ajudaram a corrigir os seus programadas. A maioria dos estudantes indicam que sempre se sentiram seguros de que seus programas não continham erros (69,7%) e que produzem resultados corretos (76,5%). Por fim, 24.1% dos estudantes se sentem sempre seguros de que são capazes de solucionar todas as questões durante o laboratório, 38,9% afirmam que não são capazes de resolver todas as questões.

No entanto, da Silva et al. [7] indicam que passar nos testes não é suficiente. Os autores investigam se códigos, ditos corretos por um *autograder*, apresentavam comportamentos que poderiam indicar falhas na aprendizagem dos conceitos abordados em disciplinas introdutórias de programação. No total, 2441 códigos foram analisados e, ao término, foi gerada uma lista de 45 programas que foram submetidos a especialistas. Os professores avaliaram a gravidade dos 45 programas. Como problemas mais graves, destaca-se a sobrescrita do iterador em um loop, o uso de while desnecessário, uso de while como condicional, não uso de condicionais encadeadas, laço for utilizado em substituição ao laço while e funções acessando variáveis fora do escopo. Assim, embora o uso de testes seja positivo para fornecer feedback imediato, melhorando o desempenho dos estudantes, outros aspectos devem ser considerados no momento da instrução.

Essel et al. [10] apresentam em seu trabalho o *chatbot* KNUSTbot utilizando em um curso de programação multimídia na Universidade KNUST. Este assistente permite que os estudantes aprendam e reflitam sobre programação multimídia via interação com o chat. Os autores realizaram um quase-experimento com testes pré- e pós-intervenção combinados com dados qualitativos. Ao total, 68 estudantes participaram do curso de 16 semanas. Os estudantes foram divididos em 2 grupos com amostra representativa da população. Os resultados demonstram que os picos de uso do *chatbot* são no início do curso, quando os estudantes estão se ambientando, e no final, por conta dos exames finais. Observou-se que o grupo experimental obteve um escore (81,1) significativamente maior que o grupo controle (65,2). Mudanças significativas foram observadas quanto a performance acadêmica entre estudantes dos dois grupos.

Verleger and Pembridge [16] apresentam o EduBot, um *chatbot* baseado em Inteligência Artificial desenvolvido pelos autores como um *plug-in* do sistema de gestão aprendizagem (LMS) Canvas. Quando o estudante interage com o EduBot, este consulta a solução em uma lista de perguntas e respostas. Caso não encontre correspondência, envia uma mensagem ao usuário e um e-mail ao professor. O professor pode responder à pergunta que é enviada ao estudante e inserida na base de dados. O EduBot foi integrado em um curso de introdução a Computação para engenheiros com MATLAB. Apenas 33% das perguntas não encontraram respostas cadastradas; 10% das respostas foram sinalizadas como não úteis (não ajudou o estudante a resolver o problema); 16,7% das respostas foram assertivas e 8,3% foram parcialmente assertivas.

O estudo revela que a interação inicial dos estudantes é baseada na curiosidade e que quando não recebem um feedback imediato costumam realizar pesquisas em motores de busca. No entanto, nenhum estudante afirmou que poderiam obter feedbacks mais precisos que o EduBot nestes motores. Os estudantes indicaram ainda que, quando eles tinham dúvidas especificas, preferiam perguntar ao instrutor em vez de escrever suas dúvidas no *chatbot*. Por fim, os estudantes afirmam que o EduBot deveria ter mais informações sobre o contexto e exemplos, informações que eles não poderiam obter facilmente por meio de pesquisas. No entanto, a própria baixa participação dos estudantes impediu a criação de uma base de dados mais abrangente [16].

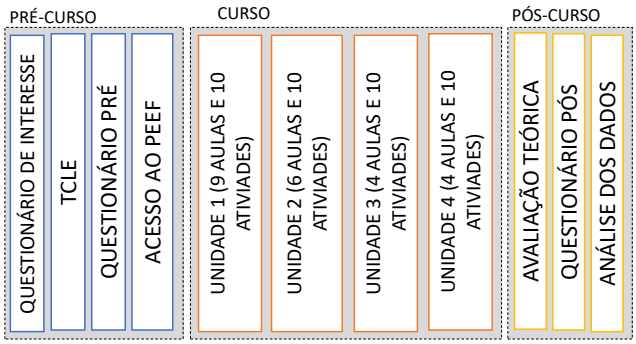

Figura 1: Planejamento das Etapas do Estudo

3 ESTRATÉGIA DE PESQUISA

Um estudo de caso sobre feedbacks para estudantes novatos foi realizado, tendo em vista o caráter exploratório do estudo e a análise dos fenômenos em um curso de programação [18].

O planejamento do estudo consistiu em três etapas (Figura 1). Na primeira, a etapa de pré-curso, foi enviado o questionário de interesse para diversos estudantes. Em seguida, foram enviados o Termo de Consentimento e Livre Esclarecimento (TCLE) e o Questionário pré-intervenção para os interessados. Por fim, foi fornecido o acesso à plataforma PEEF.

O questionário pré-intervenção serviu como filtro para selecionar os estudantes com o perfil especificado. Após essa etapa, foi realizado o curso de 4 semanas. Após o curso, foi realizada a etapa pós-curso com uma avaliação teórica sobre programação em Python e o questionário pós-intervenção. Este último resgatou questões do questionário pré-intervenção com o objetivo de compreender mudanças após o curso. Além disso, neste último questionário, foram adicionadas perguntas sobre o curso e o comportamento sobre estudantes diante dos feedbacks fornecidos.

3.1 Coleta e Análise dos Dados

O processo de coleta de dados deu-se por meio dos formulários pré- e pós-intervenção e por meio da plataforma PEEF. Estes questionários tiveram o objetivo de capturar a percepção dos estudantes sobre o nível de programação, conhecimento em Python, proficiência em inglês, experiência com mensagens de erro e testes unitários. O questionário pós-intervenção visou ainda captar respostas sobre como o ambiente utilizado, as mensagens de erro, o chat e os testes unitários ajudaram os estudantes.

O ambiente utilizado no estudo de caso possibilitou o armazenamento dos dados sobre as execuções dos estudantes, erros obtidos, testes realizados, testes que passaram ou falharam, dentre outros dados. Os dados foram armazenados em um banco de dados SQL e foram extraídos por meio de consultas SQL e scripts em PHP, dentro do ambiente PEEF. Os dados foram analisados utilizando estatística descritiva e inferencial.

Para a análise da emissão das mensagens, nós avaliamos as execuções em pares consecutivos para cada projeto de cada estudante. Sendo E um conjunto de execuções subsequentes e e_1 a primeira execução e e_2 a segunda, os pares foram formados da seguinte forma: (e_1, e_2), $(e_2, e3)...(e_{n-1}, e_n)$. Visou-se identificar se, após uma mensagem melhorada de erro, o estudante corrigia o problema ou não. Em caso negativo, busca-se distinguir entre as execuções que obtêm o mesmo erro e as que obtêm um novo erro. Quanto aos testes, também foram criados pares consecutivos de testes, visando saber o quantitativo de testes que obtiveram sucesso após falha e testes que continuaram falhando. As conversas com o chat também foram analisadas e classificadas de acordo com o comportamento dos estudantes.

4 ESTUDO DE CASO

Esta seção apresenta o estudo de caso exploratório realizado. Serão apresentados os participantes, o planejamento do curso, as ferramentas, linguagens e feedbacks utilizados.

4.1 Participantes

Para a seleção dos estudantes participantes do estudo de caso foi realizada divulgação em grupos de estudantes de uma Universidade privada do Estado da Bahia. Quarenta e quatro estudantes responderam ao questionário sobre a lista de interesse para o curso. Foram enviados o TCLE e um questionário pré-intervenção para todos os interessados no curso. No total, 25 estudantes responderam ao questionário e fizeram a devolutiva do TCLE. Foram enviadas as credenciais de acesso à plataforma, assim como informações gerais sobre o curso para todos os 25 estudantes. Apenas 17 estudantes responderam a este e-mail. Por fim, 12 estudantes participaram do curso até o seu término.

Todos os 12 participantes foram do sexo masculino. É importante observar que a participação feminina foi incentivada durante a divulgação do curso. Dos 25 estudantes que preencheram o questionário pré-intervenção, quatro eram do sexo feminino. Todos os 12 participantes eram estudantes do ensino superior. Os critérios para participação foram: i) nunca ter cursado disciplinas de programação; ii) estar matriculado em uma disciplina de programação inicial ou iii) já ter cursado uma disciplina inicial de programação, tendo concluído com reprovação ou abandonado.

O questionário pré-intervenção visou saber o nível de programação dos estudantes. Um (1) estudante sinalizou nunca ter tido contado com algoritmos (8,3%). Outro estudante sinalizou estar aprendendo no semestre do curso (8,3%). Seis estudantes afirmaram que conseguiam resolver alguns problemas (50%) e quatro estudantes afirmaram já ter cursado disciplinas de algoritmos, porém não tinham muito conhecimento sobre (33,4%). Sobre a linguagem Python, seis estudantes não tinham conhecimento sobre (50%), três estudantes sabiam apenas instruções de entrada, saída e operações aritméticas, um estudante teve um contato anterior com Python, conhecendo também condicionais e dois estudantes conheciam também loops em Python.

Buscamos saber o nível de proficiência em inglês dos estudantes. 58,3% dos estudantes afirmaram precisar de dicionários para ler textos em inglês, 16,6% dos estudantes indicaram ter nenhum conhecimento em inglês, outros dois estudantes (16,6%) afirmaram conhecer apenas palavras relacionadas à programação e apenas um (1) relatou ter proficiência em inglês. Sobre a experiência com mensagens de erro, dois estudantes sinalizaram que normalmente não lêem as mensagens, pois não entendem. Dois estudantes sinalizaram que costumam ler, mas não entendem a mensagem. Três estudantes afirmaram que entendem a mensagem facilmente e conseguem resolver o problema. Outros cinco estudantes afirmaram não ter uma prática constante quanto à leitura da mensagem.

Por fim, buscamos saber o nível de conhecimento sobre testes unitários. Dez estudantes afirmaram não saber do que se tratava (83%). Um (1) afirmou saber o que são testes unitários, mas não utiliza. Um (1) estudante afirmou tem familiaridade com testes, mas não utiliza para estudos.

4.2 Ferramentas e Planejamento do Curso

O curso foi executado no segundo semestre do ano de 2021. Os estudantes da universidade em que o estudo foi realizado estavam em aula online desde 2020, devido à pandemia da COVID-19. O curso contou com quatro semanas programadas com quatro aulas remotas aos sábados e disponibilização dos conteúdos durante a semana. Todo o curso foi realizado dentro do ambiente PEEF [2]. Foram gravadas e disponibilizadas aproximadamente três horas de explicações sobre os assuntos do curso.

Tabela 1: Organização das atividades do curso

Semana	Conteúdo
Semana I	Algoritmos; O Ambiente PEEF; Erros e Testes Unitários; Saída de Dados; Operações Aritméticas
Semana 2	Entrada de Dados; Operadores Relacionais; Condicionais (if, else, elif); Condicionais aninhadas.
Semana 3	Repetição for, while e repetição sobre uma String
Semana 4	Funções sem e com retorno, parâmetros e Listas

A Tabela 1 apresenta os conteúdos organizados por semana. Para cada semana foram disponibilizadas 10 atividades em questões da lista de exercício. Todas as atividades continham testes unitários vinculados. Os estudantes tinham acesso às descrições da atividade e exemplos de entrada e saída na plataforma PEEF, além de um editor de código integrado [2] (Figura 2).

Para cada atividade foram criados um ou mais testes unitários que permitiram aos estudantes obter um feedback sobre a adequação da sua solução. Os testes foram divididos em públicos -– o que os estudantes podem ver no momento do teste – e privados -– os que são executados sem que o estudante tenha conhecimento. Foram criados 109 testes para as 40 atividades. Um exemplo de uso dos testes unitários por um estudante pode ser visto na Figura 3.

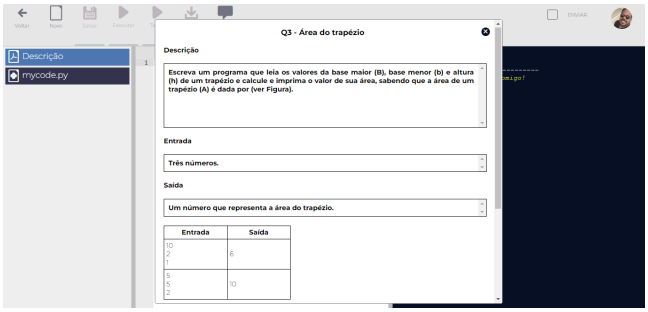

Figura 2: Descrição da Atividade na plataforma PEEF

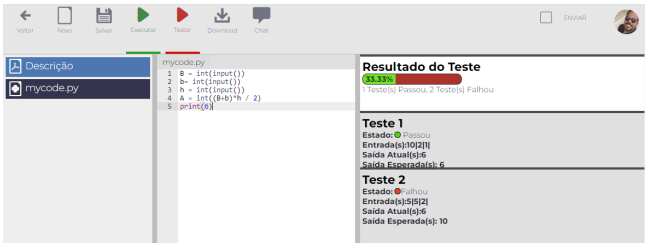

Figura 3: Execução de testes na plataforma PEEF

Para exibir as mensagens melhoradas, foram idealizadas 169 mensagens contendo tipo e subtipo do erro. A mensagem consiste em uma versão traduzida do erro e inserção de mais detalhes sobre o erro obtido, como demonstrado na Figura 4. Os tipos dos erros mais comuns foram obtidos dos trabalhos de Pritchard [14] e de Jesus et al. [8]. Estes erros foram utilizados para gerar a lista de mensagens pré-cadastradas.

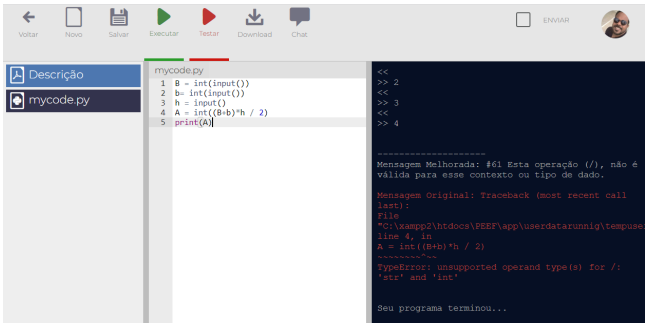

Figura 4: Apresentação da mensagem melhorada na plataforma PEEF

Além disso, foi inserido um *chatbot* utilizando a técnica de *Wizard of Oz* [17], tendo em vista que não tínhamos uma base de dados com perguntas utilizadas por estudantes. No experimento *Wizard Of Oz*, os participantes interagem com um sistema computacional que acreditam ser completamente autônomo, mas é, em partes, controlado por um agente humano [17]. Ao utilizar o *chatbot* (Figura 5), a ferramenta PEEF promove uma conversa inicial (modo automático) e envia uma notificação ao pesquisador que, em seguida, deve assumir o controle da conversa (modo manual). O *chatbot* possui algumas soluções padrão com base no tipo do erro que são apresentadas no modo automático, tal como saudação, o tipo de erro obtido e questionamento se o estudante compreendeu a mensagem de erro ou não.

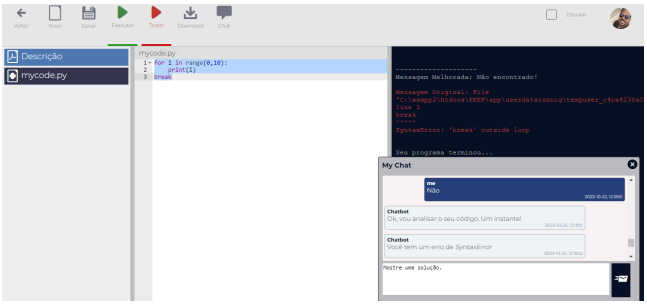

Figura 5: Conversa entre o estudante e o chatbot na plataforma PEEF

5 RESULTADOS E DISCUSSÕES

Esta seção apresenta os resultados deste estudo exploratório. Apresentamos os resultados sobre o curso seguidos pelos resultados relacionados aos tipos de feedback utilizados.

5.1 O Curso

No questionário pós-intervenção, buscamos identificar, dentre outros fatores, a mudança de percepção sobre o **nível de programação** e conhecimento em Python. Sobre o nível de programação, nove estudantes sinalizaram que conseguem compreender problemas (75%), tendo um aumento de 50% dos estudantes em comparação ao questionário pré-intervenção. 16,7% (2) dos estudantes sinalizaram que não sabiam muito sobre algoritmos, contra 33,3% (6) do questionário anterior. Um (1) estudante continuou sinalizando que ainda está aprendendo algoritmos neste semestre. A mediana das respostas pré-intervenção foi três, a mediana das resposta pós-intervenção foi quatro (Figura 6).

Sobre o **conhecimento em Python**, 50% dos estudantes sinalizaram não ter nenhum conhecimento, alternativa não sinalizada no segundo questionário. Quatro estudantes (33,3%) sinalizaram que sabia bastante (entrada, saída, condicionais, loops, listas e funções), nenhum estudante sinalizou esta opção no questionário inicial. Sete estudantes (58,3%) sinalizam que já resolviam alguns problemas (entrada, saída, condicionais e loops), tendo um aumento de 250%. A mediana das respostas pré-intervenção foi 1.5 e pós-intervenção foi quatro (Figura 7).

Para avaliar a diferença entre as medianas entre os questionários pré- e pós-intervenção quanto ao nível de programação e conhecimento em Python, foram executados testes Wilcoxon pareado. O teste Wilcoxon foi escolhido, pois os dados não seguem uma distribuição normal. Nossa Hipótese Nula (H_0) é: a mediana das respostas do questionário pré-intervenção é igual à mediana das respostas do questionário pós-intervenção. A Hipótese Alternativa (H_1) é: a mediana das respostas do questionário pré-intervenção é menor que a mediana das respostas do questionário pós-intervenção. O teste demonstrou um *p-value* = 0,0098 para o nível de programação

e *p-value* = 0,0016 para o nível de conhecimento em Python, ambos *p-value* < 0,05, confirmando as nossas Hipóteses Alternativas.

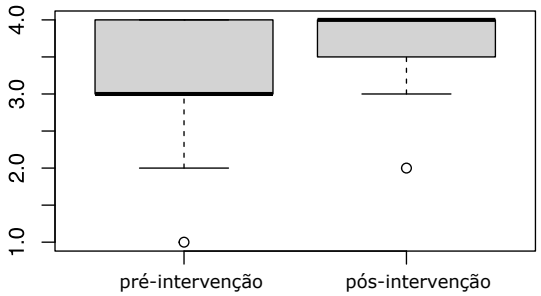

Figura 6: Box Plot dos dados sobre Níveis de Programação

Desse modo, percebe-se que o curso possibilitou aos estudantes maior confiança sobre programação e entendimento sobre a linguagem. Quanto à avaliação realizada após o curso, contendo 10 questões, os estudantes acertaram em média 70% da avaliação. Nove estudantes obtiveram notas entre sete e nove e apenas dois estudantes obtiveram nota inferior a cinco.

5.2 Feedbacks Emitidos

Durante o curso, foram emitidos três tipos de feedbacks por meio da ferramenta PEEF: testes unitários, mensagens melhoradas e mensagens via *chatbot*. Nós avaliamos os feedbacks por meio de questionários e dados obtidos da ferramenta. A seguir, apresentaremos os resultados obtidos.

5.2.1 Testes. Durante o estudo de caso foram realizados 1.150 testes. Deste quantitativo, 721 falharam (62,6%) e 429 resultaram em sucesso (37,4%). Nós criamos **pares de testes** para avaliar o que ocorre após uma falha no teste inicial (Figura 8). Foram criados 697 pares. 180 pares (25,8%) demonstraram que os estudantes obtiveram sucesso na primeira tentativa do teste. 101 pares (14,4%) demonstraram que os estudantes obtiveram sucesso após ao menos um teste falho. Os estudantes obtiveram falha em testes seguidos em 416 pares de testes (59,6%). Assim, nota-se que o quantitativo de testes que falham é superior aos que resultam em sucesso.

Esta constatação, no entanto, não significa uma má adequação dos testes, já que a falha demonstra aos estudantes o caminho a ser

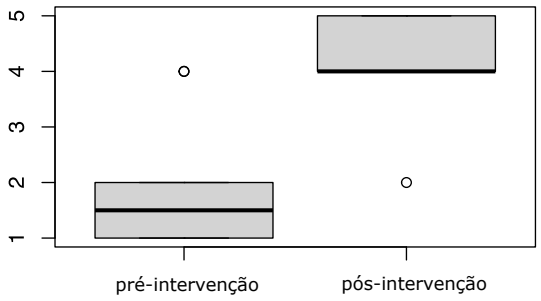

Figura 7: Box Plot dos dados obre Conhecimento em Python

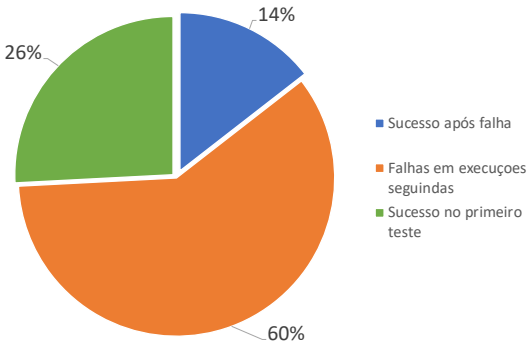

Figura 8: Situações mapeadas nos pares de teste

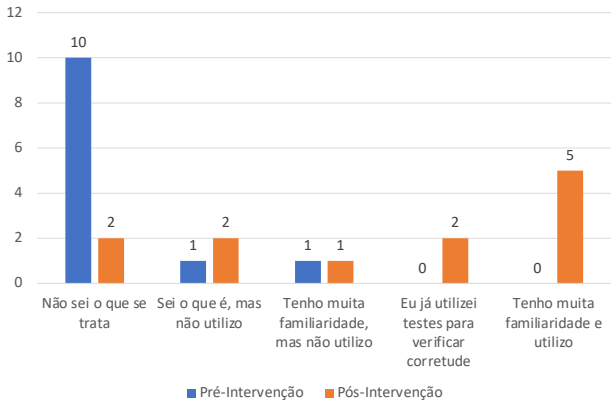

Figura 9: Percepções dos estudantes sobre testes unitários

seguido. Além disso, ao alcançar o objetivo da questão, os estudantes não continuam executando os testes. Portanto, é comum que os testes que falham sejam superiores aos que obtém sucesso. Dos 391 projetos criados, 327 tiveram testes realizados (83,6%). Destes, 296 projetos tiveram ao menos uma execução de testes que resultou em sucesso (90,5%), em sua maioria o último.

Em relação à **percepção dos estudantes quanto a testes unitários**, após o curso, dois estudantes (16,7%) sinalizaram não saber do que se trata, tendo uma redução de 80% quanto ao questionário pré-intervenção. É importante ratificar que, durante o curso, os estudantes tiveram acesso a aulas explicativas sobre o uso de testes unitários. 41,7% dos estudantes sinalizaram ter familiaridade com os testes e que os utilizam, nenhum estudante sinalizou isso anteriormente. Um (1) estudante (8,3%) continuou indicando que tem familiaridade, mas não utiliza em estudos. Dois estudantes (16,7%) sinalizaram saber o que é, mas não utilizavam. O mesmo quantitativo de estudantes (16,7%) sinalizou já ter utilizado testes para verificar a corretude dos códigos fora da plataforma. A Figura 9 sintetiza esses dados.

A análise descritiva demonstrou que a mediana das respostas sobre a percepção de testes unitários pós-intervenção é quatro e a pré-intervenção é um (Figura 10). Para avaliar se as medianas entre as respostas dos questionários são diferentes, foi executado o teste

não-paramétrico Wilcoxon pareado. Nossa Hipótese Nula (H_0) é: a mediana das respostas do questionário pré-intervenção é igual à mediana das respostas do questionário pós-intervenção. A Hipótese Alternativa (H_1) é: a mediana das respostas do questionário pré-intervenção é menor que a mediana das respostas do questionário pós-intervenção. O teste demonstrou um *p-value* = 0,004 confirmando a nossa Hipótese Alternativa. Logo, há razões para afirmar que ocorreu uma mudança significativa quanto a percepção sobre o conhecimento de testes unitários.

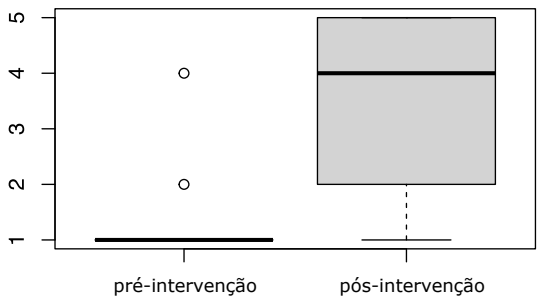

Figura 10: Box Plot dos dados sobre familiaridade com o Testes Unitários

Ao serem perguntados sobre **o que fazem após testar** o código e obter falha durante o curso, 83,3% dos estudantes afirmaram que analisam a saída esperada e a saída atual para modificar o código. Esta conduta pode melhorar a performance dos estudantes, resultado equiparado ao apresentado por Manzoor et al. [13]. Apenas 16,7% afirmaram utilizar as entradas de testes na execução para entender melhor. Esta última situação ajuda ainda a entender possíveis comportamentos dos estudantes que retornam ao modo de execução após a realização do teste. Percebeu-se que 193 pares de execuções demonstram um comportamento de retorno à execução, após o teste. Outro fator que pode explicar o retorno do estudante para execuções é a possibilidade de acesso ao feedback da mensagem de erro melhorada. Dos 193 pares, 90 testes retornaram em erro sintático.

Ao serem perguntados sobre **o que fazem ao não entenderem as saídas esperadas**, a maioria dos estudantes preferem modificar o código e executar novamente (75%), 16,7% dos estudantes preferem pesquisar na internet e apenas 8,3% abriram o chat para enviar uma dúvida. Por fim, 50% dos estudantes sinalizaram, dentre todos os tipos de feedback ofertados, os testes unitários como o tipo de feedback preferido. Adicionalmente, nós pedimos que os estudantes dessem notas a cada tipo de feedback, avaliando o quanto ele ajuda a resolver problemas de modo isolado ou em conjunto com outro feedback. Para 33% dos estudantes, o teste unitário junto com outra forma de feedback ajuda a resolver a tarefa completamente. 83,3% avaliam que os testes também ajudam a resolver os problemas de modo isolado. Desse modo, percebe-se que os participantes aderiram ao uso de testes unitários e possuem uma percepção positiva quanto a ajuda deste tipo de feedback.

5.2.2 Mensagens de Erro Melhoradas. Durante o estudo de caso, foram realizadas 2.185 execuções, sendo 49.7% executaram sem erro e 1.207 execuções retornaram com erro. Não considerando os erros em tempo de execução, o estudo obteve 995 execuções com erro sintáticos. Neste estudo, desconsiderar este tipo de erro é adequado dado o fato de que foram projetadas mensagens apenas para erros sintáticos. Das 995 execuções, 812 retornaram mensagens melhoradas pré-cadastradas, obtendo uma cobertura de 81,6%. Este dado supera a cobertura publicada por Becker et al. [5]. Nós não consideramos as execuções dos testes como execuções, pois as classificamos e analisamos separadamente.

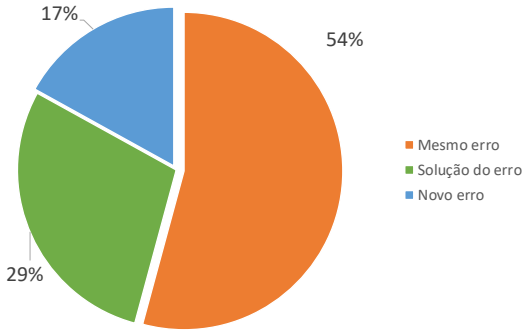

Figura 11: Situações mapeadas nos pares de execuções

Foram criados **pares de execuções** para avaliar o que ocorre após a emissão de mensagens melhoradas (Figura 11). 725 execuções foram identificadas por meio dos pares. Destas, 393 (54,2%) continuaram com o mesmo erro na execução seguinte. 209 (28,8%) execuções subsequentes não retornaram erro, ou seja, o estudante conseguiu resolver o problema. Este dado é similar ao apresentado por Kohn and Manaris [11] sobre os erros superficiais. Continuando, 123 (16,9%) execuções foram seguidas de um novo erro. A solução do erro indica que a mensagem melhorada pode ter ajudado o estudante a solucionar o problema. Tendo em vista que o tradutor da linguagem Python apresenta apenas o primeiro erro, diferentemente de tradutores de outras linguagens que apresentam uma sequência de erros, a aparição de um novo erro também podem sinalizar a resolução do primeiro erro.

A Tabela 2 apresenta a quantidade de cada **tipo de erro cometido pelos estudantes** durante o estudo. Os erros com menos de quatro ocorrências foram suprimidos, mas podem ser consultados no site de apoio[1]. Percebe-se que *SyntaxError* é o erro com maior ocorrência; isso pode ser justificado pelo fato de na traduação da linguagem Python muitos erros são classificados como *SyntaxError* [11]. *NameError* é o segundo erro como maior número de ocorrências; este erro por vezes é proveniente de uma digitação incorreta de uma variável ou função. *NameError* é seguido por *TypeError* que representa o erro de tipo, como por exemplo, concatenar um String com um inteiro. Erro de Indentação se apresenta como quarto com maior número de ocorrências. Estes resultados corroboram com os achados de [14] e [8].

Sobre a **experiência com mensagens de erro**, oito estudantes (66,7%) sinalizaram que entendem a mensagem e resolvem o

[1]Site de apoio: https://sites.google.com/view/educomp2024multiplosfeedbacks

Tabela 2: Erros mais frequentes durante o curso

Tipo	Subtipo de Erro	Quantidade
SyntaxError	invalid syntax	461
NameError	name x is not defined	133
TypeError	unsupported operand type x for +	76
IndentationError	unexpected indent	68
AttributeError	object has no attribute	37
ValueError	invalid literal for int x with base	25
IndexError	string index out of range	6

problema facilmente, representando um aumento de 100%, em relação ao questionário inicial. 16,7% dos estudantes continuaram sinalizando que não conseguem entender a mensagem. O mesmo quantitativo de estudantes sinalizou ler a mensagem, mas não conseguem solucionar os problemas. Nenhum estudante sinalizou esta alternativa no primeiro questionário. As opções sobre não é uma prática ler a mensagem e normalmente não leio a mensagem não foram sinalizadas no segundo questionário. A Figura 12 sintetiza esses dados.

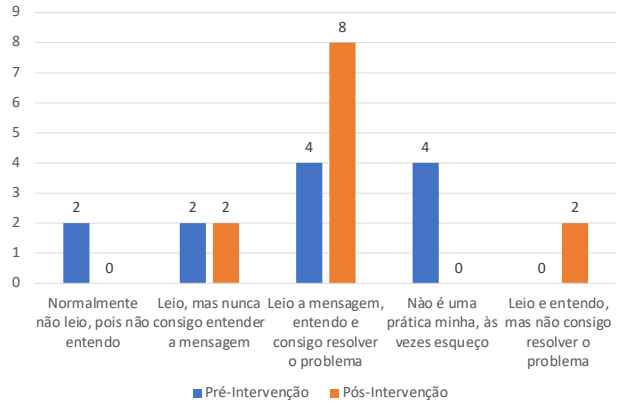

Figura 12: Percepções dos estudantes sobre mensagens de erro

A análise descritiva demonstrou que a mediana das respostas relacionadas à percepção sobre mensagens de erro pós-intervenção é cinco e a pré-intervenção é dois (Figura 13). Para avaliar se as medianas entre os questionários são diferentes, foi executado o teste não-paramétrico Wilcoxon pareado. As mesmas hipóteses do uso de testes unitários foram utilizadas. H_0: a mediana das respostas do questionário pré-intervenção é igual à mediana das respostas do questionário pós-intervenção. H_1: a mediana das respostas do questionário pré-intervenção é menor que a mediana das respostas do questionário pós-intervenção. O teste demonstrou um *p-value* = 0,0039, confirmando a nossa Hipótese Alternativa.

As mensagens melhoradas foram escritas em português, idioma nativo dos estudantes. Os estudantes foram questionados sobre o uso de **mensagens de erro em português**. Para 83,3% dos estudantes, ler mensagens em português ajudou a resolver os problemas e

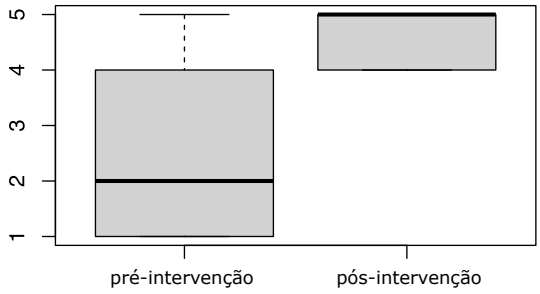

Figura 13: Box Plot dos dados sobre experiência com Mensagens de Erro

16,7% dos estudantes são neutros quanto a isso. Nenhum estudante afirmou que as mensagens em português não colaboram. Nós buscamos saber o que os estudantes **fazem após executar o código e obter um erro**: 91,7% dos estudantes afirmam que lêem a mensagem padrão e a melhorada e 8,3% dos estudantes afirmam ler a mensagem original. Este percentual corresponde ao estudante que afirmou ser proficiente em inglês, no entanto este estudante não indicou ser proficiente em inglês no questionário pré-intervenção.

Buscou-se saber o que os estudantes **faziam após ler a mensagem de erro e não entender**: 58,3% modificam o código e executam novamente, 33,3% pesquisam na internet e 8,3% utilizam o chat. Para 50% dos estudantes, as mensagens melhoradas é o tipo de feedback preferido. Desse modo, percebe-se que a emissão de mensagens melhoradas ajuda os estudantes na resolução de problemas e é um tipo de feedback adotado pelos participantes durante o curso.

Tabela 3: Média do RED e REC por estudante

Estudante	Média **REC**	Média **RED**
1	8,58	6,39
2	0,84	0,52
3	2,62	1,07
4	7,11	4,12
5	2,01	1,47
6	3,09	2,05
7	1,13	0,50
8	4,17	2,26
9	2,70	1,75
10	2,69	1,97
11	1,34	0,52
12	0,66	0,46

Adicionalmente, realizamos o **cálculo das métricas** *Repeated Error Density* (RED), proposta em [4], e *Recurrent Error Density* (REC), proposta por estes autores em um trabalho anterior [1]. A Tabela 3 apresenta a média dos escores por estudante. As médias foram calculadas pela divisão da soma dos escores pela quantidade de projetos criados. A mediana da soma dos escores são 50,30 para RED e 77,53 para REC. Em seu trabalho, Becker [4] sinalizou um valor RED de 34,0 para o grupo de intervenção e 49,4 para o grupo controle. Este estudo foi realizado utilizando a linguagem Java e o

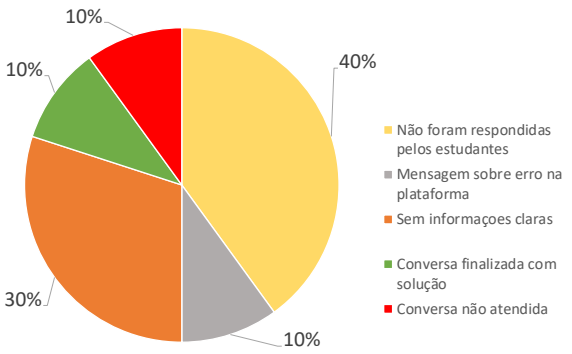

Figura 14: Situações ocorridas durante o uso do chatbot

ambiente Decaf, além disso a sua estrutura e número de participantes são bastante divergentes [5], não permitindo uma comparação entre as abordagens. Dentro do *design* de pesquisa escolhido, não é possível realizar alguma análise comparativa entre os estudantes. No entanto, a apresentação desses dados são importantes no sentido de fornecer base para futuros trabalhos que utilizarão estas métricas.

5.2.3 Chat. Durante o curso, foram enviadas 41 mensagens para o chat, referente a dez projetos de seis estudantes distintos. Percebe-se que os estudantes abandonam algumas conversas sem fornecer informações para o chat, quando solicitado. Apenas em 10 projetos os estudantes utilizando o chat (2,5% do total de projetos criados), 4 conversas não foram respondidas pelos estudantes após a apresentação da solução. Uma (1) conversa em chat visou relatar um problema de carregamento da imagem, na plataforma. Três estudantes, ao conversar com o chat, não forneceram informações direcionadas para o que pudessem resolver o problema. Uma (1) conversa foi finalizada, com a apresentação da solução para o estudante. Por fim, não foi possível atender a uma conversa, tendo em vista que o modo automático do chat chegou ao seu limite e o pesquisador não entrou na plataforma. A mensagem foi respondida 20 minutos após o envio e não obteve resposta do estudante. A Figura 14 sintetiza os dados apresentados.

No questionário pós-intervenção, apenas um (1) estudante (8,3%) sinalizou que após ler a mensagem de erro e não entender, normalmente abre o chat. O mesmo quantitativo sinalizou que, após obter falha no teste, normalmente abre o chat. Os estudantes foram questionados sobre **o que fazem após enviar uma mensagem ao chat e receber a solução**; três estudantes afirmam que analisam a informação, um (1) estudante afirma que modifica o código sem essa informação e oito estudantes afirmaram que não utilizaram o chat. No entanto, foram observadas na base de dados conversas de seis estudantes. Um fator que pode explicar essas respostas é que um estudante enviou apenas uma mensagem e outro estudante enviou apenas três mensagens, nenhum dos dois forneceram mais informações para o avanço da conversa.

Ao serem questionados **sobre o tipo de feedback preferido**, nenhum estudante sinalizou o chat. Para oito estudantes, de modo isolado o chat não é capaz de ajudar na solução do problema. Para sete estudantes, mesmo com outro tipo de feedback, o chat não ajuda na tarefa. Para dois estudantes, o chat ajuda a resolver tarefas. Apenas para um (1) estudante o chat, junto com outro tipo de feedback, ajuda a resolver a tarefa completamente. Nesse sentido, o chat não foi amplamente utilizado pelos estudantes e, diferente dos outros tipos de feedback, teve uma avaliação negativa pós-intervenção. Esta falta de engajamento é observada também no trabalho de Verleger and Pembridge [16].

6 CONCLUSÃO

Este trabalho apresentou um estudo exploratório sobre múltiplos tipos de feedback com o objetivo de compreender o uso e eficácia dos feedbacks. Para isso foi realizado um estudo de caso em um curso introdutório de Python durante quatro semanas. Os estudantes tiveram acesso a mensagens melhoradas, testes unitários e chats. Para coleta de dados, foram disponibilizados questionários pré- e pós-intervenção, além da obtenção de dados na plataforma PEEF. Os dados demonstram que o curso é similar a outros cursos já realizados tendo em vista que os erros obtidos são similares a listas de erros em trabalhos anteriores. Além disso, o curso foi adequado aos estudantes que, em geral, responderam corretamente 70% da avaliação final. Foi constatada uma mudança positiva e significativa quanto a percepção dos níveis de programação e conhecimento na linguagem Python.

Quanto a adequação e eficácia dos tipos de feedback, percebe-se que o chat é um tipo de feedback pouco utilizado e quando o estudante inicia uma conversa, na maioria das vezes, abandona ou não oferece informações precisas sobre o que deseja. Diante da existência de outros tipos de feedback, o chat não se configura como o predileto entre os estudantes.

O uso de testes unitários demonstrou ser positivo, colaborando com o progresso dos estudantes e tendo uma boa avaliação dos estudantes: 90.5% dos projetos obtiveram sucesso nos testes realizados.

Por fim, as mensagens melhoradas se configuraram também como adequadas, possibilitando uma cobertura acima de 80%. Embora o nível de assertividade tenha sido baixo para este tipo de feedback, ele foi avaliado positivamente pelos estudantes: 28,8% dos pares de execução com emissão de mensagens de erro melhoras foram solucionadas. Percebeu-se, ainda, mudanças significativas sobre a percepção dos estudantes quanto ao uso de testes e mensagens de erro.

Desse modo, pretende-se, em trabalhos futuros, aprimorar a forma de feedback de mensagens de erro melhoradas, possibilitando aumento na cobertura e, principalmente, aumento na assertividade da explicação do erro. Pretende-se, ainda, estudar formas de aumentar o uso do chat e reduzir o abandono da conversa pelos estudantes. Por fim, visa-se realizar um experimento controlado para avaliar a eficácia dos feedbacks entre estudantes na segunda turma do curso de programação introdutória em Python.

REFERÊNCIAS

[1] Luis Gustavo J Araujo, Roberto A Bittencourt, and Christina FG Chavez. 2022. Uma avaliação comparativa entre métricas de erro em um curso introdutório de programação com Python. In *Anais do II Simpósio Brasileiro de Educação em Computação*. SBC, 40–49.

[2] Luis Gustavo J. Araujo, Roberto A. Bittencourt, and Christina F. G. Chavez. 2021. Python Enhanced Error Feedback: Uma IDE Online de Apoio ao Processo de Ensino-Aprendizagem em Programação. In *Anais do Simpósio Brasileiro de Educação em Computação*. SBC, 326–333.

[3] Brett A Becker. 2015. *An exploration of the effects of enhanced compiler error messages for computer programming novices.* Ph.D. Dissertation. Technological University Dublin.

[4] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 296–301.

[5] Brett A Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3, 148–175.

[6] Tessa Charles and Carl Gwilliam. 2023. The Effect of Automated Error Message Feedback on Undergraduate Physics Students Learning Python: Reducing Anxiety and Building Confidence. *Journal for STEM Education Research* 6, 1, 1–32.

[7] Eryck Pedro da Silva, Ricardo Edgard Caceffo, and Rodolfo Azevedo. 2023. Passar nos casos de teste é suficiente? Identificação e análise de problemas de compreensão em códigos corretos. In *Anais do III Simpósio Brasileiro de Educação em Computação*. SBC, SBC, Porto Alegre, RS, Brasil, 119–129.

[8] Galileu Santos de Jesus, Jaine da Conceição Santos, Kleber Tarcísio Oliveira Santos, and Alberto Costa Neto. 2019. Análise dos erros mais comuns de aprendizes de programação que utilizam a linguagem Python. *Anais do Computer on the Beach* 10, 1, 406–415.

[9] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. Association for Computing Machinery, Nova York, Nova York, Estados Unidos, 273–278.

[10] Harry Barton Essel, Dimitrios Vlachopoulos, Akosua Tachie-Menson, Esi Eduafua Johnson, and Papa Kwame Baah. 2022. The impact of a virtual teaching assistant (chatbot) on students' learning in Ghanaian higher education. *International Journal of Educational Technology in Higher Education* 19, 1, 1–19.

[11] Tobias Kohn and Bill Manaris. 2020. Tell Me What's Wrong: A Python IDE with Error Messages. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1054–1060.

[12] Rodziah Latih, Marini Abu Bakar, Norleyza Jailani, Noorazean Mohd Ali, Syahanim Mohd Salleh, and Abdullah Mohd Zin. 2017. PC 2 to support instant feedback and good programming practice. In *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*. IEEE, IEEE, Langkawi, Malasia, 1–5.

[13] Hamza Manzoor, Amit Naik, Clifford A Shaffer, Chris North, and Stephen H Edwards. 2020. Auto-grading jupyter notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, Nova York, Nova York, Estados Unidos, 1139–1144.

[14] David Pritchard. 2015. Frequency distribution of error messages. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*. Association for Computing Machinery, Nova York, Nova York, Estados Unidos, 1–8.

[15] V Javier Traver. 2010. On compiler error messages: what they say and what they mean. *Advances in Human-Computer Interaction* 2010, 1–26.

[16] Matthew Verleger and James Pembridge. 2018. A pilot study integrating an AI-driven chatbot in an introductory programming course. In *2018 IEEE frontiers in education conference (FIE)*. IEEE, IEEE, San Jose, California, Estados Unidos, 1–4.

[17] Nick Webb, David Benyon, Jay Bradley, Preben Hansen, and Oli Mival. 2010. Wizard of Oz Experiments for a Companion Dialogue System: Eliciting Companionable Conversation. In *International Conference on Language Resources and Evaluation*. LREC, Valletta, Malta, 6.

[18] Robert K Yin. 2001. *Estudo de Caso-: Planejamento e métodos*. Bookman editora, Porto Alegre, RS, Brasil.