Python Enhanced Error Feedback: Uma IDE Online de Apoio ao Processo de Ensino-Aprendizagem em Programação

Luis Gustavo J. Araujo
UFBA – Universidade Federal da Bahia
Salvador, Bahia, Brasil
luisaraujo.ifba@gmail.com

Roberto A. Bittencourt
UEFS – Universidade Estadual de Feira de Santana
Feira de Santana, Bahia, Brasil
roberto@uefs.br

Christina F. G. Chavez
UFBA – Universidade Federal da Bahia
Salvador, Bahia, Brasil
flach@ufba.br

RESUMO

Aprender a programar pode ser um desafio para muitos estudantes. São exigidas deles diversas competências como pensamento lógico, interpretação de texto e habilidades matemáticas que tornam complexa a programação. Em geral, professores não utilizam ferramentas que fornecem bom *feedback* para quem está aprendendo, pois são projetadas para programadores experientes, fator que potencializa o problema apresentado. Ao mesmo tempo, professores dificilmente conseguem fornecer *feedback* apropriado em tempo hábil e de modo individual, dado o seu volume de tarefas. Assim, ferramentas que proveem *feedback* apropriado aos estudantes podem auxiliar neste processo e diminuir a complexidade do processo de aprendizagem de programação. Este artigo apresenta a Python Enhanced Error Feedback (PEEF), uma IDE online para auxiliar no processo de ensino-aprendizagem que fornece diversas formas de *feedback* tais como mensagens melhoradas, testes unitários e chat. São apresentados dois exemplos de uso de PEEF para cada um dos potenciais usuários. A ferramenta se apresenta como uma possibilidade de uso no ensino de programação, fornecendo *feedback* para os estudantes e professores. Por fim, possibilita um estudo mais aprofundado sobre tipos de *feedback* fornecidos aos estudantes novatos.

CCS CONCEPTS

• **Social and professional topics → CS1**.

PALAVRAS-CHAVE

Feedback, ensino de programação, mensagens melhoradas, testes unitários.

1 INTRODUÇÃO

Aprender programação pode ser um desafio complexo para estudantes [15], o que é ratificado pelas altas taxas de reprovação em cursos de Computação ao longo de décadas [3, 4, 20]. Por conseguinte, estudantes novatos necessitam de acompanhamento individualizado pois estão mais suscetíveis a cometer equívocos sobre conceitos, linguagens ou paradigmas de programação [15]. Esses equívocos podem ocorrer por questões relacionadas à linguagem ou às mensagens geradas pelos compiladores, que são em sua maioria destinadas a programadores experientes. Além disso, estudantes podem cometer equívocos pela limitação em identificar se a sua solução está correta. Neste sentido, soluções como a introdução de mensagens melhoradas, chat e testes unitários são adotadas em abordagens pedagógicas [1, 14, 17].

Percebe-se, no entanto, que muitos trabalhos utilizam ferramentas de uso geral como *BlueJ* [8], *Decaf* [1], *Eclipse* [13] e outros. Este fator dificulta o armazenamento e tratamento da informação, dificulta a implementação por professores em cenários diferentes (*e.g.*, idioma e linguagem de programação) e o acompanhamento do progresso dos estudantes. Além disso, Luxton-Reilly et al. afirmam que muitas ferramentas de *feedback* focam em fornecer auxilio de modo geral e não individualizado [14]. Ainda segundo os autores, essas ferramentas utilizam muitas formas de *feedback*, mas os estudos ainda não se aprofundaram em avaliar a eficácia de mensagens geradas pelos sistemas e mensagens escritas pelos professores para alunos específicos.

Neste contexto, a criação de uma plataforma que apoie abordagens pedagógicas com o uso de *feedback* melhorado, seja por meio de mensagens de erro, *chat* ou testes unitários se torna relevante. Além disso, torna-se ainda mais necessário a possibilidade de avaliar a eficácia de vários tipos de *feedback*. Este artigo tem o objetivo de apresentar o ambiente *Python Enhanced Error Feedback* (PEEF), que visa cobrir os problemas supracitados além de permitir um acompanhamento do desempenho dos estudantes através de coleta de métricas e *logs*.

O presente artigo está dividido em seis seções. A Seção 2 apresenta o referencial teórico sobre *Feedback*, Mensagens de Erro Melhoradas, Uso de Testes Unitários, e Métricas utilizadas para acompanhar o progresso dos estudantes. A ferramenta é apresentada na Seção 3, tanto sua arquitetura como suas funcionalidades. Na Seção 4, dois exemplos de uso são apresentados, um do ponto de vista do estudante e outro do ponto de vista do professor. Na Seção 5, são apresentados os trabalhos relacionados e, por fim, na Seção 6, as considerações finais e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Estudantes novatos cometem diversos erros quando estão aprendendo programação. Neste nível, estes estudantes estão mais suscetíveis a incorporarem equívocos conceituais [15]. No entanto, o erro é um aspecto importante no processo de aprendizagem, pois reflete o atual momento do estudante e ajuda-o a desenvolver suas habilidades. Tendo em vista que na programação estes equívocos

EduComp'21, Abril 27–30, 2021, Jataí, Goiás, Brasil (On-line)

se manifestam como erros de sintaxe ou *bugs* [11], espera-se inicialmente que os estudantes tenham auxílio dos compiladores e ambientes integrados de desenvolvimento (IDEs) para o entendimento dos seus erros.

O auxílio durante o processo de aprendizagem é de suma importância. Durante o processo de construção do conhecimento, os estudantes transitam entre o Desenvolvimento Real (DR) e o Desenvolvimento Proximal (DP). Para Vygotsky, a Zona de Desenvolvimento Proximal (ZDP) é uma faixa intermediária entre o que é possível fazer sozinho (DR) e o que só se é capaz de fazer através de mediação (DP) [19]. Assim, sem mediação, esta transição se torna mais difícil.

O auxílio dado pelos compiladores é apresentado na forma de mensagens de erro (Compiler Error Messages – CEM). No entanto, estas mensagens são frequentemente enigmáticas para novatos, colocando barreiras para o sucesso de programadores novatos [1], e tornando insuficiente o auxílio dos compiladores no processo de aprendizagem. Neste sentido, faz-se necessária a adoção de elementos que possibilitem um *feedback* mais apropriado aos estudantes novatos e, assim, auxilie-os na ZDP.

Como afirmam Ott et al., o *feedback* é visto principalmente sob o contexto de avaliação, com foco em avaliações formativas [17]. No entanto, tem-se discutido também sobre o papel do *feedback* na autorregulação e autoavaliação. O *feedback* pode ser entendido como a informação que auxilia os estudantes a diminuir o espaço entre o desempenho atual e o desempenho desejado [17]. Na perspectiva sociointeracionista, o *feedback* pode auxiliar na transição entre o Desenvolvimento Real e o Proximal. Segundo Nicol and Macfarlane-Dick [16], são características de um bom *feedback*:

i) ajuda a esclarecer o que é bom desempenho (objetivos, critérios, padrões esperados);
ii) facilita o desenvolvimento da autoavaliação;
iii) fornece informações de alta qualidade aos alunos sobre sua aprendizagem;
iv) encoraja o professor e o diálogo com os colegas sobre a aprendizagem;
v) encoraja crenças motivacionais positivas e melhoria da autoestima;
vi) oferece oportunidades para fechar a lacuna entre o desempenho atual e o desejado;
vii) fornece informações aos professores que podem ser usadas para ajudar a moldar o ensino.

No intuito de fornecer bom *feedback* em relação aos erros de programação, as CEMs podem ser adaptadas para atender às necessidades de programadores novatos. Este tipo de mensagem é denominada de mensagem de erro do compilador melhorada (Enhanced Compiler Error Message - ECEM). Uma ECEM pode conter detalhes como nome de métodos, variáveis, notas explicativas, exemplos de uso, possíveis causas do erro ou imagens que o ilustrem. Sempre que possível, a ECEM vem acompanhada da CEM (original), pois dá a oportunidade ao estudante de entender o significado original da mensagem de erro [1]. Neste sentido, a ECEM atende aos critérios i, ii, iii e vi quanto a um bom *feedback*.

No entanto, ao passo que mensagens melhoradas visam esclarecer aspectos sobre o erro gerado, auxiliando na resolução do problema, os *bugs* só podem ser identificados através de suas saídas [11]. Por isso, o uso de testes unitários é uma alternativa que fornece aos estudantes informações sobre a corretude de seus códigos. Esta abordagem é denominada Aprendizagem Baseada em Testes (Test-Driven Learning – TDL). TDL consiste em utilizar testes unitários automatizados para projetar e verificar exemplos, atividades e avaliações [9]. TDL atende aos critérios i, ii, iii e vi quanto a um bom *feedback*.

Na perspectiva do professor, o uso de testes unitários atende também ao critério vii. Além disso, *feedbacks* sobre o desempenho e progresso dos estudantes podem ser considerados para facilitar o seu trabalho e otimizar as orientações aos estudantes. Assim, estudantes com maiores dificuldades podem receber *feedbacks* específicos, em tempo hábil.

Informações sobre a linha em que o erro ocorre e o tipo do erro também compõem o grupo de informações sobre o progresso e desempenho dos estudantes. Por fim, algumas métricas podem ser utilizadas para apoiar a análise de desempenho dos estudantes. Pode-se destacar duas métricas, tendo em vista a sua proximidade com o *feedback* de mensagens melhoradas: *Error Quotient* (*EQ*), proposta por Jadud [8], e *Repeated Error Density* (*RED*), proposta por Becker [1].

Jadud propõe uma métrica baseada na verificação de pares de execuções consecutivas [8]. O autor avalia se as duas execuções possuem erros e se os erros são do mesmo tipo. Para cada uma das verificações é adicionado um peso: 8 para a primeira e 3 para a segunda. Após os cálculos, o valor é normalizado, sendo dividido por 11. Por fim, é gerada uma média de todos os escores calculados. Esta média é o valor *EQ* dentro de uma sessão de tempo do estudante.

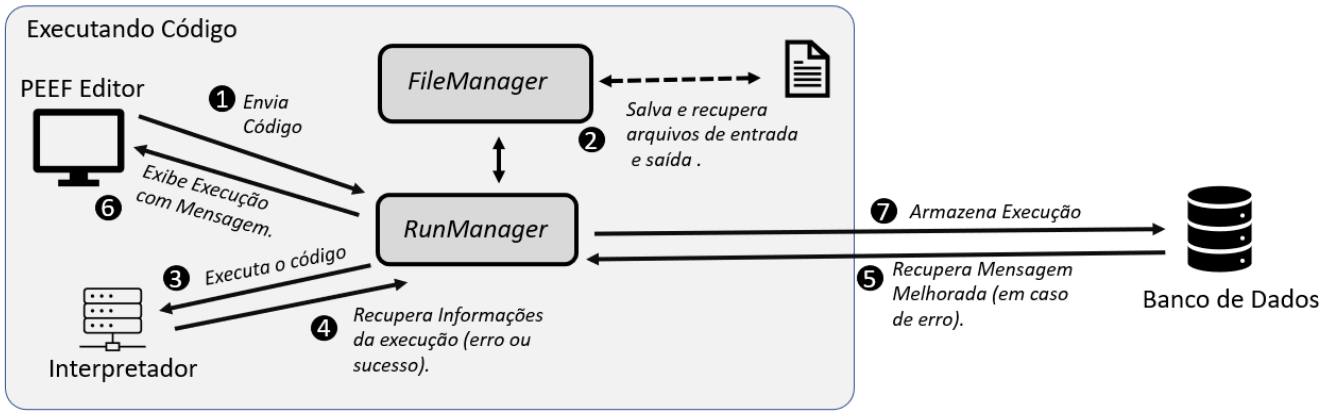
Becker propõe outra abordagem para o cálculo de desempenho dos estudantes a partir dos erros cometidos [1]. Segundo este autor, a métrica *EQ* não dá conta de distinguir certos comportamentos de execução dos estudantes em relação a erros repetidos. Assim, ele propõe uma métrica que calcula a densidade de erros repetidos. Dadas as sequências de erros repetidos em uma sessão, é utilizada a Equação 1 para o cálculo do *RED*, sendo r_i o número de repetições do erro de um mesmo tipo.

$$RED = \sum \frac{r_i^2}{r_i + 1} \tag{1}$$

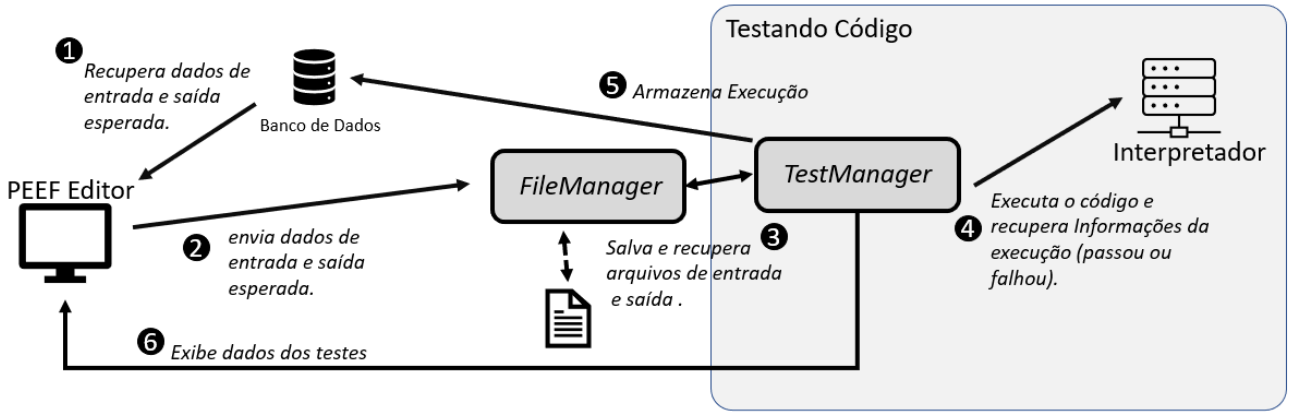
3 FERRAMENTA

Python Enhanced Error Feedback (PEEF) é um ambiente virtual de aprendizagem desenvolvido para o ensino e na aprendizagem de programação em Python. PEEF está sendo desenvolvido sob a licença GNU *General Public License* (GLP). Por meio do PEEF, professores podem disponibilizar cursos com vídeos, materiais de leitura e atividades. Os estudantes matriculados podem acessar os materiais disponibilizados pelos professores e realizar as atividades de programação por meio do editor online. A plataforma possui um conjunto de funcionalidades referentes a *feedback* para os estudantes e métricas para acompanhamento pelos professores. Além disto, é possível configurar o idioma a ser utilizado na plataforma, como, por exemplo, o português. Atualmente, PEEF está hospedado em um site dedicado ao ambiente[1].

[1] www.peefonline.com



(a) Fluxo da execução de Código



(b) Fluxo da execução de Teste Unitário

Figura 1: Arquitetura do PEEF

3.1 Arquitetura

O ambiente PEEF foi desenvolvido com tecnologias web. No *frontend*, foram utilizados HTML, CSS e *JavaScript*, juntamente com a biblioteca *JQuery*. No *backend*, foi utilizada a linguagem PHP. Os dados são armazenados em um banco de dados relacional (*MySQL*). Para o versionamento da ferramenta, foi utilizada a plataforma *GitHub*.

A ferramenta possui um editor online com *highlight* da biblioteca Ace.js. PEEF utiliza um conjunto de arquivos para garantir a execução dos códigos dos estudantes, permitindo interação de entrada e saída de dados. Cada saída gerada pelo código é armazenada em um arquivo (*output*). Os dados de saída são verificados inúmeras vezes durante uma execução. Ao ser encontrado um dado novo, ele é consumido e exibido no console do editor. Os estudantes podem digitar entradas no console. Estas entradas são armazenadas em arquivos (inputs_n) e consumidas para a execução do código. A Figura 1a apresenta um esquema visual deste fluxo.

Para a execução dos testes, é utilizado um fluxo similar ao de execução. Porém, os dados de entrada são provenientes do banco de dados. Antes da execução, são criados arquivos com os respectivos dados. Ao término da execução, é realizada uma comparação entre a saída da execução e a saída esperada, armazenada no banco de dados. Os dados são formatados em um arquivo JSON com informações dos testes executados: número de testes, porcentagem de testes que passaram, porcentagem de testes que falharam. Para cada teste, são armazenadas a entrada, a saída atual, a saída esperada e informação se o teste passou ou não. A Figura 1b apresenta um esquema visual deste fluxo.

O ambiente PEEF possui dois módulos principais: O módulo Professor e o módulo Estudante. Os módulos são focados em usuários potenciais. Usuários potenciais do módulo Professor são professores do ensino técnico, superior ou de cursos livres. Usuários potenciais do módulo Estudante são os respectivos estudantes dos cursos dos professores mencionados anteriormente.

3.2 Funcionalidades do Módulo Professor

Um professor pode criar cursos e adicionar aulas e atividades. Cada aula contém um título, uma descrição/exemplo e vídeo. Cada atividade possui um título, uma descrição e imagens de suporte que têm relação com o problema a ser resolvido.

3.2.1 Visualização de Histórico de Edição. O professor pode visualizar as modificações realizadas pelo estudante em uma atividade. Para cada execução, é calculado o valor do *diff* com o algoritmo

Tabela 1: Exemplos de Mensagens Melhoradas do PEEF

Tipo do Erro	Subtipo	Mensagem Melhorada
SyntaxError	invalid syntax	Ocorreu um erro de Sintaxe. Isso significa que no seu código há um padrão não reconhecido Python. Verifique se falta algum parêntese em comandos como if, while, for, print, input e outros. Verifique ainda o uso de ':' em comandos como if e for.
SyntaxError	Missing parentheses in call to	O erro pode ter ocorrido em uma chamada de função. Utilize os parênteses. Exemplo: print('Hello Word') .
TypeError	can\'t multiply sequence by non-int of type \'str\'	Você está tentando multiplicar um inteiro por um valor do tipo string. Verifique o tipo de dados dos valores.

Tabela 2: Logs Coletados

Nome	Descrição
Log_in	Entrou na Plataforma.
Log_out	Saiu da Plataforma.
Log_Exe	Executou um código.
Log_Error	O código executado possui um erro.
Log_Test	Executou um teste.

STRING-EDIT. Este valor representa o número mínimo de modificações no código anterior (inserção, remoção ou atualização) para que se obtenha o código atual. Este cálculo é feito comparando, linha a linha, cada par de códigos executados.

O PEEF exibe todas as execuções do estudante e marca em vermelho as linhas de código que sofreram modificação. Além disso, são exibidas *tags* em cada execução que representam o tipo do erro gerado (e.g., *SyntaxError*, *TypeError*, *IndentError*) ou sinalizando que não houve erro na execução (*NoError*). Este dado é extraído da mensagem original do compilador que também é armazenada. Além desta informação, é armazenada a linha em que o erro ocorreu. Esta funcionalidade permite ao professor analisar de modo comparativo a evolução do estudante ou identificar possíveis problemas em determinada atividade.

3.2.2 Chat. O ambiente permite que professores e estudantes conversem sobre uma determinada atividade sem sair do ambiente de edição. Cada atividade possui sua própria conversa e suas mensagens podem ser consultadas pelos estudantes durante todo o período do curso.

3.2.3 Mensagens Melhoradas (ECEMs) . A plataforma conta com mensagens melhoradas (ECEMs), previamente cadastradas. No entanto, professores podem cadastrar mensagens melhoradas para diversos tipos de erro. A base de dados inicial para obtenção de erros foi obtida nos trabalhos de Pritchard [18] e Jesus et al. [10]. As mensagens podem ser cadastradas por idioma, o que permite uma internacionalização dos cursos disponibilizados.

Uma mensagem melhorada está sempre associada a um tipo ou subtipo de erro e a sua mensagem original. Por meio do tipo e subtipo identificados em uma mensagem original do compilador/interpretador, PEEF recupera a mensagem melhorada associada e a exibe ao estudante. A Tabela 1 apresenta exemplos destas mensagens melhoradas.

3.2.4 Métricas. Em um curso, as métricas *EQ* e *RED* podem ser utilizadas para apoiar a análise de desempenho dos estudantes. Estas métricas podem ser acessadas em três formatos: i) visualização no ambiente; ii) em formato *JavaScript Object Notation* (JSON) ou iii) em formato *Comma-separated values* (CSV). Para *EQ*, são apresentados o identificador da atividade e os projetos. Para cada projeto, é apresentado o identificador do estudante e um *array* com os escores das sessões[2], no formato como no exemplo a seguir: { "activity": 1, "projects": [{"id": 1 , "scoresections": ["0.000","0.510"] } . Para *RED*, são disponibilizados o identificador da atividade, o identificador do estudante e um escore do projeto, como no seguinte exemplo: [{'id': '1', 'projects' : '[{'id': '1', 'score': '2.3333333333333'}]' }.

3.2.5 Logs. Além das métricas de erro, o ambiente coleta algumas informações que podem auxiliar na análise de comportamento dos estudantes e apoiar decisões pedagógicas. Para cada evento como entrar, sair, executar, testar e apresentar um erro, é gerado um log como o código do horário, do dia e a identificação do código editado. A Tabela 2 apresenta os logs possíveis e o seu significado.

3.2.6 Testes Unitários. Para cada atividade, o professor pode cadastrar testes unitários, fornecendo as entradas e saídas esperadas. Esses testes são armazenados no banco de dados e utilizados para exemplificação da atividade e execução dos testes. Existem dois tipos de testes possíveis de serem cadastrados pelo professor: os **testes públicos**, que são exibidos para os estudantes no resultado dos testes unitários, e os **testes ocultos**, que são executados no momento do teste, mas não são revelados para os usuários.

Assim, dentre o conjunto de testes cadastrados, apenas os testes públicos são apresentados aos alunos. Os demais testes são executados de forma oculta. Esta divisão evita que os estudantes tenham total conhecimento sobre os testes utilizados e burlem o processo de solução correta do problema.

3.3 Funcionalidades do Módulo Estudante

No módulo Estudante, os estudantes podem se matricular em cursos através de uma chave disponibilizada pelo professor, garantindo que apenas os alunos efetivos dos cursos acessem as atividades. Nos

[2]Utilizamos o conceito de sessão corrente de programação do trabalho de Jadud [8]

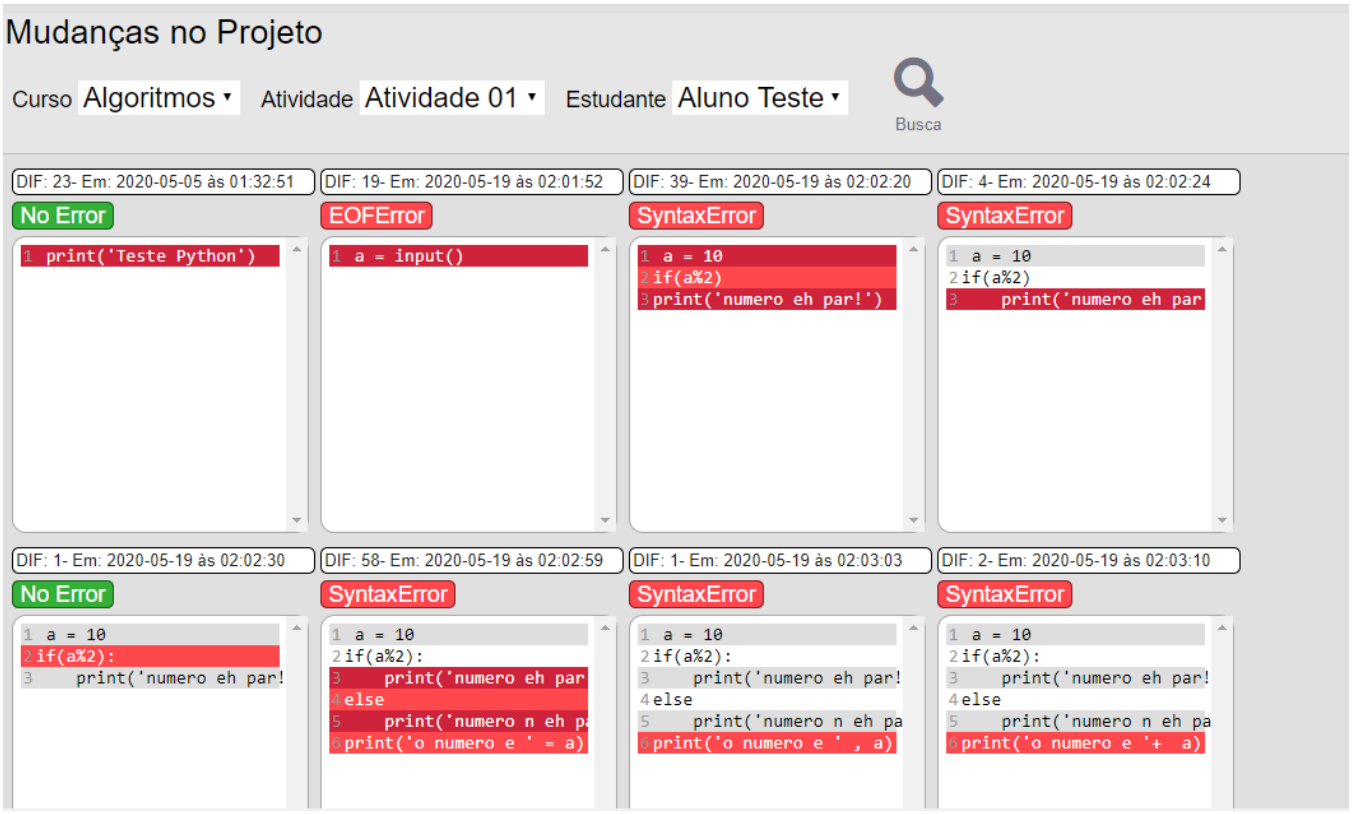


Figura 2: Cálculo do diff do Código de um estudante via STRING-EDIT

cursos, os estudantes têm acesso às aulas postadas pelos professores (descrição e vídeo). Na aba Atividades, os alunos podem acessar as atividades criadas para o curso. Para responder a uma atividade, os estudantes contam com um editor de código que é aberto ao clicar em uma atividade específica. A seguir, apresentaremos as funcionalidades do Módulo Estudante.

3.3.1 Editor de Código. O editor de código do PEEF, que pode ser visto na Figura 3a, permite criar, editar, salvar, baixar e executar arquivos Python. Todos os códigos editados são salvos no banco de dados, assim como todas as execuções. O editor de código permite a execução do código ou a execução dos testes unitários cadastrados pelo professor.

3.3.2 Testes Unitários. Para cada atividade, o estudante pode executar os testes unitários cadastrados pelo professor. Ao término dos testes, são exibidos os testes públicos executados com as entradas, saída esperada e saída atual, além da informação se o teste passou ou falhou. Adicionalmente é apresentada uma porcentagem dos testes que passaram em relação ao total de testes executados.

3.3.3 Mensagens Melhoradas. Ao executar um código com erros, o estudante obtém a mensagem padrão do compilador, adicionada de uma mensagem melhorada previamente cadastrada pelo professor. O idioma da mensagem considerará a configuração de perfil de cada estudante. Assim, um perfil com o idioma configurado para inglês receberá a mensagem em inglês, e um perfil com idioma português receberá a mensagem em português.

Caso o estudante faça a execução de um código com erro e, logo após, edite o seu código e faça a próxima execução sem erros, o ambiente emitirá um alerta sobre a utilidade da mensagem apresentada para que o código saísse de um estado de erro para um estado sem erro. Essa funcionalidade permite avaliar o grau de utilidade de cada mensagem melhorada.

3.3.4 Outras funcionalidades. Além dessas funcionalidades apresentadas, o estudante pode interagir com o professor através do chat. Além disso, os estudantes podem criar mais arquivos no editor, salvá-los e fazer o download de seu código.

4 EXEMPLO DE USO

Para exemplificar o uso do PEEF, apresentamos o fluxo de análise de métricas por um professor e o fluxo de realização de uma atividade por um estudante.

4.1 Exemplo de Uso do Professor

Para realizar esta ação, ao menos um estudante deve ter realizado uma atividade. O professor deve ter realizado o login na plataforma, acessado uma turma e uma atividade disponibilizada por ele.

4.1.1 Identificação de Estudante com dificuldades na atividade. Na tela principal da atividade, o professor pode visualizar a lista de

estudantes da turma. Para cada estudante na lista, é exibido: i) o percentual de testes unitários que não passaram em relação ao total de execuções dos testes; ii) o percentual de execuções com erros em relação ao total de execuções; e iii) as métricas de *EQ* e *RED*. Todos os dados são relacionados à atividade específica.

Por meio destes dados, o professor pode identificar estudantes com dificuldades (número alto de testes e/ou execuções malsucedidas e/ou métricas mais elevadas). Tendo identificado os estudantes com dificuldades, o professor deve clicar no item da lista que representa o estudante em questão e será encaminhado para uma tela com as edições do estudante.

4.1.2 Análise das Edições de Um Estudante. Na tela de edições de um estudante, o professor pode visualizar todas as versões do código do estudante que foram executadas ou testadas. De modo comparativo, o professor pode visualizar a distância de edição entre cada execução, através do valor do *diff*, como mencionado na Subseção 3.2.1.

Assim, é possível identificar uma sequência de execuções como erros similares (mesmo tipo do erro) em um local (linha do código) próximo, como as Execuções 6, 7 e 8 da Figura 2. Neste caso, aparentemente, o estudante está com dificuldade em concatenar uma string com um valor inteiro.

O professor pode enviar uma mensagem direta (*chat*) para o estudante, explicando sobre a forma correta de realizar este tipo de concatenação. A mensagem será exibida no Editor de Código, quando o estudante abrir a atividade.

4.2 Exemplo de Uso do Estudante

Para realizar esta ação, o professor deve ter disponibilizado ao menos uma atividade. O estudante deve ter realizado o login na plataforma, acessar uma turma e uma atividade disponibilizada pelo professor, já editada anteriormente.

4.2.1 Descrição da atividade e edição do código. O usuário deve clicar em Descrição para consultar o enunciado da atividade, informações sobre os formatos de entrada e saída, além de uma tabela com exemplos de entradas e saídas. Ao clicar no ícone de arquivo Python, o editor é aberto e é exibido o código editado anteriormente. O estudante edita o código no editor e pode salvá-lo, conforme a Figura 3a.

4.2.2 Execução do Código. O usuário executa o código clicando no botão Executar. Ao ser identificado um erro de execução, o erro original proveniente do compilador é exibido no console, juntamente com uma mensagem melhorada 4.

Em seguida, o usuário corrige o código e pode executá-lo novamente. Ao iniciar a execução, a possibilidade de escrita no console será ativada quando for solicitada alguma entrada. Ao digitar no console e pressionar a tecla Enter, a entrada é passada para a execução do programa. Ao término da execução, é exibida a mensagem de que a execução terminou. Ao término da execução bem sucedida, dada uma execução com erro seguida de uma execução sem erro, será exibida uma pergunta sobre a efetividade da mensagem melhorada apresentada.

4.2.3 Execução de Testes Unitários. Após editar seu código, o usuário poderá também executar os testes unitários cadastrados para a atividade. Para executar os testes, o usuário precisa clicar no botão Testar. O código é executado através dos testes e são exibidos os casos de testes públicos com informações de entrada, saída esperada e saída atual, conforme a Figura 3b. O usuário pode editar o seu código e executar os testes novamente. A porcentagem de 100%, quanto aos testes, indica que o estudante possui uma solução válida para a atividade disponibilizada.

5 TRABALHOS RELACIONADOS

Os trabalhos relacionados podem ser divididos em ferramentas que adotam o uso de mensagens melhoradas e ferramentas que adotam o uso de testes unitários.

5.1 Mensagens do Compilador Melhoradas

Estudantes frequentemente sentem-se confusos com mensagens fornecidas por compiladores, dada a sua falta de clareza e imprecisão [2]. Por este motivo, pesquisadores tendem a analisar como estas mensagens afetam a aprendizagem e como elas podem ser melhoradas. Ferramentas que visam melhorar mensagens de compiladores existem desde a década de 1960, como *Ditran*, o *Code Analyser for Pascal* (1990) e o *Helpmeout* (2010) [7]. Estas ferramentas utilizam diversas técnicas como mensagens pré-catalogadas, informações adicionais, soluções de pares para problemas similares, dentre outras.

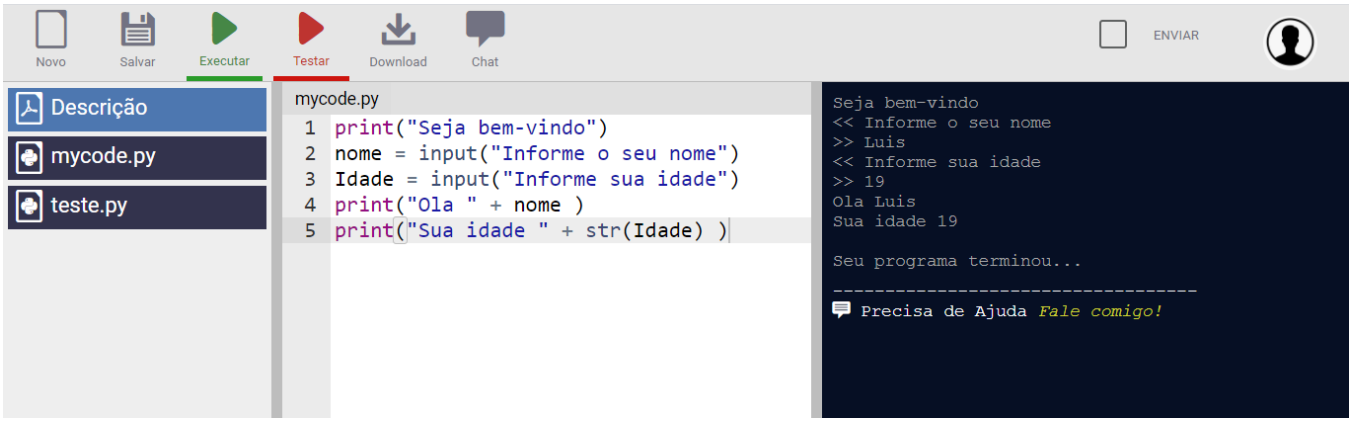
A emissão de mensagens melhoradas provenientes de um catálogo de erros emitidos pelo *BlueJ* é uma das estratégias utilizadas por Becker (2016) [1]. Adicionalmente, Denny et al. (2014) apresentam uma explicação e exemplos incorretos e corretos do uso de um determinado conceito, usando a ferramenta de código aberto *WriteCode* [6]. Estes estudos mais recentes destacam-se pelo uso de métricas como *EQ* [8] e *RED* [1], que visam avaliar o potencial da intervenção ou prever alunos em risco.

Percebe-se, no entanto, que as mensagens melhoradas focam principalmente em erros sintáticos, falhando em atacar os erros lógicos ou bugs, na perspectiva de Kohn [11]. Neste sentido, pesquisadores estudam também o *feedback* sobre as soluções a problemas propostos (*feedback* lógico e semântico). Uma destas abordagens ocorre através do uso de testes unitários automatizados.

Kohn and Manaris apresentam o TigerJython, um ambiente *desktop* de programação para Python focado em estudantes do ensino fundamental e baseado em Jython [12]. Este ambiente modifica o tratamento de entrada de dados e a exibição de saídas e mensagens. Como principal funcionalidade, TigerJython apresenta mensagens melhoradas, previamente cadastradas.

5.2 Testes Unitários

O uso de testes unitários para verificar a corretude do código e fornecer *feedback* é um abordagem utilizada no ensino de programação. Craig and Petersen apresentam um sistema que aceita submissões dos estudantes, executa o código e, em caso de não obter erros, fornece detalhes sobre testes executados sobre o código [5]. Lappalainen et al. apresentam um alternativa ao *JUnit* para diminuir as dificuldades dos estudantes [13]. Inicialmente, os estudantes foram apresentados aos Desenvolvimento Dirigido por Testes, enquanto realizavam exercícios com testes para o *plugin ComTest*.



(a) Execução de Código no PEEF

(b) Execução de Testes Unitários no PEEF

Figura 3: PEEF Editor

Alguns dos estudos apresentados utilizam ferramentas de uso geral como *BlueJ*, um ambiente educacional para o ensino de Java, *WriteCode*, ou ferramentas profissionais como *Eclipse* e *JUnit*. Além disso, como apresentam Luxton-Reilly et al., muitas destas ferramentas de *feedback* procuram fornecer um suporte geral, diminuindo o trabalho do professor [14]. Embora muitas ferramentas busquem fornecer vários tipos de *feedback*, segundo os autores, não foram encontrados trabalhos que visem comparar este tipos ou a diferença de eficácia de *feedbacks* gerais e direcionados.

6 CONCLUSÕES

Este trabalho apresentou um ambiente intitulado *Python Enhanced Error Feedback* (PEEF), que visa fornecer diversas formas de *feedback* sobre erros de estudantes novatos. A plataforma PEEF pode ser utilizada para a prática do ensino e aprendizagem de programação em diferentes modalidades, tais como o ensino presencial, o ensino remoto e o ensino a distância. Enquanto vantagens pedagógicas, PEEF permite que estudantes acessem aulas de modo remoto e assíncrono, revisitando as aulas enquanto solucionam os problemas propostos nas atividades. Estudantes podem editar códigos, salvá-los, executá-los e testá-los de acordo com testes cadastrados pelos professores.

O professor pode acompanhar os seus alunos, amparado por métricas de erro e modificação do código, além de *logs* de usuários. O módulo de visualização de mudanças permite que o professor identifique grandes mudanças entre códigos ou possíveis dificuldades de estudantes. Além disso, a plataforma permite uma comunicação entre estudante e professor e a adição de testes unitários como suporte ao estudante. Destaca-se que a integração de diversas funcionalidades dá ao ambiente apresentado um diferencial em relação a outras ferramentas.

Em trabalhos futuros, pretende-se aplicar o ambiente PEEF em cursos de Python online com estudantes novatos para estudos sobre os efeitos de *feedback* melhorado. Pretende-se avaliar comparativamente as diferentes formas de feedback oferecidos pela ferramenta e o potencial de cada uma delas para a melhoria do código de estudantes novatos e para a redução das dificuldades na aprendizagem de programação.

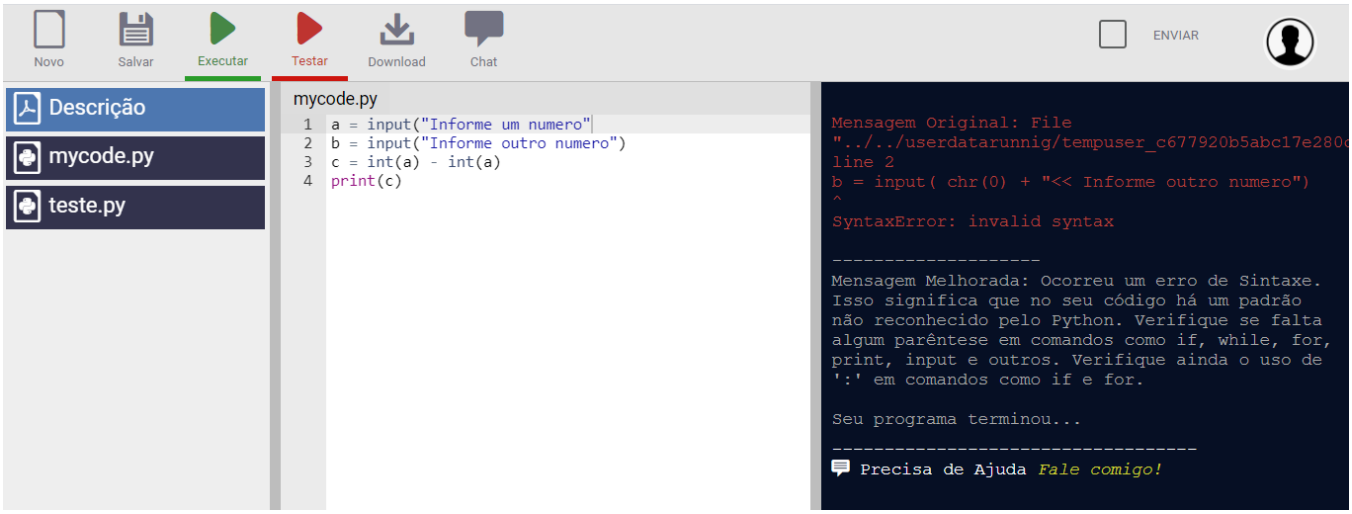


Figura 4: Exibição das Mensagens Original e Melhorada

REFERÊNCIAS

[1] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 126–131. https://doi.org/10.1145/2839509.2844584

[2] Brett A. Becker, Cormac Murray, Tianyi Tao, Changheng Song, Robert McCartney, and Kate Sanders. 2018. Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages. In *Proceedings of the 49 th ACM Technical Symposium on Computing Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 634–639. https://doi.org/10.1145/3159450.3159453

[3] J. Bennedsen and M. E. Caspersen. 2007. Failure Rates in Introductory Programming. *SIGCSE Bull.* 39, 2 (June 2007), 32–36.

[4] Yorah Bosse and Marco Gerosa. 2015. Reprovações e Trancamentos nas Disciplinas de Introdução à Programação da Universidade de São Paulo: Um Estudo Preliminar. In *Anais do XXIII Workshop sobre Educação em Computação* (Recife). SBC, Porto Alegre, RS, Brasil, 426–435. https://doi.org/10.5753/wei.2015.10259

[5] Michelle Craig and Andrew Petersen. 2016. Student Difficulties with Pointer Concepts in C. In *Proceedings of the Australasian Computer Science Week Multiconference* (Canberra, Australia) *(ACSW '16)*. Association for Computing Machinery, New York, NY, USA, Article 8, 10 pages. https://doi.org/10.1145/2843043.2843348

[6] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing Syntax Error Messages Appears Ineffectual. In *ITiCSE Í4 Proceedings of the 2014 conference on Innovation & technology in computer science education* (Uppsala, Sweden) *(ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 273–278. https://doi.org/10.1145/2591708.2591748

[7] Maria Janaina da Silva Ferreira et al. 2015. *EMS: um plug-in para exibição de mensagens de erro dos compiladores.* Ph.D. Dissertation. Universidade Federal de São Carlos.

[8] Matthew C Jadud. 2005. A first look at novice compilation behaviour using BlueJ. *Computer Science Education* 15, 1 (2005), 25–40.

[9] David Janzen and Hossein Saiedian. 2008. Test-Driven Learning in Early Programming Courses. *SIGCSE Bull.* 40, 1 (March 2008), 532–536. https://doi.org/10.1145/1352322.1352315

[10] Galileu Jesus, Kleber Santos, and Jaine Conceição e Alberto Neto. 2018. Análise dos erros mais comuns de aprendizes de programação que utilizam a linguagem Python. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)* 29, 1 (2018), 1751. https://doi.org/10.5753/cbie.sbie.2018.1751

[11] Tobias Kohn. 2019. The Error Behind The Message: Finding the Cause of Error Messages in Python. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. Association for Computing Machinery, Minneapolis, MN, USA, 524–530.

[12] 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*. Association for Computing Machinery, Portland, OR, USA, 1054–1060.

[13] Vesa Lappalainen, Jonne Itkonen, Ville Isomöttönen, and Sami Kollanus. 2010. ComTest: A Tool to Impart TDD and Unit Testing to Introductory Level Programming. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (Bilkent, Ankara, Turkey) *(ITiCSE '10)*. Association for Computing Machinery, New York, NY, USA, 63–67. https://doi.org/10.1145/1822090.1822110

[14] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *ITiCSE 2018 Companion: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[15] Andy Nguyen, Christopher Piech, Jonathan Huang, and Leonidas Guibas. 2014. Codewebs: Scalable Homework Search for Massive Open Online Programming Courses. In *WWW '14: Proceedings of the 23rd international conference on World wide web* (Seoul, Korea) *(WWW '14)*. Association for Computing Machinery, New York, NY, USA, 491–502. https://doi.org/10.1145/2566486.2568023

[16] David J Nicol and Debra Macfarlane-Dick. 2006. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education* 31, 2 (2006), 199–218.

[17] Claudia Ott, Anthony Robins, and Kerry Shephard. 2016. Translating principles of effective feedback for students into the CS1 context. *ACM Transactions on Computing Education (TOCE)* 16, 1 (2016), 1–27.

[18] 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, New York, NY, USA, 1–8.

[19] Lev Semynovitch Vygotsky. 2008. *A formação social da mente.* Martins Fontes, São Paulo.

[20] Christopher Watson and Frederick W.B. Li. 2014. Failure Rates in Introductory Programming Revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (Uppsala, Sweden) *(ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 39–44. https://doi.org/10.1145/2591708.2591749