Uma avaliação comparativa entre métricas de erro em um curso introdutório de programação com Python

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

O *feedback* oferecido por mensagens de erro produzidas por compiladores e interpretadores tipicamente não é suficiente para apoiar os estudantes novatos. A inserção de *feedback* adicional nestas mensagens, na forma de mensagens de erro melhoradas, pode ser uma solução para melhor apoiar os estudantes novatos na aprendizagem de programação. No entanto, o oferecimento deste *feedback* adicional necessita de avaliação acerca da sua eficácia. Para esta avaliação, diversas técnicas são utilizadas, desde abordagens manuais a automatizadas, como as métricas de erro dos estudantes. Da literatura, destacamos as métricas *Error Quotient* (EQ), *Watwin Algorithm* e *Repeated Error Density* (RED). Apesar da existência destas métricas, questões relacionadas à avaliação da eficácia das mensagens permanecem em aberto. Neste artigo, objetivamos comparar as métricas existentes e propor uma nova métrica com base nos achados. Para tal fim, foi ofertado um curso introdutório de Python no ambiente PEEF. Por meio dos dados obtidos no ambiente, as métricas foram mensuradas para todos os projetos. Como resultado, ratificamos a dependência de contexto das métricas EQ e Watwin, além da adequação de RED para abordagens de programação introdutória com Python. Entretanto, destacamos algumas limitações desta última e propomos uma nova métrica, *RECurrent Error Density* (REC), visando melhor responder ao questionamento sobre o impacto das mensagens melhoradas na aprendizagem dos estudantes novatos. Os resultados demonstram que REC consegue mensurar erros em 43,3% dos projetos dos estudantes, atribuindo avaliação a 30,9% de projetos a mais que a métrica RED. Estes resultados demonstram que o fenômeno de recorrência de erro ocorre e não é considerado adequadamente pelas métricas apresentadas.

CCS CONCEPTS

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

PALAVRAS-CHAVE

Educação de computação, Brasil, EduComp

EduComp'22, Abril 24-29, 2022, Feira de Santana, Bahia, Brasil (On-line)

1 INTRODUÇÃO

Dificuldades na aprendizagem de programação são um problema já relatado por diversos autores [9]. Em consequência deste problema, muitos pesquisadores buscam formas de compreender e reduzir as dificuldades dos estudantes. Dentre as diversas abordagens propostas, podemos destacar a escolha de linguagens de programação mais apropriadas aos novatos. Alguns estudos buscam inserir a programação com linguagens baseadas em blocos ou linguagens textuais com objetivos educacionais [2]. Outros estudos sugerem o uso de linguagens comerciais como Python [14]. Segundo Grandell et al. [8], Python é uma linguagem de alto nível, originalmente criada para facilitar a aprendizagem. Embora a linguagem Python tenha vantagens quanto à sua sintaxe e simplicidade, esta linguagem possui um grupo limitado de tipos de erros, fator que leva o estudante a se deparar com mensagens frequentemente vagas. Este problema faz com que mensagens emitidas pelo interpretador não enunciem o erro subjacente, levando o estudante a um estado de confusão e frustração, além de requerer maior tempo do professor para suporte em sala de aula [14].

Problemas relacionados às mensagens de erro do compilador ou interpretador são reportados há cerca de 40 anos [4]. Estes problemas fazem com que pesquisadores busquem formas de aprimorar as mensagens de erro fornecendo *feedback* adicional. Uma solução proposta é a criação de mensagens melhoradas (em inglês, *Enhanced Compiler Error Messages* – ECEM). ECEMs são mensagens de compilador ou interpretador com informações adicionais sobre a causa do erro, normalmente acompanhadas da mensagem original e traduzidas para o idioma do estudante. Na literatura, a criação de ECEMs é realizada por uma variedade de técnicas e para diversas linguagens, como Java, C ou Python.

Tendo em vista a inserção de informações adicionais para minimizar as dificuldades dos estudantes, surge a necessidade de avaliar a eficácia deste método. A comunidade acadêmica desenvolveu algumas métricas como Error Quotient (EQ), *Watwin Algorithm* e Repeated Error Density (RED) e as utilizam para avaliar o comportamento do estudante a respeito dos erros reportados, mais especificamente, para avaliar a adequação das mensagens de erro emitidas. Estas métricas possuem uma variedade de aplicações, pois além de avaliar a eficácia das mensagens melhoradas, podem prever o desempenho dos estudantes, permitindo o acompanhamento pedagógico em tempo real. No entanto, muitas métricas não são de uso geral por dependerem de contextos específicos vinculados à sua criação, tais como a linguagem de programação utilizada pelos estudantes ou a complexidade das atividades propostas. Os estudos em torno desta temática têm demonstrado resultados positivos, no entanto, segundo Denny et al. [6], esta área ainda possui uma

lacuna quanto à estratégia adequada para mensurar a eficácia das mensagens melhoradas, fato também enunciado por Kohn [13].

Entendendo a importância de uma métrica que capture situações relevantes ao contexto de aprendizagem dos estudantes, buscamos neste trabalho avaliar de modo comparativo as métricas já existentes como EQ, RED e *Watwin* e propor uma nova métrica.

Para este fim, realizamos um estudo de caso com estudantes participantes de um curso de programação introdutória com a linguagem Python utilizando o Ambiente PEEF. Como resultado, ratificamos que o RED é mais adequado que o EQ e *Watwin Algorithm* para contextos similares ao apresentado. No entanto, de modo exploratório, apontamos comportamentos não identificados pela métrica RED e propomos uma nova métrica para mensurar a eficácia das mensagens melhoradas motivados pelas limitações apontadas. Além disso, apresentamos a lista dos erros com maior número de ocorrências, além da lista de erros recorrentes na nossa abordagem.

O presente trabalho está dividido em seis seções. A primeira apresenta uma introdução do trabalho. Os trabalhos relacionados são apresentados na segunda seção. A terceira seção apresenta a metodologia empregada, os participantes, o planejamento, o processo de coleta e análise dos dados. A quarta seção apresenta uma nova proposta de métrica. Os resultados são apresentados na quarta seção e discutidos na quinta. Por fim, as conclusões são apresentadas na sexta seção.

2 TRABALHOS RELACIONADOS

A inserção de mensagens melhoradas do compilador/interpretador não é algo novo. Segundo Becker et al. [5], pesquisas nesta área datam de cerca de 40 anos. Em 1995, Schorsch apresentou um analisador para a linguagem Pascal (CAP) que fornecia mensagens melhoradas com informações sobre o erro, o motivo da ocorrência e como corrigi-lo [20]. Outra abordagem proposta por Hristova e colaboradores visou incorporar um pré-compilador que escaneia o código Java mapeando 20 erros mais comuns e oferecendo ECEMs quando possível [10]. Já Kohn apresentou um *parser* em Python que mapeia os principais erros sintáticos (*Syntax Error*) e fornece uma mensagem mais específica para cada um dos erros mapeados [13]. Uma abordagem anterior nossa utiliza um banco de dados de mensagens vinculadas a erros e suberros que podem ser configuradas pelo professor de acordo com o idioma especificado, linguagem e curso [1]. Dada uma execução com o erro, verificada-se se há uma mensagem melhorada para ser exibida ao estudante [1].

Paralelamente a essas abordagens pedagógicas, surgiu a necessidade de avaliar o desempenho dos estudantes mediante o contato com as mensagens melhoradas. Assim, Jadud propôs uma métrica chamada *Error Quotient* (EQ) para quantificar aspectos relacionados ao comportamento dos estudantes novatos [11]. Esta métrica visa quantificar o comportamento do estudante novato a partir de suas compilações, levando em conta o tipo de localização do erro, assim como a sua frequência, para produzir um escore normalizado ([0, 1]) para todas as sessões do estudante. Jadud considera uma sessão um conjunto de no mínimo sete compilações, sendo finalizada caso o intervalo de tempo entre compilações ultrapasse um valor alfa especificado. Caso o estudante feche o BlueJ (ferramenta utilizada no estudo) e retorne em um intervalo de cinco minutos, estes eventos são considerados como uma continuação da sessão anterior. Este tempo é considerado pois os estudantes podem ter problemas técnicos no uso da ferramenta.

O algoritmo de EQ possui quatro etapas: Coleta (cria pares consecutivos de eventos (e_1, e_2), (e_2, e_3), (e_3, e_4) até $(e_n - 1, e_n)$; Cálculo: cria um escore para cada par com base no algoritmo proposto; Normalização: divide cada escore calculado por 11 – o máximo de pontuação possível. e Média: a soma dos escores dividida pelo número de pares gerados. Esta média calculada é o valor EQ para a sessão.

Como é possível perceber em Jadud [11], dado um par de eventos que resultaram em erro, o escore recebe o valor 8. Caso estes erros sejam do mesmo tipo, é adicionado mais 3 ao escore. EQ pode ser utilizado como uma métrica para calcular a eficiência de mensagens melhoradas, já que um conjunto de mensagens melhoradas eficientes reduziria a quantidade de erros cometidos pelos estudantes e, por consequência, o escore EQ.

A métrica *Watwin* foi proposta por Watson et al. [22]. Este algoritmo considera o tempo gasto por um estudante para resolução de um erro em relação ao tempo médio gasto por todos os estudantes para a solução do mesmo erro. *Watwin* surge da observação dos autores sobre limitações da EQ, em especial questões relacionadas a múltiplos arquivos e tempo de resolução.

Por esta razão o algoritmo de *Watwin* possui uma sequência de etapas similar a EQ: Preparação (criação de pares de eventos simultâneos); Quantificação do Comportamento (para cada par de eventos, aplica-se o algoritmo); Normalização (divide-se a soma do escore por 35); Média (a soma dos escores é dividida pelo número de pares gerados).

No processo de Preparação, o algoritmo considera a hipótese de o estudante trabalhar em múltiplos arquivos, agrupando os eventos por arquivo. Pares de eventos com o mesmo *snapshot* (amostra de código) são removidos, assim como eventos em que a métrica de inserção ou mudança é igual a zero e de deleção for maior que zero. As mensagens fornecidas pelo compilador são generalizadas para que possa ocorrer uma comparação entre ela, não levando em conta informações de identificadores. Por fim, é estimado o tempo para cada par de eventos (e_f, e_t). Considerando ainda que o estudante pode intercalar o seu trabalho entre vários arquivos, o algoritmo *Watwin* combina pares de eventos e de invocações (h_1,h_2, .. h_n), ordenados pelo *timestamp*. Caso haja um h_i cujo timestamp é $e_f > h_i > e_t$ o tempo gasto em (e_f, e_t) é a diferença entre e_t e h_i. Após isso, é aplicada uma penalidade baixa para estudantes que corrigem um tipo de erro em um tempo abaixo da média dos demais e uma penalidade alta para um tempo maior que a mesma média. Detalhes deste algoritmo pode ser visto em Watson et al. [22].

Em ambos os algoritmos EQ e Watwin, os pesos foram definidos por meio de uma busca por força bruta e foram utilizados para previsão de desempenho. No entanto, segundo Becker et al. [5], o algoritmo EQ possui algumas limitações. A primeira limitação identificada é a sua dependência de contexto. EQ necessita de atividades em que os estudantes trabalhem um período suficiente para gerar sessões válidas. Além desta limitação, Becker et al. apresentam situações em que EQ não consegue gerar escores distintos para perfis distintos de estudantes. Ou seja, falha na sua avaliação.

Tendo isso em vista e visando responder à pergunta colocada por Jadud [11] "*Se um estudante falha em em corrigir um erro de*

Tabela 1: Valores de EQ e RED para algumas sequências de erro.

estudante	sequência S	x	r	s	EQ	RED
1	x	1	0	1	-	0
2	... x ... x ...	2	0	2	-	0
3 (A)	... x x ...	2	1	1	1	0.5
4 (B)	... x x x ..	3	2	1	1	1
5 (C)	... x x ... x x ...	4	2	2	1	1.3

'início ilegal de expressão' no decurso de três compilações, e um outro, no decurso de 10 compilações, o segundo estudante é [cerca] de três vezes pior que o outro? E se um estudante lida com uma cadeia ininterrupta de erros, e a repetição deste erro particular é apenas uma de várias?", Becker et al. propuseram uma nova métrica [5]. A métrica proposta leva em consideração os erros repetidos, tendo em vista que haver repetições de erro é um possível sinal de que o estudante está enfrentando problemas. Assim, Becker et al. instituíram uma métrica que visa calcular a densidade do erro repetido, intitulada de *Repeated Error Density* (RED). O valor de RED para uma dada sequência S de n erros repetidos é a soma de $r_i^2/(r_i+1)$ para cada sequência S_i em S, e r_i é o número de erros repetidos na sequência S_i. Detalhes da métrica podem ser vistos em Becker [3].

A Tabela 1 é uma adaptação das Tabelas 1 e 2 do trabalho de Becker [3]. A Tabela 1 apresenta situações em que EQ falha na distinção de estudantes e, ao mesmo tempo, o valor da métrica atribuída por RED é distinto. Nesta tabela, x é um erro que se repete dentro das sequências S_i. É possível notar, pelo S_2, que Becker está interessado apenas em erros repetidos de modo consecutivo. RED também pode ser utilizada para avaliar a eficácia das mensagens melhoradas. Esta métrica foi utilizada em um estudo realizado por Becker et al. [5] com aproximadamente 100 estudantes. No experimento controlado, os estudantes foram divididos em dois grupos: um grupo teve acesso às mensagens melhoradas e outro teve acesso apenas às mensagens originais emitidas pelo compilador. Becker et al. encontraram evidência que a mensagem melhorada reduz o número de erros por estudante e que a métrica RED é capaz de identificar este fenômeno.

Diferentemente de EQ e *Watwin*, RED é menos suscetível a contexto, por não considerar outros parâmetros como o tempo e divisão por sessões, sendo, portanto, ideal para códigos menores que requerem um menor esforço do programador.

No entanto, Rodrigo e colaboradores [19] publicaram um estudo sobre a aplicação de protocolos online para a previsão do desempenho dos estudantes. Segundo este estudo, EQ e Watwin tiveram melhor desempenho que RED. Apesar de ser um dos poucos estudos sobre a utilização de RED como preditor, os autores acendem um alerta sobre o desempenho da métrica proposta por Becker. O desempenho e a adequação desta métrica podem estar associados ao fato de considerar apenas os erros repetidos de modo local.

Kohn and Manaris [14] apresentam uma abordagem com a utilização da ferramenta TrigerJython, desenvolvida inicialmente para desktop. Por meio desta ferramenta, os estudantes podem enviar dados sobre os seus códigos e erros para um banco de dados da ferramenta. Além de outras funcionalidades como mudanças nas funções de entrada e saída e adição do loop Repeat (não presente no Python), TrigerJython fornece mensagens melhoradas aos usuários. Suas mensagens são traduzidas do inglês para a língua nativa do programador e fornecem algumas dicas adicionais.

Em outro trabalho, Khon [13] apresenta uma análise das reações dos estudantes sobre as mensagens de erro, adotando uma metodologia de classificação dos *snapshots*. As classificações possíveis são Deleção [*del*], Reescrita [*rwr*] e Correção [*fix*]. Khon aponta que as mensagens melhoradas ajudam os estudantes, pois eles corrigem o erro em mais de 80% dos casos. Apesar disso, apenas 25% dos estudantes seguem o que é proposto pela mensagem de erro [13]. Por fim, embora rica, a análise de Khon é comprometida tendo em vista que os objetivos das atividades são desconhecidos, não permitindo fazer uma análise mais aprofundada dos dados. Além disso, a sua abordagem não é automatizada, o que dificulta seu uso para um grande volume de dados.

3 METODOLOGIA

Esta seção descreve a linguagem e ferramentas utilizadas, o planejamento do curso introdutório de Python e os procedimentos de coleta e análise de dados.

3.1 Linguagens e Ferramentas

Com base em estudos anteriores realizados pelos autores [2] e em trabalhos de outros pesquisadores [8, 15], a linguagem Python foi selecionada por suas características adequadas ao ensino introdutório de programação. Após a escolha da linguagem, foi preciso escolher um ambiente adequado aos novatos. Existem inúmeros ambientes que dão suporte ao ensino de programação Python para novatos como o TrygerPython e JES. No entanto, nenhuma destas ferramentas permite obter o conjunto de dados específico de nosso interesse e, ao mesmo tempo, incorporar um curso à plataforma de programação.

Pelo exposto, decidimos utilizar o PEEF, um ambiente idealizado pelos autores que permite: acesso a aulas em formato de vídeos; atividades disponibilizadas pelo professor com descrição; exemplos de entrada e saída; ambiente de programação com chat, mensagens melhoradas em português e testes unitários disponíveis aos estudantes. Para o professor, o PEEF oferece um controle da turma, permitindo visualizar em tempo real os avanços de cada estudante, além de prover acesso a métricas como quantidade de execuções, erros, testes e códigos. Mais informações sobre o PEEF são descritas em [1].

3.2 Participantes

Para a execução do estudo de caso, um curso introdutório em Python foi idealizado e disponibilizado para estudantes de uma universidade privada. Para se candidatar, os estudantes preencheram um questionário inicial com os seus dados para contato e receberam por e-mail o Termo de Consentimento Livre e Esclarecido (TCLE). Após o aceite do estudante, um segundo questionário com questões relacionadas ao perfil e experiência relacionada a *feedback* do compilador/interpretador foi enviado. Após a respostas deste segundo questionário, foi encaminhado o login de acesso ao sistema PEEF. Tivemos 44 estudantes interessados em participar do curso. Porém, apenas 25 aceitaram os termos do TCLE e preencheram o segundo questionário e 17 estudantes confirmaram o recebimento do e-mail

Tabela 2: Dados gerais do Curso

Período do Curso	26/07/21 a 30/08/21
Aulas Gravadas	~3h (178 minutos e 32 seg)
Encontros síncronos	4
Semanas	4
Atividades	40
Total de Testes Unitários	109
Total de Mensagens Melhoradas	169

com credenciais. Por fim, 12 estudantes participaram do curso efetivamente. Cinco estudantes que confirmaram o e-mail relataram problemas variados como falta de tempo para se dedicar ao curso e problemas de saúde. Todos os 12 estudantes que participaram do curso são do sexo masculino (4 estudantes do sexo feminino receberam o e-mail com credenciais, 2 responderam ao e-mail, mas não prosseguiram com o curso pelos motivos já informados). Todos os estudantes cursam o ensino superior. 10 estudantes ingressaram no ano de 2020, os demais ingressaram no ano de 2018 e 2021 respectivamente. Neste trabalho, nos referimos a cada um dos doze estudantes por meio da letra E acompanhada de uma numeração sequencial.

Sobre o nível de programação pré-curso, 50% dos estudantes afirmam conseguir resolver problemas com algoritmos, 16% nunca tinham programado ou estava aprendendo programação no início do curso. 50% dos estudantes não tinham conhecimentos sobre Python e 25% sabiam muito pouco (entrada, saída, e operações básicas). 17% dos estudantes não tinham nenhum conhecimento em inglês (língua padrão das mensagens de erro) e 58% dos estudantes conseguiam ler textos em inglês com auxílio de um dicionário. Quanto à experiência com mensagens de erro, 17% não as leem, pois não entendem inglês, 17% as leem, mas não as entendem, 33% revelam que a leitura da mensagem não é uma prática e 33% entendem o que a mensagem quer dizer e então resolvem o problema. Embora a maioria dos estudantes já tenham tido algum contato com programação, mesmo que mínimo, a maioria não conhecia a linguagem Python e não consideram a leitura das mensagens de erro uma tarefa fácil.

3.3 Planejamento

Diferentemente de outros trabalhos que coletam códigos e dados de execuções por meio de base de dados de ferramentas como BlueJ ou TrygerPython, nós idealizamos um curso introdutório de Python a fim de avaliar o comportamento dos estudantes. O curso ocorreu dentro da plataforma PEEF e os estudantes tiveram acesso às aulas e atividades. Foram idealizadas 23 aulas disponibilizadas em formato de vídeo, totalizando aproximadamente três horas de conteúdo, durante quatro semanas. Foram programados quatro encontros síncronos/remoto para solucionar dúvidas. A Tabela 2 apresenta o planejamento de modo detalhado.

Para cada atividade disponibilizada, foi criado um conjunto de testes unitários. Foram criadas quantidades variadas de testes para cada questão, dependendo da complexidade da questão e das possibilidades de entrada e saída. As mensagens melhoradas cadastradas foram idealizadas pelos pesquisadores com base nos principais erros relatados por Pritchard [18] e Jesus et al. [12]. Desse modo, não consideramos todos os erros possíveis em Python. Vale observar que Python não possui um repositório com todas as mensagens de erro emitidas, pois este elemento faz parte da implementação da linguagem.

Tabela 3: Planejamento do Curso

Semana	**Conteúdo**
Semana I	Algoritmos; O Ambiente PEEF; Erros e Testes Unitários; Saída de Dados; Operações Aritméticas.
Semana II	Entrada de Dados; Operadores Relacionais; Condicionais (if, else, elif); Condicionais aninhadas.
Semana III	Repetição for; Repetição while.
Semana IV	Listas; Funções sem retorno; Funções com retorno; Funções com parâmetros.

A Tabela 3 apresenta os conteúdos do curso divididos por semana.

3.4 Coleta e Análise de Dados

Em paralelo à execução do curso ocorreu uma pesquisa exploratória. Inicialmente coletamos dados demográficos e sobre o perfil dos estudantes por meio de um questionário (pré-intervenção). Durante o curso, coletamos dados por meio da plataforma PEEF. Dentre os dados coletados, estavam eventos de execução, códigos submetidos, informações relacionadas ao erro obtido ou ausência dele, além da data e hora da execução. Outras informações que fogem ao escopo desse artigo foram coletadas, como navegação pela plataforma, interação com as aulas, dados de conversa no chat, entrega da atividade, dentre outros.

Ao final das quatro semanas, os estudantes realizaram uma avaliação de múltipla escolha que visou compreender como os conhecimentos sobre algoritmos e linguagem Python foram construídos. Após a intervenção, os estudantes responderam a um questionário com questões sobre perfil e sobre a experiência com o uso da plataforma[1]. Após o término do curso, foram extraídos dados da plataforma, tal como a quantidade de projetos criados, projetos entregues, quantidade de erros e resultados de testes, dentre outros. Por meio dos dados relacionados às execuções, foram calculadas as métricas EQ e RED, visando avaliar a sua adequação. Elencamos ainda dados sobre a ocorrência dos principais erros, e calculamos o desempenho dos estudantes nos testes e nos projetos. A Figura 1 apresenta um resumo da metodologia.

Os dados quantitativos foram analisados por meio de estatística descritiva e métricas específicas. Utilizamos exemplos de códigos para compreender o comportamento dos estudantes.

4 RECURRENT ERROR DENSITY - REC

Com base nos pontos apresentados na Seção 2 sobre as limitações das formas de avaliação da mensagens melhoradas, apresentaremos uma proposta de métrica adequada a um curso introdutório de programação. Esta métrica considera não apenas a reação do estudante à mensagem de erro de modo local, mas também a interação do estudante com mensagens de erro melhoradas ao longo do processo de aprendizagem de programação durante as atividades. Tomamos

[1] https://sites.google.com/view/educomp2022rec/

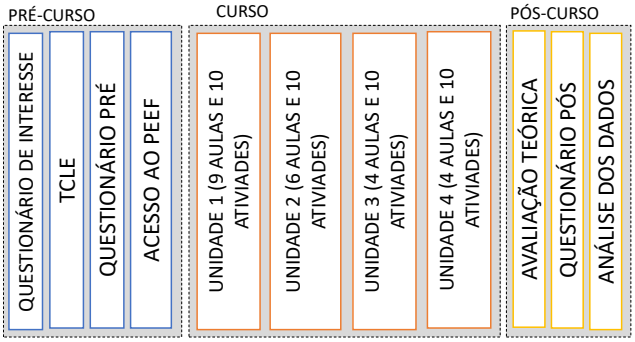

Figura 1: Sequência de passos metodológicos

como ponto de inicial a métrica RED, partindo do pressuposto elucidado por Becker [3]. Na Seção 5 apresentaremos de modo mais aprofundado esta adequação.

4.1 Motivação

A fim de esclarecer pontos importantes da métrica proposta, iremos inicialmente analisar a métrica RED e suas limitações enquanto ponto de motivação para a criação de uma nova métrica.

Segundo a análise de Rodrigo et al., RED não possui um bom desempenho como preditor [19]. Embora o aspecto de predição não seja o ponto central no estudo da eficácia das mensagens melhoradas, há de se esperar que uma boa métrica do comportamento dos estudantes frente a dificuldades de programação também seja minimamente eficaz na previsão do desempenho.

Um segundo aspecto é que RED não avalia o reaparecimento do erro de modo completo, pois foca apenas em erros consecutivos, tal como demonstrado. Ou seja, a métrica proposta por Becker [3] não considera o impacto da mensagem melhorada para além da análise local. Assim RED não se enquadra como uma boa métrica, segundo Kohn [13], já que "*Em particular, uma boa métrica deve mensurar se o estudante é capaz de corrigir o erro real, em vez de apenas reagir à CEM*" [13].

Dentro deste contexto, percebe-se que não é suficiente analisar a reação do aluno à mensagem (análise local). É preciso olhar adiante e investigar como a mensagem afeta a aprendizagem do estudante em curto ou longo prazo. Ou seja, avaliar se o estudante é capaz de solucionar o problema após compreender o erro.

Diferentemente do que RED avalia, a nossa hipótese é que se o estudante não construiu conhecimento sobre o que ocasionou o erro, o erro tornará a aparecer em algum momento e não necessariamente na próxima compilação.

De modo complementar ao que Kohn [13] apresenta sobre a reação à ECEM, destacamos alguns cenários no comportamento dos estudantes que comprometem a avaliação de erros consecutivos, considerados por RED. Após ter acesso à ECEM, o estudante pode: i) solucionar o erro por meio de pesquisa na internet ou outro material; ii) solucionar o erro mediante ajuda de um colega; iii) solucionar o erro mediado pelo suporte do professor ou tutor; iv) solucionar o erro ao acaso; v) adicionar outro erro ao tentar solucionar o primeiro erro; vi) remover a linha que causa o erro. Para todos esses casos, uma análise local, como a de RED, avalia que o estudante compreendeu a mensagem de erro e o corrigiu.

Os cenários apresentados podem ser indicadores de que os estudantes não compreenderam a mensagem de erro. Logo, são situações suscetíveis ao reaparecimento do erro. A fim de exemplificar as limitações de RED quanto ao reparecimento do erro, deixamos um questionamento. Com base na Tabela 5, podemos afirmar que o estudante D possui a mesma dificuldade que o estudante E? Do ponto de vista do reaparecimento do erro, podemos notar que o estudante E teve maior dificuldade de compreensão da mensagem melhorada pois foi reincidente no erro, mesmo após ter acesso à mensagem melhorada no primeiro erro.

Por outro lado, do ponto de vista da métrica RED, os estudantes D e E são iguais (ver Tabela 5), ambos com escore zero. O fato de RED não considerar este aspecto no processo de aprendizagem pode influenciar tanto na avaliação da eficácia de mensagens melhoradas como na utilização desta métrica como preditor, por não mapear esta dificuldade do estudante. Estes fatores motivaram a criação de uma nova métrica que visa minimizar esses problemas.

4.2 Uma nova métrica de avaliação

Neste sentido, propomos uma nova métrica que, além dos erros repetidos, vise mapear os erros recorrentes, aqueles que voltam a aparecer após um tempo de modo não consecutivo. Para isto, somamos à métrica RED o valor computado para os erros recorrentes, referenciados aqui como (*ErrRec*). A Tabela 4 apresenta um detalhamento das siglas utilizadas nesta subseção.

Tabela 4: Descrição das siglas utilizadas

Sigla	Descrição
S	Sequência de erros obtidos pelos estudantes.
Q	Sequência de erros recorrentes obtidos pelos estudantes. É derivada de S.
Q_i	Subsequências presentes na sequência Q. As subsequência são divididas pelos intervalos sem erros recorrentes (...).
y	Número de erros recorrentes em Q.
y_i	Número de erros recorrentes em Q_i.

Para o cálculo do *ErrRec*, obtemos os erros dos estudantes e geramos uma sequência de erros (S). Nesta sequência, identificamos os erros recorrentes, gerando uma nova sequência (Q), considerando os erros consecutivos como uma única ocorrência e excluindo da sequência a primeira ocorrência do erro. Nesta sequência, podemos identificar subsequências (Q_i) que são a presença dos erros recorrentes de um determinado tipo.

Para exemplificar como ocorrem as subsequências Q_i, considere uma sequência Q com erros do tipo x e z dada por "x ... z ... x ...". Temos Q_1 igual a "x", Q_2 igual a "z" e Q_3 igual a "x". A presença de três pontos representa execuções sem erro no intervalo entre as execuções que geraram erros. Para cada Q_i contabilizamos o número de erros, este número é o y_i da sequência. Logo, y_1, y_2 e y_3 são iguais ao valor 1. A seguir, apresentaremos exemplos apenas com um tipo de erro (x) de modo a tornar a descrição mais compreensível.

$$REC = \sum_{i=1}^{n} \frac{r_i^2}{r_i + 1} + \sum_{j=1}^{m} \frac{y_j^2}{y_j + 1} \quad (1)$$

O valor de *ErrRec* para uma sequência Q com n erros recorrentes e não consecutivos é a soma de $y_i^2/(y_i + 1)$ para cada sequência Q_i. A Equação 1 apresenta a métrica proposta. A primeira parcela é referente à métrica RED, já a segunda parcela é a nossa contribuição para erros recorrentes, *ErrRec*. A soma das duas parcelas constitui a métrica REC.

Tomando como base a Tabela 5, percebe-se que o estudante E obteve o erro do tipo x, solucionou este erro e tornou a obter o mesmo erro em seguida. A sequência S que representa esta sequência é "... x ... x ...". Já a sequência Q que representa este comportamento é "... x ...". O estudante F, na Tabela 6, possui uma sequência S dada por "x ... xx ... xx". Para obtermos a sequência Q devemos considerar a sequência de erros consecutivos como uma única ocorrência e ignorar a primeira ocorrência do erro. Logo a sequência Q é "... x ... x ...". Para esta sequência Q, temos duas subsequências de erro (Q_1 e Q_2), logo $y = 2$.

Sabendo que Q_1 = "x" e Q_2 = "x", teríamos $y_1 = 1$ e $y_2 = 1$. Logo, dada a Equação 1 teríamos $\frac{1^2}{2}$ para y_1 e $\frac{1^2}{2}$ para y_2. Este padrão se repete para todos os casos de erros recorrentes pela própria natureza da métrica ao ignorar os erros consecutivos e criar subsequências com apenas um erro.

Desse modo, como não há variações dos números nas ocorrências em y_i , tal como ocorre em RED para x_i, teremos sempre um somatório de $1/2$ para cada ocorrência. Logo, considerando y o número de erros em Q, a fórmula inicial da métrica REC pode ser simplificada, conforme pode ser observado na Equação 2:

$$REC = \sum_{i=1}^{n} \frac{r_i^2}{r_i + 1} + y \cdot 0.5 \quad (2)$$

Como estamos interessados no processo de aprendizagem de programação e não na solução local, analisamos o reaparecimento do erro de modo geral. Isso significa que o estudante pode ter obtido um erro na atividade 1 e outro erro similar na atividade 10, o erro da atividade 10 será considerado como uma recorrência do erro. A Tabela 5 é uma modificação da Tabela 2 de Becker [3] que mostra a sequência S para RED e a sequência Q considerada por REC, além dos seus respectivos valores.

A Tabela 5 evidencia a diferença entre RED e REC. As sequências 2 e 3 possuem o mesmo REC e isto é compreensível, tendo em vista que o estudante E pode ter resolvido o erro ao acaso ou gerado um erro diferente na tentativa de solucionar o problema. Isto não o faz diferente do estudante da sequência 3, pois ambos não compreenderam a primeira mensagem de erro.

A Tabela 6 apresenta algumas situações em que RED não ajuda a distinguir entre estudantes com perfis de comportamento distintos (1 e 2 ou 3 e 4).

Respondendo à questão colocada anteriormente, REC consegue fazer distinção entre os estudantes D e E, que já tiveram acesso à informação por meio de mensagens melhoradas (Tabela 5). Além disso, consegue mapear a recorrência do erro para todos os casos de problema apresentados (i – vi). Vale observar que os valores de REC para as sequências 4 e 5 são iguais na Tabela 6, pois a sequência de erros é a mesma, diferindo apenas em sua ordem.

Tabela 5: Valores de REC e RED para sequências de erro

no.	sequência S (RED)	*r*	sequência Q (REC)	*y*	RED	REC
1(D)	x	0	...	0	0	0
2 (E)	... x ... x ...	0	... x ...	1	0	0,5
3	... xx ...	1	...	0	0,5	0,5
4	... xx .. xx ...	2	... x ...	1	1	1,5
6	... xx ... xx ... xx ...	3	... x ... x ...	2	1,5	2,5
7	... xxx... xx ...	3	... x ...	1	1,83	2,33
9	... xx ... xx ... xx ... xx ...	4	... x ... x ... x ...	3	2	3,5
10	... xx ... xx ... xxx ...	4	... x ... x ...	2	2,3	2,3
11	... xxx ... xxx ...	4	... x ...	1	2,6	3,1
12	... xx ... xxxx ...	4	... x ...	1	2,75	3,25

Tabela 6: Comparativo entre REC e RED

no.	sequência S (RED)	*r*	sequência Q (REC)	*y*	RED	REC
1	xx	1	...		0,5	0,5
2	x ... xx	1	... x ...	1	0,5	1
3	... xx ... xx ...	2	... x ...	1	1	1,5
4	... xx ... x ... xx	2	... x ... x ...	2	1	2
5 (F)	x ... xx ... xx	2	... x ... x ...	2	1	2
6	x ... xx ... xx ... xx	3	... x ... x ... x	3	1,5	3
7	x ... xx ... xx ... xx ... xx	4	... x ... x ... x ... x	4	2	4

É importante pontuar que embora REC consiga mapear situações que RED não consegue, ela herda as propriedades de RED enunciadas por Becker [3]. As sequência 9, 11 e 12 na Tabela 5 assim como as sequências 5, 6 e 7 na Tabela 6 exemplificam algumas das propriedades a seguir.

(1) Mensura erros repetidos e recorrentes de forma específica e exclusiva.
(2) Não depende de parâmetros que devem ser combinados com o contexto e assuntos em debate, ao contrário de EQ.
(3) Contabiliza a quantidade de erros repetidos e recorrentes em uma sequência.
(4) Leva em conta os comprimentos da sequência de erros repetidos e recorrentes em uma sequência.
(5) Leva em conta o tamanho da sequência de erros recorrentes em uma sequência.
(6) Tem um valor de zero para sequências com zero erros repetidos e recorrentes.
(7) É aditivo: a sequência 6 é a adição da sequência 2 com a sequência 5 (Tabela 6).
(8) É proporcional: valores de sequências relacionados são proporcionais, conforme as sequências 5 e 7 (Tabela 6).

(9) Atribui maiores penalidades para maior densidade de erros repetidos e recorrentes (por exemplo, as sequências 9, 11 e 12 na Tabela 5).
(10) REC ∈ R ≥ 0, diferentemente de EQ e Watwin, ambos ∈ [0, 1].

Becker et al. [4] apontam a necessidade da formalização de uma métrica que avalie a "legibilidade" da mensagem melhorada. Embora muitos autores tenham definições variadas sobre a "legibilidade", Flowers e colaboradores [7] afirmam que "*menções que mensagens legíveis são mais memoráveis*" ao passo que Traver [21] afirma que"*escrever mensagens de erro mais legíveis promove a correção de erros pelo reconhecimento em vez de pela mera lembrança*". Ou seja, avaliar a "legibilidade" é também avaliar o impacto das mensagens sobre a aprendizagem.

É importante salientar que este aspecto se aproxima de heurísticas propostas para a avaliação de usabilidade de sistemas. Dentre os nove princípios básicos de usabilidade propostos por Nielsen and Molich [17], podemos destacar a promoção de feedback e a redução do uso da memória. Para Molich and Nielsen [16], um sistema deve fornecer feedback apropriado e em tempo hábil ao usuário. Além disso, eles ratificam que a memória de curto prazo dos usuários é limitada, logo, os usuários não deveriam lembrar de informações, mas reconhecê-las facilmente sempre que apropriado.

Um estudante que compreendeu o erro dificilmente irá cometer o mesmo erro ao longo da sua atividade. É neste sentido que REC considera este fenômeno para computar o escore dos estudantes. Assim, REC utiliza as propriedades adequadas do RED e adiciona informações de modo a minorar as limitações apontadas[2].

5 RESULTADOS

Nesta seção, apresentamos os resultados divididos em cinco seções: projetos e avaliação; erros mais cometidos pelos estudantes; os erros mais recorrentes; adequação das métricas e, por fim, um comparativo entre RED e REC.

5.1 Projetos e avaliação

Conforme mencionado, os estudantes do curso de Python tiveram acesso a 40 atividades sobre conceitos vistos no curso. Para cada atividade, o estudante criava um projeto e codificava a sua solução. Apresentamos os resultados sobre os projetos como uma forma de situar a nossa abordagem e fornecer bases para os resultados relacionados ao objetivo principal deste trabalho.

Como resultados, apresentamos que 392 projetos foram criados pelos estudantes, de um total de 480 projetos possíveis (81,6%). Deste número, 341 foram entregues, ou seja, marcados como finalizados na plataforma. Foram realizadas 2.185 execuções e 1.150 testes. Nós utilizamos os testes unitários para avaliar a adequação dos projetos entregues.

Adicionalmente, foi realizada uma verificação manual para avaliar se o estudante utilizou o conceito solicitado na questão. Por exemplo, em uma questão que solicita o uso de loops, o estudante pode ter repetido um trecho de código ou não ter utilizado funções para uma funcionalidade requerida na atividade.

[2]O código da métrica REC utilizado nesta estudo pode ser consultado em https://github.com/LuisAraujo/RECRecurrentErrorDensity

Tabela 7: Erros mais recorrentes

Ranking	Tipo do Erro	Ocorrências
1	MISSING_COMMA	74
2	TypeError	42
3	IndentationError	18
4	EXTRA_TOKEN	15
5	ValueError	13
6	MISSING_ASSIGNMENT_SOURCE	11
7	AttributeError	8
8	NO_VIABLE_ALTERNATIVE	6
9	SyntaxError	6
10	USE_SEMICOLON_INSTEAD_OF_COMMA	5

Para cada projeto, foi dada uma nota que variou entre zero, cinco e dez. A nota zero corresponde à atividade que não passou no último teste unitário executado ou ao fato de o estudante não ter utilizado o conceito solicitado. A nota cinco corresponde à atividade que passou no último teste, mas o estudante utilizou o conceito solicitado de modo parcial. A nota dez corresponde à atividade que passou no último teste e o uso correto do conceito solicitado. Para compor a nota dos estudantes, calculamos a média entre as notas do projeto e a nota na avaliação final. 75% dos estudantes tiveram média acima de seis.

5.2 Erros mais cometidos

Mapeamos os erros mais cometidos pelos participantes, no intuito de investigar até que ponto nosso curso está próximo às abordagens que serviram como base para a geração das mensagens melhoradas. A maioria dos erros reportados por Jesus et al. [12] e Pritchard [18] foram cometidos pelos participantes. Identificamos dois erros não reportados por Jesus et al.: "*TypeError: unsupported operand type - for +*" e "*SyntaxError: invalid character in identifier*"; e um erro não reportado por Pritchard: "*SyntaxError: Missing parentheses in call to*". Assim como no trabalho de Pritchard, "*SyntaxError: invalid syntax*" e "*NameError: name - is not defined*" são os dois erros com maiores ocorrências. O nosso erro com maior ocorrência aparece como o segundo maior em ocorrências no trabalho de Jesus et al. [12]. Esses fatores demonstram uma forte adequação na nossa abordagem em comparação a outros trabalhos.

5.3 Erros mais recorrentes

Apresentamos ainda, como resultado desta pesquisa, a lista de erros com maior recorrência em nossa abordagem. A Tabela 7 apresenta os erros com mais de uma recorrência. Isso significa que o erro pode ter aparecido mais vezes de modo consecutivo, o que não é considerado aqui. A fim de aprofundar o entendimento sobre o erro, nós utilizamos o *parser* proposto por Kohn [13] para categorizar os erros sintáticos dos estudantes.

Como é possível perceber, os estudantes constantemente esquecem o uso de vírgulas, apresentam problemas com indentação ou adicionam *tokens* não reconhecidos por Python como, por exemplo, símbolos matemáticos para realizar operações. Do pontos de vista

de erros de execução, os estudantes costumam ter problemas com tipos e valores.

5.4 Adequação das métricas

Como um dos resultados principais, apresentamos a adequação das métricas para a avaliação da nossa abordagem. A média do escore EQ para todos os projetos é 0,004. O desvio padrão é 0,02, a mediana é 0,0 e a variância 0,0003. A métrica EQ conseguiu mensurar apenas 37 projetos (9,4%). A dispersão dos dados mostra que não há uma variabilidade que permita distinguir entre perfis de estudantes, ratificando a baixa adequação de EQ para o nosso contexto. Isto se deve ao alto número de projetos com escore zero, dada as restrições desta métrica.

Do mesmo modo, aplicamos a métrica RED para os projetos dos estudantes. Foram mensurados 126 (32%) dos projetos. A média RED para os projetos é 4,61. A mediana é 1,50 e o desvio padrão é 7,61. A fim de comparar as métricas RED e REC, contabilizamos apenas os projetos que possuem um escore maior que zero para REC, o que se justifica pois RED está contido em REC.

Nós não aplicamos o algoritmo *Watwin* tendo em vista a sua limitação quanto a sessões. Assim, consideramos o percentual de 9,4% baixo para a realização do estudo. Este percentual foi comprovado pela métrica EQ.

5.5 Comparativo entre RED e REC

Outro resultado principal é apresentado a seguir. Avaliamos a métrica REC apresentada na Seção 4, de modo similar ao realizado com as outras duas métricas. Foram mensurados 170 (43,3%) dos projetos através de REC. A média de REC é 6,07. A mediana é 2,75 e o desvio padrão é 8,98. A Figura 2 demonstra a diferença entre RED e REC quanto à quantidade de projetos com escore maior que zero. Levando em conta a quantidade de projetos que obtiveram escore positivo, percebe-se que REC levou a um aumento de 30,9% em relação a RED. A Tabela 8 sumariza esses resultados. O valor mínimo atribuído à métrica RED é zero e à REC, é 0,5. Já os valores máximos são, respectivamente, 66,8 e 79,8.

Tabela 8: Comparação entre as métricas EQ, RED e REC

	Média	***Mediana***	**Desvio Padrão**
EQ	0,004	0,000	0,0225
RED	4,613	1,500	7,612
REC	6,079	2,750	8,983

A mudança proposta em REC impacta na dispersão dos dados, gerando maior variabilidade de perfis de comportamento. A Figura 3 apresenta o gráfico boxplot dos escores atribuídos às atividades de cada estudante (E1 a E12) em que é possível notar a dispersão dos dados de modo comparativo. Ao analisar as médias dos escores de todos os projetos dos estudantes, é possível ratificar a diferença na dispersão dos dados. A Média REC possui um valor de 5,16 e a Média RED, 3,64, enquanto que o desvio padrão, respectivamente, é de 4,5 e 3,7.

Um aspecto interessante que mostra a distinção na mensuração dos perfis é a diferença entre as médias REC e RED dos estudantes. Nós calculamos a diferença entre as médias RED e REC de

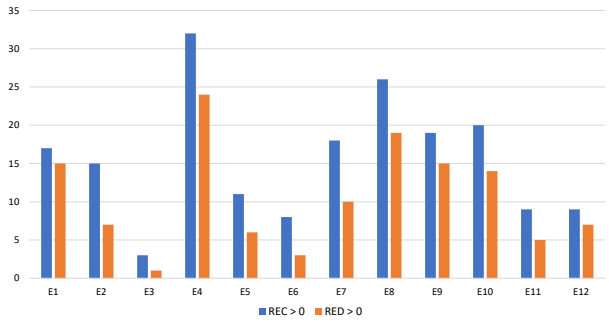

Figura 2: Quantidade de projetos por estudante avaliados por REC e RED

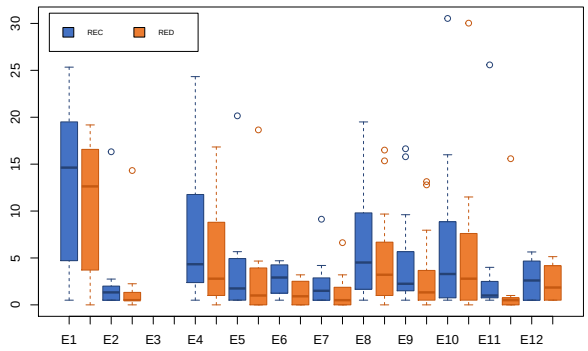

Figura 3: Boxplot dos escores REC e RED dos estudantes

cada estudante. Percebe-se que a diferença entre as médias não é constante, sendo média igual a 1,55 e desvio padrão de 0,9. Logo, REC permite não apenas uma ampliação dos valores RED, e sim adicionar informações sobre os erros dos estudantes.

Vale ressaltar que a variabilidade presente em REC poderá ajudar em um melhor resultado para previsões e avaliação de desempenho, bem como a avaliação da eficácia das mensagens melhoradas levando em consideração a sua reincidência não consecutiva. A principal contribuição desta métrica é a identificação de situações que demonstram dificuldades enfrentadas pelos estudantes que não eram consideradas até então por outras métricas.

6 DISCUSSÃO

Nesta seção apresentaremos as discussões baseadas nos seis situações que dificultam a avaliação local do erro e, portanto, subsidiam a proposta da métrica REC. Como foi possível observar nos resultados, a maioria dos estudantes obteve nota superior a seis (numa escala de 0 a 10), fato que demonstra a adequação do nível de instrução e avaliação para o público. Além disso, os erros cometidos pelos estudantes são similares aos apresentados por outros trabalhos, fato que demonstra a proximidade da nossa abordagem com outras já

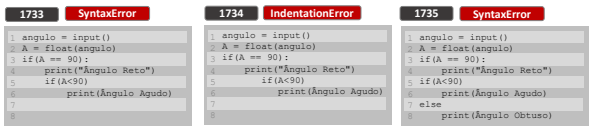

Figura 4: Execuções 1733,1734 e 1735 de E1

publicadas. Finalmente, percebe-se que o fenômeno da recorrência do erro é real e que ocorre em uma escala considerável. Devido à convergência da nossa abordagem com outros trabalhos, temos indícios de que este fenômeno ocorra em outros contextos.

Tendo em vista as proposições elucidadas anteriormente, percebe-se que a proposição i) "pesquisa material/internet" e ii) "ter sido ajudado por colega" são difíceis de avaliar em vista dos dados obtidos para as compilações. Ainda assim, apontamos alguns fatores para a análise destas preposições. O tempo elevado entre pares de compilações pode sugerir que os estudantes trabalham mais tempo na correção ou devotam um tempo adicional para a pesquisa da solução. Além disso, o *diff* pode ser um termômetro para verificar se o estudante fez uma grande modificação após o tempo gasto entre soluções. Logo, um tempo longo e um *diff* pequeno pode indicar que o estudante fez alguma pesquisa ou recebeu algum auxílio. Já o tempo curto e um *diff* grande pode indicar cópia de solução de algum colega ou de algum material.

Outro fator que reforça a ideia de que os estudantes pesquisam em materiais ou obtêm ajuda de colegas foi demonstrado por meio do questionário pós-intervenção. 75% dos estudantes pesquisaram no Google, Stack Overflow ou similares, sendo que 25% dos estudantes utilizaram a pesquisa em cerca de 40 a 100% dos projetos. Estas pesquisas e suportes resumem-se a entender o funcionamento de alguma função, resolver problemas lógicos, aprender o funcionamento de algo não abordado no curso ou resolver algum problema sintático.

Quanto ao item iii) "sobre ajuda do professor/tutor", diferentemente de Kohn [13], Becker et al. [5] e outros, nossos dados são provenientes do curso introdutório criado pelos pesquisadores. Assim, temos certeza que este fato não ocorreu. Nos encontros síncronos, o professor tirou dúvidas gerais, sem oferecer suporte individual para erros obtidos nos projetos.

Para os itens iv) "ter solucionado o erro ao acaso" é possível observar as correções que a princípio parecem válidas, mas que podem ser incluídas no item iv caso o estudante se depare com situações similares e não consiga corrigir o seu código. Analisar esse tipo de situação requer uma análise mais profunda. No entanto, encontramos alguns indícios dessa situação em nossa abordagem.

Sobre os itens v) "adicionado um outro erro" e vi) "deletado linha com erro" podemos fazer uma análise sobre as execuções dos estudantes. A fim de aprofundar a discussão sobre estes fenômenos, buscamos trazer alguns exemplos de execuções dos estudantes (*snapshot*).

A Figura 4 apresenta um exemplo da situação v). Nela, é possível observar uma sequência de execuções 1733,1734 e 1735 do Estudante E1. Na primeira execução, o estudante obteve um erro sintático pelo fato de não ter usado o símbolo de : na estrutura if. A modificação

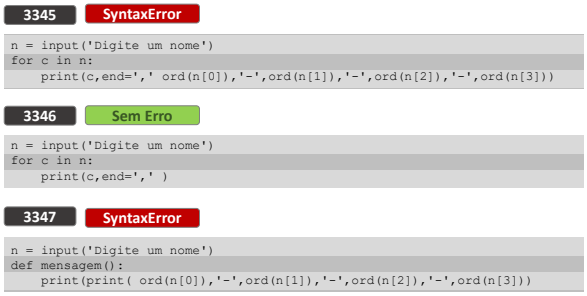

Figura 5: Execuções 3345, 3346 3347 de E4

realizada foi a indentação desta linha com erro, o que ocasionou em um segundo erro de indentação. A execução seguinte desfaz toda a indentação retornando ao erro sintático anterior.

Um exemplo da situação vi) pode ser visto na Figura 5. A figura apresenta a sequência 3345 até 3347 de execuções. Nesta sequência, é possível observar que o estudante E4 comete um erro sintático por não ter adicionado vírgula após o atributo end. Ao se deparar com um erro na execução 3345, o estudante deleta o trecho com erro, obtendo uma execução sem erro em 3346. Este comportamento foi relatado por Kohn [13] em sua análise manual. O mesmo tipo de erro retorna na execução 3347 e, embora não pela mesma causa, para Python ambos são erros de sintaxe. Esta sequência apresenta ainda uma possibilidade de ajuda ou pesquisa, pois o estudante adiciona uma função 4 minutos e 27 segundos após a execução anterior (maior intervalo entre compilações neste projeto).

Com base nos dados apresentados, observamos que a métrica REC possibilita o mapeamento de situações que demonstram dificuldade dos estudantes e que não são reconhecidas por RED. Este fenômeno de reaparecimento não é exclusivo de um tipo de erro, como pode ser observado na lista de erros recorrentes na Tabela 7. Esta lista é inédita na área de métrica de erros e pode ser um ponto de partida para outros trabalhos ou auxiliar em abordagens pedagógicas.

Ratificamos aqui as possibilidades de adoção dessa métrica. Por ser uma métrica focada no comportamento do estudante perante erros, REC pode: avaliar a eficácia de mensagens melhoradas, ser computada para grupos com e sem suporte adicional [5]; promover formas de predição de desempenho ao ser correlacionada com notas de exames [19] e, assim, possibilitar o acompanhamento do estudante. O processo para a geração da métrica REC pode fornecer informações gerais sobre a turma como, por exemplo, quais os erros mais cometidos pelos estudantes. Esta informação pode guiar o planejamento do professor, promovendo aulas de revisão.

6.1 Limitações

Dada a amostra limitada, não foi possível realizar testes estatísticos como regressão linear para demonstrar a potência da métrica RED como preditor. Além disso, não foi possível realizar um estudo controlado para avaliar se há diferença estatística entre um grupo que tem acesso à ECEM e outro que apenas acessa a mensagem

original. Este teste é importante para mensurar a adequação do REC enquanto uma métrica que avalia a eficácia da mensagem melhorada. Entretanto, este trabalho teve um objetivo exploratório, o que levou à própria ideia da métrica REC.

7 CONSIDERAÇÕES FINAIS

Este trabalho visou, por meio de um estudo de caso exploratório, avaliar a adequação de métricas de erro para um curso introdutório com linguagem Python. Como resultado, percebe-se a baixa adequação das métricas EQ e *Watwin* e uma adequação aceitável para a métrica RED. No entanto, verificamos que a métrica RED não considera fatores relacionados ao reaparecimento de erro. Com isso, propomos uma nova métrica para mapear este fenômeno (REC). A diferença entre a dispersão dos dados das métricas RED e REC demonstram que o fenômeno do reaparecimento de erros é algo que ocorre em nossa abordagem. A métrica REC proposta consegue mapear 34% a mais de projetos se comparada com a métrica RED, gerando uma maior variabilidade dos dados e mapeando situações não consideradas anteriormente.

REC herda diversas características da RED e permanece menos suscetível a contexto. Assim, se apresenta como uma uma opção viável para avaliação da eficácia de mensagens melhoradas, em especial em abordagens introdutórias com atividades que demandam um tempo menor de trabalho dos estudantes como o nosso curso de Python.

7.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se replicar o curso introdutório de Python para um número maior de estudantes. A partir disso realizar um experimento controlado com dois grupos de estudantes para avaliar o uso de mensagens melhoradas. Pretende-se adotar o *parser* proposto por Khon em face da comprovação da limitação do Python quanto a mensagens de erro. Os dados provenientes desse estudo serão analisados por meio de testes estatísticos para validar a adequação do REC na avaliação das mensagens melhoradas em comparação com outras métricas. Além disso, objetiva-se realizar testes para avaliação de REC como preditor de desempenho.

REFERÊNCIAS

[1] Luis Gustavo Jesus Araujo, Roberto Almeida Bittencourt, and Christina von Flach Garcia 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.

[2] Luis Gustavo J Araujo, Roberto A Bittencourt, and David MB Santos. 2018. Contextualized spiral learning of computer programming in brazilian vocational secondary education. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE.

[3] 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*. 296–301.

[4] Brett A Becker, Paul Denny, James Prather, Raymond Pettit, Robert Nix, and Catherine Mooney. 2021. Towards Assessing the Readability of Programming Error Messages. In *Australasian Computing Education Conference*. 181–188.

[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] Paul Denny, James Prather, Brett A Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B Powell. 2021. On Designing Programming Error Messages for Novices: Readability and its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[7] Thomas Flowers, Curtis A Carver, and James Jackson. 2004. Empowering students and building confidence in novice programmers through Gauntlet. In *34th Annual Frontiers in Education, 2004. FIE 2004.* IEEE, T3H–10.

[8] Linda Grandell, Mia Peltomäki, Ralph-Johan Back, and Tapio Salakoski. 2006. Why complicate things? Introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. 71–80.

[9] Mark Guzdial. 2003. A media computation course for non-majors. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*. 104–108.

[10] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin* 35, 1, 153–156.

[11] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*. 73–84.

[12] 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, 1751.

[13] 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*. 524–530.

[14] 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*. 1054–1060.

[15] Linda Mannila, Mia Peltomäki, and Tapio Salakoski. 2006. What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education* 16, 3, 211–227.

[16] Rolf Molich and Jakob Nielsen. 1990. Improving a human-computer dialogue. *Commun. ACM* 33, 3, 338–348.

[17] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 249–256.

[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] Maria Mercedes T Rodrigo, Emily Tabanao, Ma Beatriz E Lahoz, and Matthew C Jadud. 2009. Analyzing online protocols to characterize novice java programmers. *Philippine Journal of Science* 138, 2, 177–190.

[20] Tom Schorsch. 1995. CAP: an automated self-assessment tool to check Pascal programs for syntax, logic and style errors. *ACM SIGCSE Bulletin* 27, 1, 168–172.

[21] Vicente Javier Traver Roig. 2010. On compiler error messages: what they say and what they mean.

[22] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *2013 IEEE 13th international conference on advanced learning technologies*. IEEE, 319–323.