EME: Uma ferramenta para auxiliar a correção de erros de aprendizes de programação

Galileu Santos de Jesus[1], Kleber Tarcísio Oliveira Santos[1], Jaine da Conceição Santos[1], Alberto Costa Neto[1].

[1]Departamento de Computação – Universidade Federal de Sergipe.
Av. Marechal Rondon, Jardim Rosa Elze, São Cristóvão - SE, 49100-000.

galilasmb@gmail.com, klebertarcisio@yahoo.com.br

jainecs@dcomp.ufs.br, alberto@ufs.br

Abstract. *This paper presents a proposal to support teaching-learning of computer programming, improving the online judge The Huxley by including feedback messages that are easily understood by the learners of initial programming courses and guiding them through the syntax errors presented when performing submissions to an online judge. An analysis was performed in the database to catalog the programming syntactic errors of each language available in this tool. The Python language was the one with more wrong submissions, becoming our target for analysis. When analyzing the errors of this language, the results indicate that there are 143 types of syntactic errors, distributed in 20 classes. Finally, it was accomplished the requirements elicitation, the implementation and integration with the online judge tool. The study concluded that friendly messages are easier to understand than the original messages, and that they were more useful in correcting syntactic errors.*

Resumo. *Este trabalho apresenta uma proposta para apoiar o ensino-aprendizagem de programação de computadores, aprimorando o juiz on-line The Huxley através da capacidade de produzir mensagens de feedback que sejam facilmente compreendidas pelos aprendizes de disciplinas iniciais de programação, norteando-os sobre os erros de sintaxe apresentados ao realizar uma submissão ao juiz on-line. Foi feita uma análise na base de dados para catalogar os erros sintáticos de programação de cada linguagem disponível desta ferramenta. A linguagem Python foi a que mais apresentou submissões erradas, tornando-se nosso alvo de análise. Ao analisar os erros desta linguagem, os resultados apontam que existem 143 tipos de erros sintáticos, distribuídos em 20 classes. Por fim, foi realizada a elucidação dos requisitos, a implementação e integração com o juiz on-line. O estudo concluiu que as mensagens amigáveis são mais fáceis de serem entendidas do que as originais e também que foram mais úteis na correção dos erros sintáticos.*

1. Introdução

Para estudantes iniciantes em programação, mensagens de erros podem ser muitas vezes fonte de medo e frustração, visto que é uma mensagem de difícil compreensão, frequentemente coibindo o aprendiz em progredir na aprendizagem em sala de aula. Além disso,

para entender a mensagem é requerido um conhecimento mínimo não só da língua inglesa mas também de erros referentes ao processo de compilação ou interpretação.

De acordo com Weber, Brusilovsky e Steinle (2014), aprender a programar exige diversas habilidades, principalmente em resolver problemas, aprender uma sintaxe, lógica de programação e utilização de ferramentas. De acordo com o trabalho de Miyadera, Huang e Yokoyama (2000) uma das maiores dificuldades de um estudante de programação é entender cada passo da execução do programa e consertar erros de sintaxe em um programa.

Para exemplificar esta problemática, a Figura 1 ilustra uma mensagem de *feedback* fornecida ao aluno, ao realizar uma submissão utilizando a linguagem Python, no sistema de juiz *on-line* The Huxley. A transcrição da saída produzida pelo interpretador da linguagem Python tenta reportar que o aluno tentou utilizar um identificador que não foi definido anteriormente, por meio da mensagem: "*NameError: name 'P' is not defined*" - linha 4 e linha 25, sinalizando que o nome *P* não foi definido. Porém, como fator complicador, foram exibidos diversos erros pertencentes ao sistema de exceção, encontrados nas linhas 5 a 19, não trazendo nenhuma informação útil que possa sanar o problema.

```
Traceback (most recent call last):
File "/Aprovados.py", line 3, in <module>
while P>0:
NameError: name 'P' is not defined
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Aprovados.py", line 3, in <module>
while P>0:
NameError: name 'P' is not defined
```

Ver saída amigavel

Figura 1. Exemplo de erro de sintaxe fornecido pelo The Huxley para uma submissão realizada na linguagem Python.

Foram encontrados na literatura alguns trabalhos que tratam do tema deste artigo, dentre eles, destacam-se os seguintes.

O Code Analyzer for Pascal (CAP) (SCHORSCH, 1995), foi desenvolvido para proporcionar aos alunos um *feedback* amigável e automatizado sobre erros comuns de sintaxe, lógica e de estilo dos programas Pascal. Tais erros foram recolhidos e classificados através de um estudo informal de todos os 520 alunos matriculados no curso de Introdução

à Ciência da Computação - CS110. Após executar uma análise estática do código fonte, informa ao estudante o que está errado, porque está errado e como corrigir o problema específico, podendo relatar 111 mensagens de diagnóstico diferentes (incluindo 62 para erros de sintaxe, 21 para erros de lógica e 28 para erros de estilo).

A ferramenta Expresso (HRISTOVA et al., 2003), gera melhores mensagens de erros do que as existentes nos compiladores e também fornece sugestões sobre como corrigir o código-fonte. Foi elaborada uma lista com 62 erros de programação da linguagem Java, dos quais 20 foram identificados como essenciais. Esta seleção foi feita baseando-se nos relatos de tutores que ensinam Java em universidades e de professores de ciência da computação de 58 escolas. Os erros foram classificados em três categorias: erros de sintaxe, erros de semântica e erros lógicos.

O trabalho de Marceau, Fisler e Krishnamurthi (2011) buscou analisar a efetividade das mensagens de erros exibidas pela ferramenta DrScheme. Tenta resolver o problema de que linguagens de programação são projetadas para especialistas, e não para o ensino de programação. Esta ferramenta oferece diversos níveis de linguagem, onde cada nível é um subconjunto do próximo nível. As mensagens apresentadas são referentes ao nível atual dos conceitos aprendidos pelos alunos até então, podendo progredir até cinco níveis, iniciando em Beginner Student Language (BSL) até Advanced Student Language (ASL).

O trabalho de Pelz (2014) fez diversas alterações no sistema do Portugol Stúdio[1] com o objetivo de aprimorar o processo de correção automática de problemas. Uma das alterações efetuadas procurava identificar construções obrigatórias ou proibidas no código-fonte. A outra alteração foi na exibição de erros semânticos. Não há especificação de quais mensagens de erros semânticos foram melhoradas. Além disso, o trabalho não englobou exercícios contendo laços de repetição e manipulação de vetores e matrizes.

Este artigo tem como contribuição principal apresentar uma ferramenta intitulada de EME - *Explain My Error*, que proporcione melhorias no processo de ensino e aprendizagem, provendo mensagens de erros de sintaxe que são facilmente compreendidas no contexto de aprendizes de programação de computadores em disciplinas introdutórias. A abordagem contribui com a identificação da causa do erro, para reduzir sua frustração inicial através de uma experiência construtivista de educação em um ambiente virtual de aprendizagem, que tem um juiz *on-line* como apoio à aprendizagem de programação. A Figura 2 ilustra a exibição e transcrição da mensagem amigável quando aplicada à mensagem ilustrada na Figura 1.

Considerando a necessidade da utilização de ambientes virtuais de aprendizagem que possuam *feedback* de erros sintáticos mais entendíveis por seus usuários, com a finalidade de avaliar a utilização desta abordagem como apoio ao ensino e aprendizado de programação, tem-se como hipótese: A utilização de mensagens de erros sintáticos mais entendíveis, guiará melhor os aprendizes de programação na correção de erros, impulsionando o ensino-aprendizagem através da redução das fontes de medo e frustrações iniciais.

[1] lite.acad.univali.br/portugol

```
Erro na linha 3
No trecho de código:
while P>0:

Descrição: Foi feito o uso de uma variável que não foi definida ou um comando que foi escrito de forma
errada. Verifique a palavra 'P'.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 3
No trecho de código:
while P>0:

Descrição: Foi feito o uso de uma variável que não foi definida ou um comando que foi escrito de forma
errada. Verifique a palavra 'P'.
```

Ver saída original

Figura 2. Exemplo de mensagem exibida com a abordagem desenvolvida ao aplicar na mensagem ilustrada na Figura 1.

2. Desenvolvimento da ferramenta

Neste trabalho foi utilizado o juiz *on-line* The Huxley pelo fato de que foi disponibilizada uma API para conexão direta com a base de dados atual, permitindo acesso a diversas funções disponíveis no sistema, possibilitando assim a extração de diversos dados acerca das submissões de programas de aprendizes para realização de uma análise estatística, assim como a realização da integração da ferramenta desenvolvida.

2.1. Requisitos e Mensagens

No momento da extração dos dados, em Janeiro de 2017, foram encontradas exatamente 667.836 submissões, sendo distribuídas dentre todas as linguagens disponíveis para submissão e os tipos de retorno da ferramenta. A relação de acertos e erros da linguagem de programação Python é de aproximadamente 1/5, enquanto que em C e C++ é de cerca 1/3, motivando-nos a escolher Python como primeira linguagem de estudo.

A Figura 3 ilustra a notação da mensagem de erro mapeada, onde, inicialmente é identificada a linha que a contém, posteriormente sua classe e por fim, o subtipo do erro com sua descrição. Esta notação é específica da linguagem Python.

Foram encontrados 143 subtipos de erros em Python, agrupados em 20 classes, ao analisá-los e catalogá-los, foi feita a elucidação dos requisitos funcionais, através da análise da base de dados, assim como pesquisas do significado, disponível em[2]. Para cada subtipo de erro, que provém de um tipo de mensagem de erro, foi associado um requisito funcional que visa explicar o subtipo do erro.

[2]gg.gg/requi

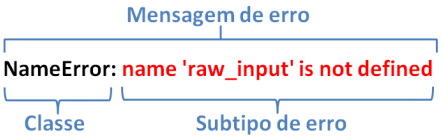

Figura 3. Mensagem de erro mapeada.

Em Engenharia de Software, um requisito funcional define uma função de um sistema de software ou seu componente. Os requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação de dados e de processamento e outras funcionalidades específicas que definem o que um sistema, idealmente, será capaz de realizar (PRESSMAN, 2016).

A Tabela 1 ilustra as 21 (vinte e uma) mensagens mais frequentes que, após o processo de análise, foram encontradas na base e distribuídas de acordo com a quantidade de problemas utilizados na experimentação da ferramenta, representando 87.63% do total de ocorrência dos erros.

Tabela 1: Mensagens de erros Originais e Amigáveis.

Original	**Amigável**
SyntaxError: invalid syntax	Verifique se está faltando algum parêntese, a ausência de “:” e declaração correta de alguns comandos, tais como: atribuição, if, for, while, input, print e outros.
NameError: name ’raw_input’ is not defined	Foi feito o uso de uma variável que não foi definida ou um comando que foi escrito de forma errada. Verifique a palavra ’raw_input’.
ValueError: invalid literal for int() with base 10: ‘3 5 2’	O comando ‘int()’ necessita de literais compatíveis com seu tipo. Verifique os tipos ou dados de leitura: ’3 5 2’.
EOFError: EOF when reading a line	EOF - end of file (final de arquivo) - ocorreu um erro de final de arquivo ao realizar a leitura dos dados. Verifique a declaração do input ou as entradas.
SyntaxError: Missing parentheses in call to ’print’	Verifique o uso do parêntese, a partir da versão 3.0 do Python seu uso é obrigatório no comando print.
IndexError: list index out of range	O índice da lista está fora do intervalo, verifique o índice de acesso e seu valor.
IndentationError: expected an indented block	Era esperado um bloco indentado. Por favor verifique a indentação. A indentação é uma característica peculiar na linguagem. Em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual. Fique atento para não errar na próxima!
IndentationError: unexpected indent	Indentação inesperada. Por favor verifique a indentação. A indentação é uma característica peculiar na linguagem. Em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual. Fique atento para não errar na próxima!

Continuação da Tabela 1	
Original	**Amigável**
AttributeError: str’ object has no attribute ‘itmes’	O objeto do tipo “str” não tem nenhum atributo com o nome “itmes”. Verifique se digitou corretamente o nome do atributo ou o objeto.
TypeError: unsupported operand type(s) for -: ‘list’ and ‘list’	A operação - não é suportada para os tipos ‘list’ e ‘list’. Verifique os tipos e suas operações.
IndentationError: unindent does not match any outer indentation level	A indentação não corresponde a nenhum nível de indentação externa, por favor verifique a indentação. Em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual. Fique atento para não errar na próxima!
ValueError: could not convert string to float: ‘100 170 100 95’	Não é possível converter o tipo string para float, verifique se os dados de leitura são do tipo float: ‘100 170 100 95’.
ZeroDivisionError: float division by zero	Não é possível realizar divisão por zero. Verifique se isso ocorre em seu código.
TypeError: int’ object is not iterable	O objeto do ’int’ não é iterável, é apenas uma literal. Verifique se não é necessário o uso de uma lista ou o comando range(). Para transformar números em lista, você pode utilizar o operador [ ], exemplo: lista = [n1, n2, n3].
TypeError: unorderable types: str() <= int()	Os tipos não são comparáveis, por isso não é possível realizar a comparação: str() e int(). Modifique-os para que sejam do mesmo tipo.
KeyError: ’12’ 12 1000 13 1 ’a’	A chave digitada não é válida. Verifique seu tipo ou seu uso.
SyntaxError: unexpected EOF while parsing	Foram utilizados caracteres que não seguem o padrão da linguagem, assim ocorreu o erro de final de arquivo inesperado. Verifique os caracteres utilizados.
TypeError: not all arguments converted during string formatting	Nem todos os argumentos foram convertidos durante a formatação. Verifique a formatação, os tipos das variáveis e argumentos.
TypeError: int() argument must be a string or a number, not ’list’	A função int() deve ter o argumento do tipo string ou um número, não outro tipo.
TypeError: Can’t convert ’int’ object to str implicitly	Não é possível converter o tipo ’int’ implicitamente para o tipo str, ou seja, por mais que seja compreensível sua conversão.
TypeError: int() argument must be a string, a bytes-like object or a number, not ’list’.	O argumento da função int() deve ser uma sequência de caracteres ou bytes - como um objeto ou um número, não ’list’.

2.2. Integração

O projeto segue a arquitetura de um *Web Service*, onde a classe *main* realiza sua execução bem como identificação da mensagem passada via requisição do cliente. Então, é feito o retorno ao cliente a mensagem com a respectiva dica, conforme ilustra a arquitetura da Figura 4.

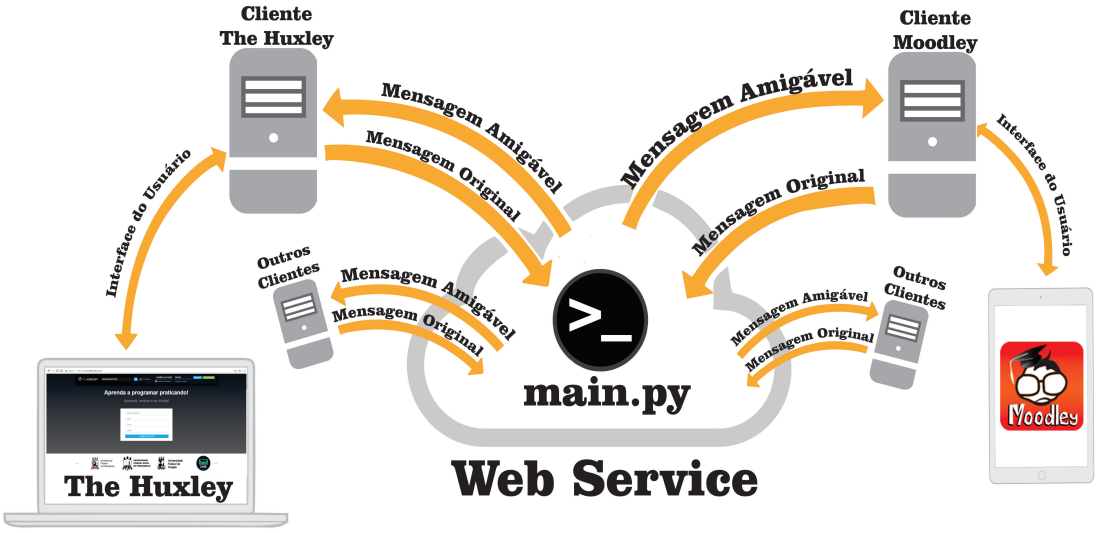

Figura 4. Arquitetura da aplicação integrada ao The Huxley e ao Moodley.

Com a criação do *web service* é possível que novas aplicações possam interagir com o projeto desenvolvido através de requisições. Por exemplo, já foi realizada a integração com outro projeto de pesquisa, que tem o objetivo de desenvolver uma ferramenta para dispositivos móveis que possibilite o acesso ao The Huxley e ao Moodle.

A integração com o The Huxley foi feita através de interface gráfica como ilustra a Figura 5, onde ao clicar no link *ver saída amigável* é feita a requisição ao *web service* que está sendo executado, passando os respectivos parâmetros e é obtida uma *string* como retorno contendo o resultado do processamento da requisição. Esta parte, devido à necessidade de conhecer os detalhes de implementação da interface Web do The Huxley, foi realizada pela própria equipe de desenvolvedores do The Huxley.

A Figura 5 ilustra a apresentação de mensagem de erro informada ao usuário ao executar um determinado código-fonte que possui erro sintático.

A Figura 6 ilustra a apresentação de mensagem de erro informada ao usuário ao executar um determinado código-fonte que possui erro sintático. Ao clicar no link *ver saída amigável*, é realizado processamento e requisição à ferramenta desenvolvida, apresentando o resultado na tela do usuário.

3. Avaliação

Como forma de avaliar a usabilidade e funcionalidade da ferramenta, foi realizada sua implantação em produção no The Huxley, sendo disponibilizada para todos os usuários

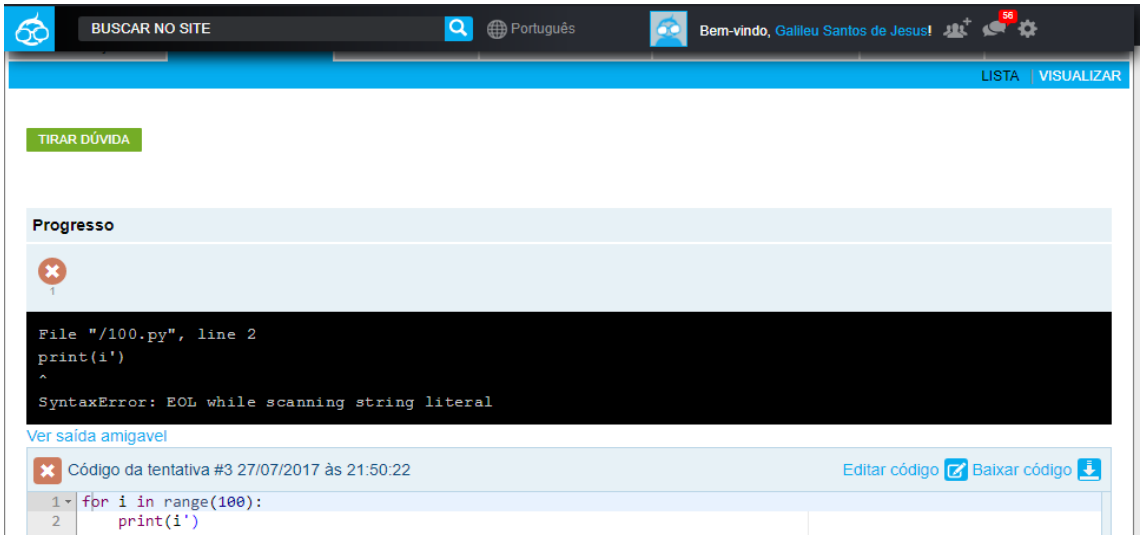

Figura 5. Mensagem de erro Original, apresentada pelo The Huxley.

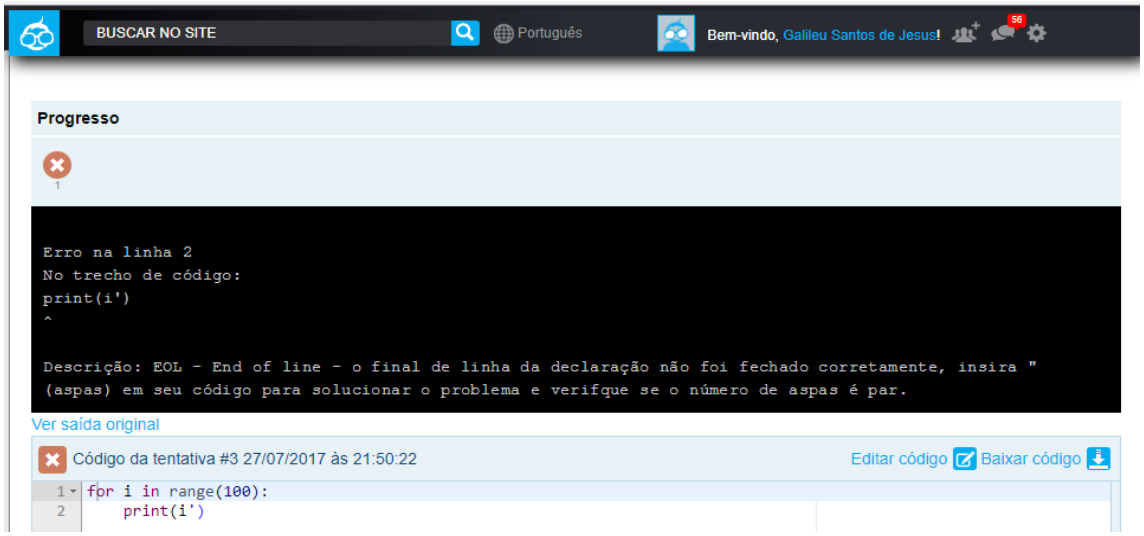

Figura 6. Mensagem de erro com saída amigável, apresentada pelo The Huxley após clicar em *ver saída amigável*.

do juiz *on-line*. Além disto, ao final das disciplinas, foi realizado um experimento com 26 (vinte e seis) participantes. O objetivo principal foi avaliar a abordagem original e a amigável, através da correção de 18 (dezoito) mensagens de erros, distribuídas em 6 (seis) problemas, onde cada um continha 3 (três) erros de codificação, representando os erros mais comuns ocorridos na base de dados do The Huxley. Foi disponibilizado um problema extra como forma de absorver os procedimentos a serem seguidos.

Foi definido um estudo experimental seguindo as orientações de Basili e Weiss (1984). Ao utilizar o modelo GQM - do inglês, Goal Question Metric, tem-se que o objetivo foi: **Analisar** a abordagem atual de apresentação de mensagens de erros sintáticos, fornecida pelo juiz *on-line* The Huxley, **com a finalidade de** avaliar, **com respeito à** eficiência e eficácia no auxílio à correção de erros sintáticos de programas, **do ponto de vista** de programadores iniciantes, **no contexto de** disciplinas iniciais de programação.

Ao final do experimento, foi aplicado um questionário para coletar dados visando responder questões de pesquisa. Com relação ao nível de dificuldade em entender as mensagens de erros originais e amigáveis, fica claro de acordo com a Figura 7 que a abordagem amigável tem uma concentração maior no nível menor de dificuldade quando comparada à abordagem original, ou seja, a abordagem amigável fornece mensagens mais simples de se entender na opinião dos participantes. Uma avaliação estatística das abordagens utilizadas na ferramenta está disponível no trabalho de Jesus et al. (2018).

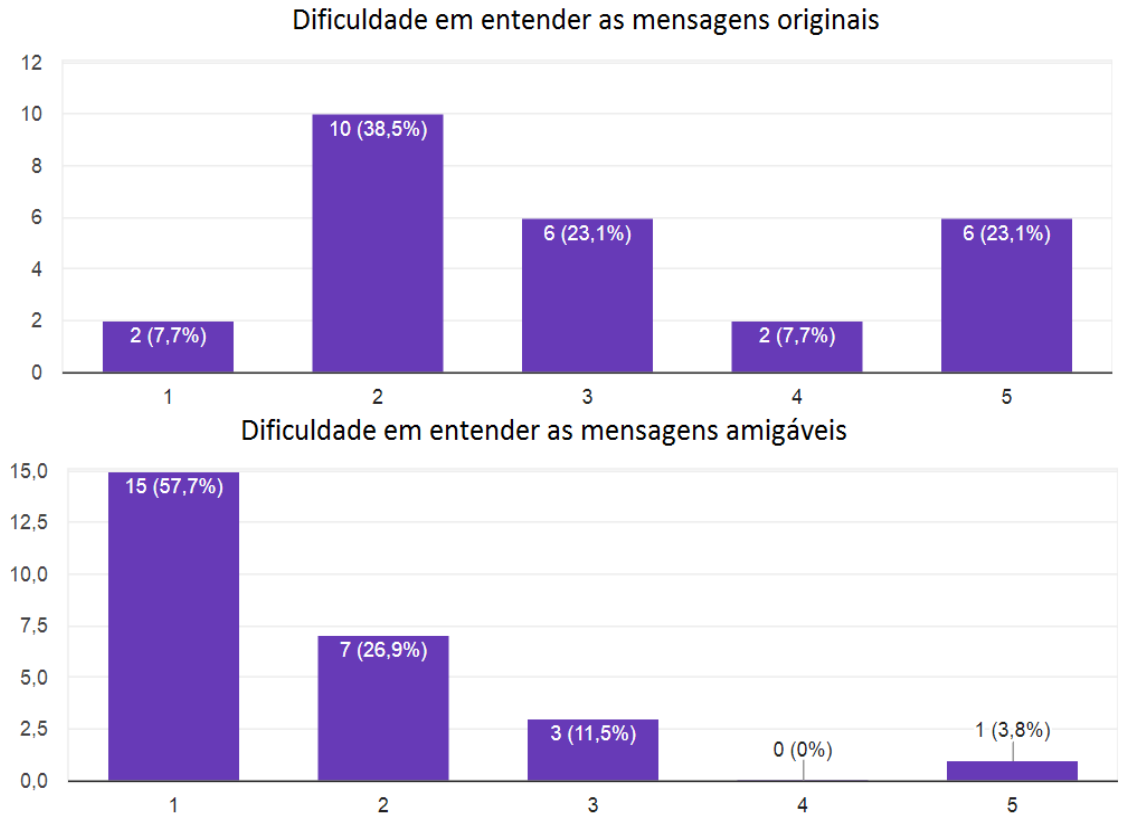

Figura 7. Níveis de dificuldade em entender as mensagens originais e amigáveis, respectivamente, do menor (1) ao maior (5).

4. Considerações Finais e Trabalhos Futuros

Diante dos resultados supracitados, norteados pela aplicação do questionário, há fortes indícios de que as mensagens amigáveis podem ajudar no entendimento dos erros sintáticos, guiando o aluno na correção ao apresentar dicas e sugestões de forma mais simples e compreensível. De acordo com as opiniões dos participantes do experimento, constatou-se que as mensagens geradas pela ferramenta EME apresentaram um menor nível de dificuldade para serem entendidas quando comparadas com as mensagens originais. Adicionalmente, demonstraram ser mais úteis, guiando melhor os aprendizes de programação na correção de erros.

No contexto de aprendizes de programação, a utilização das mensagens amigáveis pode diminuir o medo e frustrações, impulsionando portanto o ensino-aprendizagem. Porém, não descartamos o uso das mensagens originais.

Todos os participantes acharam importante que um juiz *on-line* possua mensagens amigáveis, informando que é de extrema importância para uma melhor compreensão do erros, pois mesmo sabendo inglês às vezes a mensagem não é tão clara.

Apesar de abranger todos os erros da base encontrada, não podemos afirmar que abrange os erros mais comuns da linguagem Python, porém abrange praticamente a totalidade dos erros dos usuários que a utilizam na ferramenta de juiz *on-line* supracitada.

Como trabalho futuro, faremos o mesmo nas outras linguagens de programação aceitas pelo The Huxley: C, C++, Java, Octave e Pascal. A linguagem C já está em fase de implantação no The Huxley, o que demonstra que a arquitetura escolhida é capaz de acomodar outras linguagens com facilidade. Além disso, pretende-se disponibilizar publicamente os Web Services para serem utilizados por outras ferramentas.

Referências

BASILI, V. R.; WEISS, D. M. A methodology for collecting valid software engineering data. *IEEE Transactions on software engineering*, IEEE, n. 6, p. 728–738, 1984.

HRISTOVA, M. et al. Identifying and correcting java programming errors for introductory computer science students. In: ACM. *ACM SIGCSE Bulletin*. [S.l.], 2003. v. 35, n. 1, p. 153–156.

JESUS, G. S. de et al. Avaliação de uma abordagem para auxiliar a correção de erros de aprendizes de programação. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2018. v. 29, n. 1, p. 1.

MARCEAU, G.; FISLER, K.; KRISHNAMURTHI, S. Measuring the effectiveness of error messages designed for novice programmers. In: ACM. *Proceedings of the 42nd ACM technical symposium on Computer science education*. [S.l.], 2011. p. 499–504.

MIYADERA, Y.; HUANG, N.; YOKOYAMA, S. A programming language education system based on program animation. In: *Proceedings of Educational Uses of Information and Communication Technologie, in IFIP 16th World Computer Congress*. [S.l.: s.n.], 2000. p. 258–261.

PELZ, F. D. *Um gerador de dicas para guiar novatos na aprendizagem de programação*. Tese (Dissertação (Mestrado em Computação Aplicada)) — Universidade do Vale do Itajaí, 2014.

PRESSMAN, R. S. *Engenharia de software - Uma Abordagem Profissional*. [S.l.]: Amgh Editora, 2016. v. 8.

SCHORSCH, T. Cap: an automated self-assessment tool to check pascal programs for syntax, logic and style errors. In: ACM. *ACM SIGCSE Bulletin*. [S.l.], 1995. v. 27, n. 1, p. 168–172.

WEBER, G.; BRUSILOVSKY, M.; STEINLE, F. Elm-pe: An intelligent learning environment for programming, 1996. *Obtain in: www. psychologie. uni-trier. de*, v. 8000, 2014.