Avaliação de uma abordagem para auxiliar a correção de erros de aprendizes de programação

Galileu Santos de Jesus[1], Kleber Santos[1], Jaine Conceição[1], Elisalvo Ribeiro[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, elisalvo.ribeiro}@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 the initial programming disciplines and guiding them through the syntax errors presented when performing a submission to the online judge. In order to evaluate this approach, a case study with undergraduate classes was also conducted. A controlled experiment, including an analysis with statistical tests confirms the hypothesis, where the approach increased the ability to correct errors, especially among students with low english proficiency that have succeeded in the initial programming discipline.*

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. Também foi feito um estudo de caso com turmas de graduação para avaliar esta abordagem, através de um experimento controlado, assim como sua análise com testes estatísticos para confirmação de hipótese, onde o estudo concluiu que a abordagem aumentou a capacidade de corrigir erros, além de guiar os alunos mais enfaticamente, principalmente entre alunos com baixo domínio da língua inglesa e que lograram êxito na disciplina inicial de programação.*

1. Introdução

Segundo Miyadera, Huang e Yokoyama (2000) uma das maiores dificuldades de um estudante de programação é entender cada passo da execução do programa. Assim como entender e aprender a gramática de uma linguagem de programação, consertar erros de sintaxe em um programa, desenvolver novos algoritmos, escrever um novo programa, depurar e consertar erros em um programa, em ordem crescente de dificuldade.

Segundo Weber, Brusilovsky e Steinle (2014), a tarefa de aprender a programar inclui:

- Adquirir habilidade em resolução de problemas, que envolve a identificação de metas e construção de planos de programação;

DOI: 10.5753/cbie.sbie.2018.1

- Aprender a sintaxe de uma linguagem de programação;
- Aprender sobre lógica de programação, ou seja, entender o comportamento do computador na execução de um programa, o que inclui compreender a semântica da linguagem de programação;
- Utilização de um ambiente de programação;
- Realização de testes e depuração de programas.

Para estudantes iniciantes em programação, mensagens de erros podem ser muitas vezes fonte de medo e frustração, coibindo frequentemente 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.

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 exibe que o aluno tentou utilizar uma variável que não foi definida anteriormente, então foi exibida a seguinte mensagem: ”*NameError: name ’P’ is not defined*- nas linhas 4 e linha 25, sinalizando que o nome *P* não foi definido. Porém, foram exibidos diversos erros pertencentes ao sistema de exceção, encontrados nas linhas 6 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.

Este trabalho aborda as dificuldades dos alunos em resolver problemas de programação para disciplinas iniciais, que muitas vezes se tornam objeto de medo e frustração, pois as ferramentas frequentemente não fornecem um *feedback* de forma clara

e concisa. O público alvo são alunos que estão aprendendo a programar, os quais estão desfrutando do primeiro contato com uma linguagem de programação e por isso estão ainda aprendendo a utilizar conceitos introdutórios como estruturas condicionais, estruturas de laços de repetição, declaração de funções, uso de funções e comandos específicos, declaração de variáveis e outros.

Este artigo tem como contribuição principal fazer uma avaliação de uma abordagem que almeja proporcionar 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 exibiçao e transcrição da mensagem amigável quando aplicada à mensagem ilustrada na Figura 1.

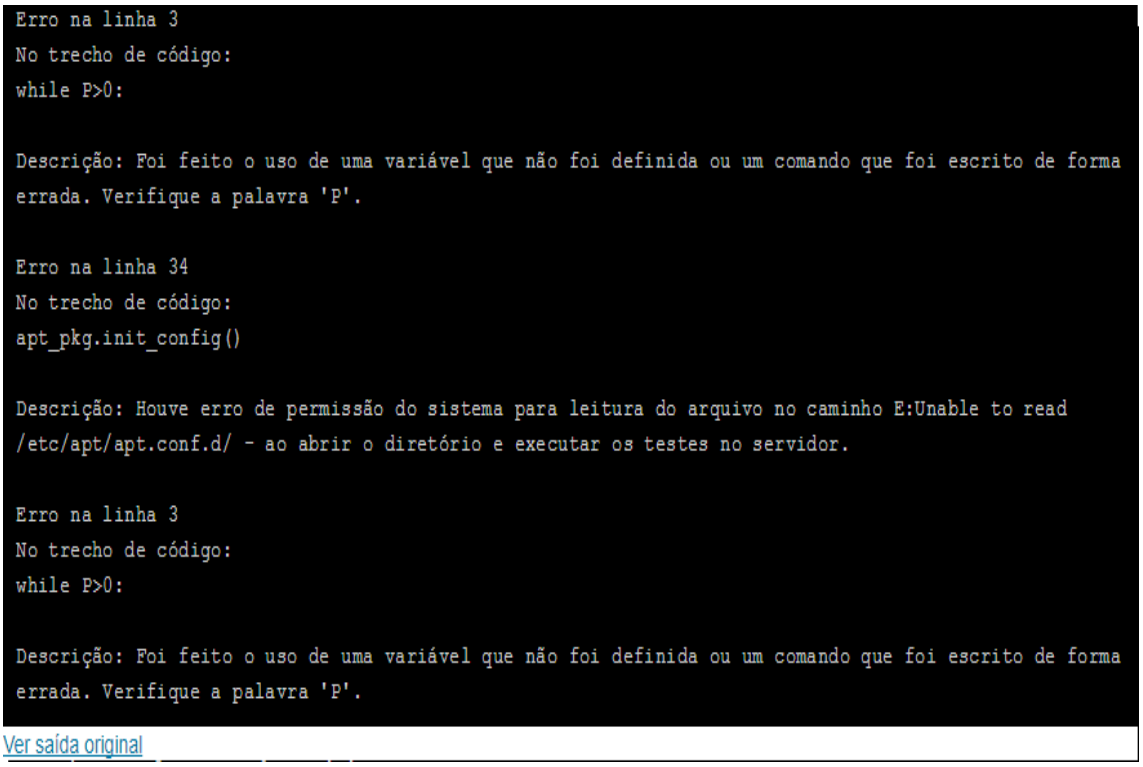

Figura 2. Exemplo de mensagem exibida com a abordagem desenvolvida ao aplicar na mensagem ilustrada na Figura 1.

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, permitirá uma maior eficiência na resolução destes erros, facilitando o ensino-aprendizagem de disciplinas iniciais de programação devido ao processo de correção ser mais rápido e fácil.

2. Trabalhos relacionados

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) proporciona aos alunos um *feedback* amigável e automatizado sobre erros comuns de sintaxe, lógica e de estilo dos programas Pascal – capaz de 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.

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.

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.

A Figura 3 ilustra uma comparação dos trabalhos encontrados de acordo com o *feedback*, considerando os seguintes critérios:

- Exibição dos casos de teste: se exibe os casos de testes onde ocorreu erro no código-fonte, obedecendo a dinâmica de juiz *on-line*;
- Possui mensagens melhoradas: se oferece o recurso de *feedback* melhorado dos erros sintáticos;
- É uma ferramenta *on-line*: se é uma ferramenta disponível *on-line* que pode ser acessada remotamente;
- Linguagem: quais linguagens a ferramenta aplica seu estudo ou funcionalidade;
- Experimentação: se foi realizada alguma experimentação com o uso da ferramenta ou para validação de hipóteses.

O The Huxley aceita várias linguagens, mas como resultado da implementação e integração da abordagem proposta por nosso trabalho no mesmo, passou a oferecer mensagens amigáveis (entendíveis) apenas em Python. Apesar disto, não há qualquer limitação ou restrição que impeça de se estender o suporte às outras linguagens disponíveis atualmente no The Huxley.

3. Metodologia

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

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

Critério	Expresso	CAP	Portugol	DrScheme	The Huxley + Proposta
Exibe casos de teste			X		X
Mensagens melhoradas	X	X	X	X	X
On-line					X
Linguagem	Java	Pascal	Portugol	Scheme	Python, C, C++, Java e Octave
Experimentação			X	X	X

Figura 3. Comparação dos trabalhos relacionados com o The Huxley.

das submissões de programas de aprendizes para realização de uma análise estatística. Além de permitir submissões utilizando as linguagens C, C++, Python, Java, Octave e Pascal.

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.

Obtivemos que 71,97% das submissões são consideradas como erradas. Fazendo uma análise por linguagem de programação, Python se destaca, onde concentra 82,13% das submissões erradas contra 17,87% corretas. Isso significa que dentre todas as linguagens, o maior percentual de submissões não aceitas é desta. Em segundo lugar está a linguagem Octave, com 72% consideradas erradas contra 28% corretas. A terceira posição é ocupada pela linguagem Java, com 68,48% erradas e 31,52% corretas. A quarta posição é da linguagem C, com 68,45% erradas e 31,55% corretas. Seguida pela linguagem Pascal com 65,83% erradas e 34,17% corretas e a linguagem C++ com 63,48% erradas e 36,52% corretas. Este foi o principal motivo que nos fez escolher a linguagem Python como estudo inicial do trabalho.

A Figura 4 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.

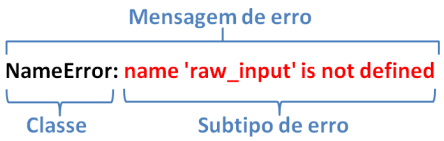

Figura 4. Mensagem de erro mapeada.

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.

A seguir são ilustradas algumas mensagens utilizando a abordagem *original* e logo em seguida a mensagem amigável correspondente.

[2] gg.gg/requi

- *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.
- *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.

A integração com o The Huxley foi feita através de interface gráfica, onde ao clicar no link *ver saída amigável* é feita a requisição ao *web service* no qual está sendo executada a nossa abordagem, passando os respectivos parâmetros e obtendo uma *string* como retorno contendo o resultado do processamento da requisição. 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.

Após isto, 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.

A seleção dos participantes ocorreu de forma aleatória entre os que estavam presentes em horário da disciplina. Foram selecionados 26 (vinte e seis) participantes, sendo possível somente a participação destes, por conta da logística e quantidade de computadores disponíveis.

O experimento foi projetado em um contexto pareado em que um mesmo grupo avaliará a abordagem original e outro a amigável, sendo realizado ao final da disciplina. Cada participante resolveria 6 (seis) problemas, cada problema com 3 (três) erros sintáticos, utilizando a abordagem amigável em 3 (três) problemas e a original nos outros 3 (três), representando os erros mais comuns ocorridos na base de dados do The Huxley. Foi usado um problema extra para que os participantes pudessem absorver os procedimentos a serem seguidos durante o experimento. Isso mitiga problemas relacionados ao tempo necessário para entender o fluxo de como se comportar durante o experimento.

A distribuição se deu da forma mais aleatória possível, sendo feita com o auxílio de um programa em Python, onde o *n-ésimo* participante representa o oposto do seu sucessor, sendo agrupados em pares, ou seja, o participante 1 (um) começa o problema 2 (dois) com a amigável, então o participante 2 (dois) começa com a abordagem original. A sequência de solução de problemas também foi aleatória, para que não influenciasse no

resultado final, simulando ao máximo situações cotidianas.

Os problemas foram selecionados a partir de uma busca feita nas submissões que continham erros sintáticos na base do The Huxley e cruzados com os problemas já resolvidos pelos participantes, para que fossem inéditos. Ao encontrar um determinado código-fonte que apresentava aquele erro, foi verificado o nível do problema e se sua descrição estava coesa.

Todavia, o estudo experimental deste trabalho tem por objetivo obter dados quantitativos relacionados à aplicação das duas abordagens: uso de mensagens amigáveis e o uso de mensagens originais, sendo realizado em turmas reais de graduação durante a disciplina Programação Imperativa.

Ao final do experimento, foi aplicado um questionário visando responder algumas questões de pesquisa e sabermos opiniões dos participantes acerca das abordagens, assim como informações qualitativas sobre os mesmos e agruparmos os dados.

Foram aplicados testes estatísticos para verificar se os dados seguem uma distribuição normal ou não. Para os dados que não seguem, foi utilizado o Teste de *Wilcoxon Pareado* e para os testes que seguem uma distribuição normal, foi utilizado o *Teste-T Pareado*, conforme recomendações de Wohlin et al. (2000), com o objetivo de comparar as duas abordagens de acordo com o tempo gasto para solucionar um determinado erro, comparando-se se a média de tempo da abordagem amigável foi **menor** que a original.

Para criação das hipóteses, os dados foram agrupamos em 3 (três) grupos. O primeiro foi por participantes. O segundo foi por nível de inglês do participante, informado através de preenchimento do formulário. O terceiro foi de acordo com a média obtida na disciplina.

4. Resultados

De acordo com o trabalho de Torman, Coster e Riboldi (2012) ao utilizar amostras pareadas, é necessário que a variável aleatória da diferença entre as duas amostras tenha distribuição normal. Por isso, ao aplicar o teste de *Shapiro-Wilk* para normalidade dos dados, foi utilizada a diferença das duas abordagens.

Com a aplicação do teste de *Shapiro-Wilk*, considerando α = 0,05, inferindo *p-value* $< \alpha$ para todas as abordagens e agrupamentos analisados diante de suas variáveis de estudo. Sendo assim, a hipótese H_0 é rejeitada caso a sentença *p-value* $< \alpha$ seja verdadeira, caso contrário, não é rejeitada.

Como ferramenta para agrupar os dados, foi utilizada o software Excel, através de recursos como tabelas dinâmicas e agrupamento por filtros. Para aplicação dos testes de normalidade foi utilizada a ferramenta R Studio com a linguagem de programação R, usando-se o comando *shapiro.test(dados)* sobre os dados em questão.

Tabela 1: Aplicação do teste de normalidade *Shapiro-Wilk* com relação ao tempo em segundos.

Agrupamento	**Shapiro(Amigável-Original)**	**Inferência**
Por participante	*p-value* = 0.04839	**Não normal**
Por nível de inglês	*p-value* = 0.5451	**Normal**

Por média	*p-value* = 0.01082	**Não normal**

Ao aplicar os testes com relação ao tempo, tem-se como objetivo comparar se a média de tempo da abordagem amigável foi **menor** que a original. Para isso, foram utilizados os seguintes parâmetros:

- *amigavel* e *original* representam os dados de entradas com as médias de tempo de acordo com seu respectivo agrupamento;
- *paired=TRUE* representa que está usando o teste com os dados pareados;
- *alternative="**less**"* está aplicando o teste demarcando que a hipótese alternativa seja menor, ou seja, que os dados referentes à abordagem amigável são menores que a original;
- *confLevel=0.95* significa que o nível de confiança é de 95%.

(A) **Hipótese 1:** Dados agrupados por participante
De acordo com as análises apresentadas na Tabela 1, temos que os dados não seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Wilcoxon Pareado* sobre os dados.
Aplicou-se o teste de *Wilcoxon Pareado* na linguagem R: *wilcox.test(amigavel, original, paired=TRUE, alternative="less", confLevel=0.95)*. Foi obtido um ***p-value*** de **0.2262**, sendo **maior** que o α (**0.05**), logo, **não rejeitamos a hipótese $\mathbf{H}_0$**. Com isso, não podemos afirmar que as médias de tempos das duas abordagens com relação aos participantes são diferentes.

(B) **Hipótese 2:** Dados agrupados por nível de inglês
De acordo com as análises apresentadas na Tabela 1, temos que os dados seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Teste-T Pareado* sobre os dados.
Aplicou-se o teste de *Teste-T Pareado* na linguagem R: *t.test(amigavel, original, paired=TRUE, alternative="less", conf.level=0.95)*. Foi obtido um ***p-value*** de **0.01988**, sendo **menor** que o α (**0.05**), logo, **rejeitamos a hipótese $\mathbf{H}_0$**. Com isso, podemos afirmar que as médias de tempos das duas abordagens com relação ao nível de inglês dos participantes são diferentes.

(C) **Hipótese 3:** Dados agrupados por média do aluno
De acordo com as análises apresentadas na Tabela 1, temos que os dados não seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Wilcoxon Pareado* sobre os dados.
Aplicou-se o teste de *Wilcoxon Pareado* na linguagem R: *wilcox.test(amigavel, original, paired=TRUE, alternative="less", confLevel=0.95)*. Foi obtido um ***p-value*** de **0.02325**, sendo **menor** que o α (**0.05**), logo, **rejeitamos a hipótese $\mathbf{H}_0$**. Com isso, podemos afirmar que as médias de tempos das duas abordagens com relação à média de notas dos participantes são diferentes.

A Tabela 2 ilustra uma síntese da aplicação dos testes ilustrando o *p-value* de cada hipótese e a rejeição ou não da hipótese nula.

Tabela 2: Aplicação dos respectivos testes de acordo com a normalização dos dados e à média dos tempos.

Agrupamento	Teste estatístico	Resultado (*p-value*)	Inferência
Por participante	*Wilcoxon Pareado*	**p-value = 0.2262**	**Não rejeitamos H_0**
Por nível de inglês	*Teste-T Pareado*	**p-value = 0.01988**	**Rejeitamos H_0**
Por média	*Wilcoxon Pareado*	**p-value = 0.02325**	**Rejeitamos H_0**

4.1. Ameaças à validade

O nível diferenciado de conhecimento dos participantes da linguagem Python pode fazer com que os participantes menos experientes afetem os resultados negativamente. O nível de inglês foi definido em questionário aplicado ao final do experimento, porém não foi realizada nenhuma prova para aferir com precisão este valor.

Não foi contabilizado nos testes experimentais o tempo adicional gasto pelos participantes para clicar no link *ver saída amigável* para acessar a mensagem amigável, esta ação contabiliza cerca de três segundos: clicar no link, aguardar o sistema processar a requisição, exibir a mensagem amigável e finalmente o usuário rolar a tela até o início da mensagem para visualizá-la. Contudo, a abordagem original informa a mensagem assim que é feita a submissão.

5. Considerações Finais e Trabalhos Futuros

Foi realizada uma análise na base de dados do juiz *on-line* The Huxley para determinar qual linguagem de programação faria parte da implementação deste trabalho. Nesta análise foram encontradas evidências que enfatizaram a escolha da linguagem Python como predileção, visto que foi a linguagem que mais possuiu um percentual de submissões com erros sintáticos na base em questão.

Há fortes indícios de que a utilização de mensagens amigáveis pode ajudar no entendimento de mensagens com erros sintáticos, guiando o aluno para sua correção ao apresentar dicas e sugestões de forma mais simples e compreensível. Ao validar as hipóteses de acordo com o nível de inglês e média final obtida na disciplina pelos participantes, foi constatado que as duas abordagens se diferem estatisticamente com relação ao tempo, havendo evidências que uso da abordagem amigável diminui o tempo para solucionar um erro sintático, mesmo para quem possui um nível de inglês mais alto, visto que há mensagens que não são tão claras, porém não há indícios de diferença estatística dos dados ao comparar os dados por participante com relação ao tempo gasto para resolver um problema.

Após o experimento, cada participante respondeu a um questionário. Com relação ao nível de dificuldade em entender as mensagens de erros originais, cerca de 23,1% responderam que é muito alto, 7,7% que é alto, 23,1% que é médio, 36,5% que é baixo e 7,7% que é muito baixo. De acordo com o nível de dificuldade em entender as mensagens amigáveis, 3,8% responderam que é muito alto, 0% que é alto, 11,5% que é médio, 26,9% que é baixo e 57,7% que é muito baixo. No tocante ao nível de utilidade do uso das mensagens amigáveis, 30,8% responderam que é muito alto, 38,5% que é alto, 23,1% que é médio, 0% que é baixo e 7,7% que é muito baixo. Desta forma, considerando a opinião

dos participantes do experimento, pode-se concluir que as mensagens amigáveis são mais fáceis de serem entendidas do que as originais e também que foram mais úteis.

No contexto de aprendizes de programação, a utilização da técnica de apresentação de mensagens amigáveis pode diminuir o medo e frustrações, possibilitando assim uma correção mais rápida e melhor guiada. Porém, não descartamos o uso das mensagens originais, visto que não foram encontradas diferenças estatísticas evidentes com relação à quantidade de erros corrigidos e ao agrupar por participante com relação ao tempo na realização do estudo experimental.

Como trabalho futuro, faremos a mesma pesquisa com as outras linguagens de programação aceitas pelo The Huxley, como C, C++, Java, Octave e Pascal.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

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.

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.

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.

TORMAN, V. B. L.; COSTER, R.; RIBOLDI, J. Normalidade de variáveis: métodos de verificação e comparação de alguns testes não-paramétricos por simulação. *Clinical & Biomedical Research*, v. 32, n. 2, 2012.

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.

WOHLIN, C. et al. *Experimentation in software engineering: an introduction. 2000*. [S.l.]: Kluwer Academic Publishers, 2000.