Uma ferramenta para apoiar generalização de *feedback* em disciplinas de programação introdutória

Luiz Fernando da Silva[1,2], **Alexandre de Andrade Barbosa**[1]

[1]Universidade Federal de Alagoas (UFAL)
Arapiraca – AL – Brasil

[2]Bolsista Edital PIBIC UFAL 2019-2020

{luiz.fernando, alexandre.barbosa}@arapiraca.ufal.br

Abstract. *The basic concepts programming are part of introductory programming disciplines , being of great importance for training the students. Generally, to assess the understanding of these concepts, teachers adopt practical coding activities, which can be a difficult task, if the feedback from the teacher isn't in due time, and may lead the student to not know if their understanding is correct. To minimize difficulties of this nature, several computational resources are sought by teachers. Thus, this work aims to present a tool proposal, which helps in generalizing feedback through code groupings based on software metrics.*

Resumo. *Os conceitos básicos de programação fazem parte das disciplinas de programação introdutória, sendo de grande importância para a formação dos estudantes. Geralmente para avaliar o entendimento destes conceitos, os professores adotam atividades práticas de codificação, o que pode ser uma tarefa difícil, caso o feedback do professor não esteja em tempo adequado, podendo levar o aluno a desconhecer se o seu entendimento é correto. Para minimizar dificuldades desta natureza, diversos recursos computacionais são buscados pelos docentes. Portanto, o presente trabalho tem como objetivo apresentar uma proposta de ferramenta, que auxilie na generalização de feedback por meio de agrupamentos de códigos baseados em métricas de software.*

1. Introdução

O ensino nas disciplinas de programação introdutória em cursos de nível técnico ou superior, apresentam os conceitos básicos de programação ao discente, sendo em alguns dos casos o primeiro contato com o tema. Os conteúdos abordados nestas disciplinas são considerados de grande importância para a formação dos estudantes [Amaral et al. 2017]. Assim, sobre o contexto do processo de ensino-aprendizagem, comumente são adotas atividades práticas de codificação [Barbosa et al. 2017].

As atividades de codificação envolvem, no geral, entender os problemas e desenvolver soluções utilizando uma linguagem de programação, onde é fundamental que o discente consiga interpretar o problema a fim de analisar as possíveis soluções, com o intuito de que o mesmo absorva os conceitos trabalhados [Souza et al. 2016].

No entanto, a codificação pode ser um processo difícil para parte dos estudantes por diversos fatores, alguns deles por exemplo, ligado ao método de ensino; o nível de abstração dos problemas; e o *feedback* do professor em tempo adequado, no qual é

destacado, pois, sem o *feedback* em tempo adequado, o discente desconhece se o seu entendimento sobre o conteúdo é correto [Santos et al. 2019, Barbosa et al. 2017]. Consequentemente, para apoiar o ensino e minimizar as dificuldades de atendimento aos alunos, os professores buscam recursos computacionais distintos que possam auxiliá-los [Alves and Jaques 2014].

Dentro deste contexto, durante a avaliação dos códigos observa-se similaridade entre as soluções, ou seja, existem semelhanças entre elas. Na tese de doutorado [Barbosa 2018] averiguou a possibilidade de agrupar as soluções similares, a partir das medidas de métricas de software, com o intuito de reutilizar os *feedbacks*. Portanto, com base na referida tese, o objetivo deste artigo é apresentar uma proposta de ferramenta, para o apoio da generalização de *feedback* em disciplinas de programação introdutória.

Diante disto, o presente artigo está organizado da seguinte forma: na Seção 2 são apresentados os trabalhos relacionados; na Seção 3 é descrita a fundamentação teórica; na Seção 4 são apresentadas as tecnologias utilizadas para a construção da ferramenta; na Seção 5 é descrita a ferramenta e o desenvolvimento de suas funcionalidades; na Seção 6 são apresentadas as considerações sobre o trabalho e as próximas etapas a serem executadas; e finalmente na Seção 7 estão os agradecimentos à instituição onde o projeto é desenvolvido.

2. Trabalhos relacionados

Na literatura, publicações relacionadas ao uso de ferramentas que auxiliem a avaliação de códigos em disciplinas de programação são diversificadas. Sendo comum a utilização de juízes online, que possuem como característica principal a aplicação de testes de aceitação, para fins avaliativos das soluções, como descrito no trabalho de [Yulianto and Liem 2014], que aborda o aprimoramento da capacidade de análise de juízes online, com o objetivo de avaliar também a qualidade dos códigos produzidos pelos estudantes.

No trabalho de [Silva et al. 2014] investigou-se a construção de um arcabouço com mecanismos de análise automática, aplicado em soluções submetidas por discentes em turmas de programação introdutória, no qual pode-se observar o uso combinado de um analisador sintático, um analisador de construtos e um analisador estrutural. Estes analisadores por sua vez, verificavam características do código sobre: corretude sintática, corretude semântica, variáveis, operadores, estruturas de repetição, e entre outras.

A pesquisa conduzida por [Yin et al. 2015] buscou agrupar soluções de problemas desenvolvidos por estudantes, utilizando métricas de similaridade (distância de edição de texto e distância de edição de árvore) e o algoritmo de agrupamento *OPTICS*. Com o estudo, os autores concluíram que para a maioria das submissões, existe a possibilidade de gerar agrupamentos de modo automático, assim o professor pode utilizar as informações destes grupos para produzir o *feedback* aos estudantes.

Nenhum dos trabalhos citados utilizam técnicas de aprendizagem de máquina para fornecer opções de adaptar uma avaliação, por meio de critérios oriundos do professor sem fixá-los, como apresentado no trabalho de [Barbosa 2018], que utilizou o algoritmo de *clustering* (agrupamento) *K-means* para a formação dos agrupamentos e as seguintes métricas de software como critérios na geração dos grupos: métrica de complexidade

ciclomática; métricas de contagem de linhas; métricas de Halstead; métricas de corretude (sintática e funcional) e as métricas de similaridade (coeficiente de Jaccard, distância de edição de texto e distância de edição de árvore). Os resultados alcançados na investigação sugerem que é possível reutilizar *feedbacks*, pois, observou-se que a concordância obtida entre as notas geradas e as fornecidas por um avaliador, variaram entre uma concordância razoável no pior cenário, e uma concordância perfeita para o melhor cenário.

Portanto, para generalizar o *feedback* avaliativo, sem que critérios sejam fixados, a proposta na tese de [Barbosa 2018] para geração de agrupamentos baseado sobre um conjunto de métricas de software, fomenta a principal contribuição para o desenvolvimento da ferramenta descrita neste artigo.

3. Fundamentação teórica

Diante do exposto, se faz necessária a compreensão dos seguintes conceitos envolvidos na construção do trabalho: o conjunto de métricas de software [Sommerville 2011] e o algoritmo de *clustering K-means* [Piech 2013].

3.1. Métricas de software

Uma métrica de software de acordo com [Sommerville 2011] é uma característica de um sistema que pode ser medido, ou ainda um meio de verificar se um software é detentor de determinada propriedade. Deste modo, as métricas de contagem de linhas e as métricas de Halstead [Halstead 1977] compõem o estado atual da ferramenta.

As métricas de contagem de linhas tem como função calcular a quantidade de linhas no código, com o objetivo de mensurar o tamanho da solução. Dentre as métricas existentes destacam-se:

- *Lines Of Code* (LOC) – calcula o número total de linhas.
- *Source Lines Of Code* (SLOC) – calcula o número total de linhas de código.
- *Logical Lines Of Code* (LLOC) – calcula o número de linhas lógicas de código.

As métricas de Halstead tem como objetivo identificar propriedades (operadores e operandos) mensuráveis do código e as relações entre estas propriedades [Lacchia 2020]. Então, a partir da extração dos seus valores são computadas as medidas a seguir:

- Número de operadores distintos (n_1) – quantidade de operadores no código, desconsidera as repetições.
- Número de operandos distintos (n_2) – quantidade de operandos no código, desconsidera as repetições.
- Número total de operadores (N_1) – quantidade de operadores no código, considera as repetições.
- Número total de operandos (N_2) – quantidade de operandos no código, considera as repetições.
- Vocabulário ($n = n_1 + n_2$) – representa a soma dos operadores e operandos distintos.
- Tamanho ($N = N_1 + N_2$) – representa a soma dos operadores e operandos totais.
- Volume ($V = N \times \log_2 n$) – representa a quantidade de informação no código, proporcional ao tamanho necessário para o armazenamento do mesmo.

3.2. *K-means*

Um algoritmo de *clustering* é uma técnica que utiliza a abordagem de aprendizagem não-supervisionada, para agregar elementos em determinados *clusters* (grupos) considerando as características semelhantes entre os elementos. O objetivo é formar grupos com características homogêneas entre os componentes de um mesmo grupo e heterogêneas entre os componentes de grupos distintos [Oliveira et al. 2014].

O *K-means* recebe este nome, pois, encontra 'k' *clusters* diferentes num determinado conjunto de dados, o 'k' determina o parâmetro da quantidade de grupos e o centroide usado para definir o *cluster* (grupo). O centro de cada *cluster* denomina-se centroide, desta maneira, a tarefa do algoritmo é localizar o centroide através de alguma métrica ou aleatoriamente, e atribuir o ponto encontrado a um *cluster* [Piech 2013, Honda 2017].

Por meio de iterações o *K-means* aprimora seus resultados até encontrar um resultado final. Cada centroide define um *cluster*, logo cada elemento será associado ao seu centroide mais próximo. O cálculo da distância é realizado, na maioria dos casos, através da distância euclidiana entre dois elementos [Honda 2017]. Na ferramenta, este algoritmo é o responsável por auxiliar na formação dos grupos de códigos.

4. Materiais e métodos

O desenvolvimento da ferramenta é realizado empregando o método de desenvolvimento de software *eXtreme Programming* (XP). O método ágil XP é voltado para equipes de pequeno e médio porte, que baseiam a criação de software em requisitos que se modificam com frequência e em que o desenvolvimento é orientado a objeto [Beck 2004]. Este processo também utiliza o desenvolvimento iterativo ou incremental, onde uma versão base do sistema é implementada no início do projeto e vai ganhando novas funcionalidades ao longo do tempo [Teles 2014].

Para o *front-end* da aplicação utilizou-se o Bootstrap 4.3.x, uma biblioteca que faz uso de HTML, CSS e JavaScript, voltada para a construção do *front-end* de projetos web [Bootstrap 2020]. Enquanto o *back-end* da ferramenta utiliza o Django, um *framework* web de alto nível e de código aberto escrito com a linguagem de programação Python. Toda a aplicação descrita neste artigo foi criada com a versão 2.2.x do *framework*, que necessita da linguagem de programação Python a partir da versão 3.5.x ou superior para ser executada [Django 2020].

O Django faz uso do *Model Template View* (MTV) um padrão arquitetural de software que separa a aplicação em três camadas, descritas a seguir com base nas ações do *framework* [MDN 2019]:

- *Model*: é a camada responsável pelos objetos em Python que define a estrutura de dados da aplicação, nela são fornecidos os mecanismos de gerenciamento (adicionar, atualizar e excluir) e as consultas dos registros na base de dados.
- *Template*: é um arquivo de texto, geralmente um arquivo HTML, com espaços reservados usados para apresentar o conteúdo de uma *view* com dados de um *model*.
- *View*: é uma função que tem como objetivo manipular solicitações através de uma *request* (requisição) HTTP, retornando uma *response* (resposta) HTTP, esta camada acessa os dados necessários para satisfazer uma requisição por meio do *model* e apresentar a resposta no *template*.

Para a persistência de dados optou-se pelo Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL na versão 8.0.x. Este SGBD utiliza banco de dados relacional, ou seja, é um tipo de banco que armazena os registros da aplicação através de tabelas. O modelo lógico (banco de dados, tabelas, visualizações, linhas e colunas) oferece um ambiente de programação flexível, desta forma o programador define as regras aplicadas nos relacionamentos entres as entidades envolvidas [Oracle 2020].

Sobre a extração das métricas de software foi adicionada a ferramenta Radon, que tem como funcionalidade calcular as medidas de software de códigos Python. Dentre as métricas disponibilizadas pelo Radon, destacam-se as métricas de contagem de linhas e as métricas de Halstead, pois são utilizadas no projeto e têm seus valores salvos no banco de dados [Lacchia 2020].

Por fim, no que refere-se à geração dos agrupamentos, a ferramenta descrita neste trabalho utiliza a biblioteca SciPy da linguagem Python, uma vez que esta biblioteca possui a implementação do algoritmo *K-means*. Na criação dos grupos, adotou-se o valor de $k = 10$, o mesmo valor utilizado pelos experimentos conduzidos por [Barbosa 2018], com o objetivo de evitar discordâncias e inconsistências entre os agrupamentos de códigos formados pelo algoritmo.

5. Desenvolvimento da ferramenta

A ferramenta atualmente funciona com a submissão de códigos desenvolvidos com a linguagem de programação Python, e o seu desenvolvimento ocorre como parte de um projeto do Programa Institucional de Bolsas de Iniciação Científica (PIBIC) 2019-2020 da Universidade Federal de Alagoas (UFAL).

Para o levantamento de requisitos do projeto foram definidos casos de uso, a partir desta atividade constatou-se que deveriam existir dois tipos distintos de atores: 'Professor' e 'Estudante', estes por sua vez são uma especificação de um ator genérico denominado 'Usuário' que oferece interações comuns de cadastro e acesso ao sistema. Separar as interações que são diferentes entre os atores, contribuiu numa maior compreensão do papel a ser desempenhado pelos futuros usuários da ferramenta (Figura 1).

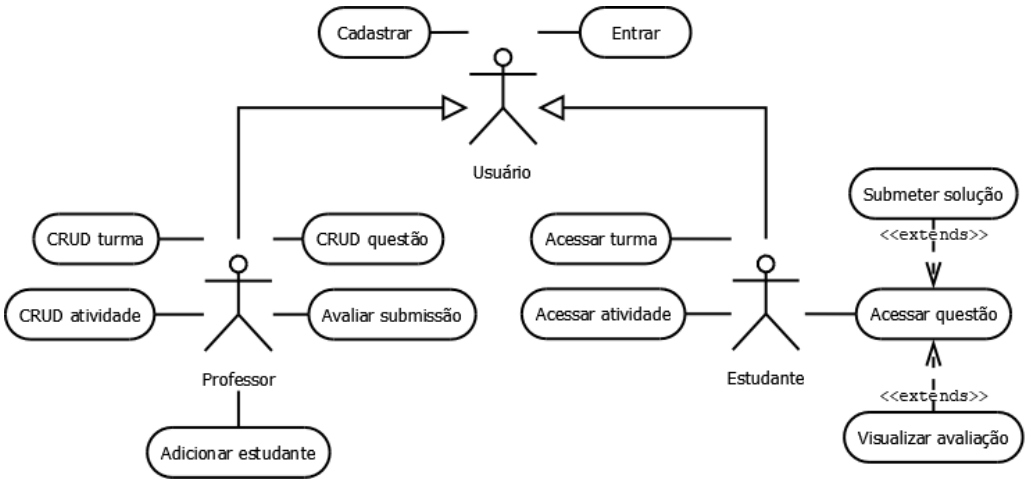

Figura 1. Diagrama de casos de uso da ferramenta.

Com a intenção de auxiliar o processo de codificação foi elaborado um diagrama de classes, que de acordo com [Sommerville 2011] tem por objetivo mostrar as classes de um sistema e as associações entre elas, onde uma classe pode ser pensada como uma instância do sistema. Na Figura 2 são apresentados os *models* que compõem a ferramenta, além dos seus atributos e as associações entre os mesmos.

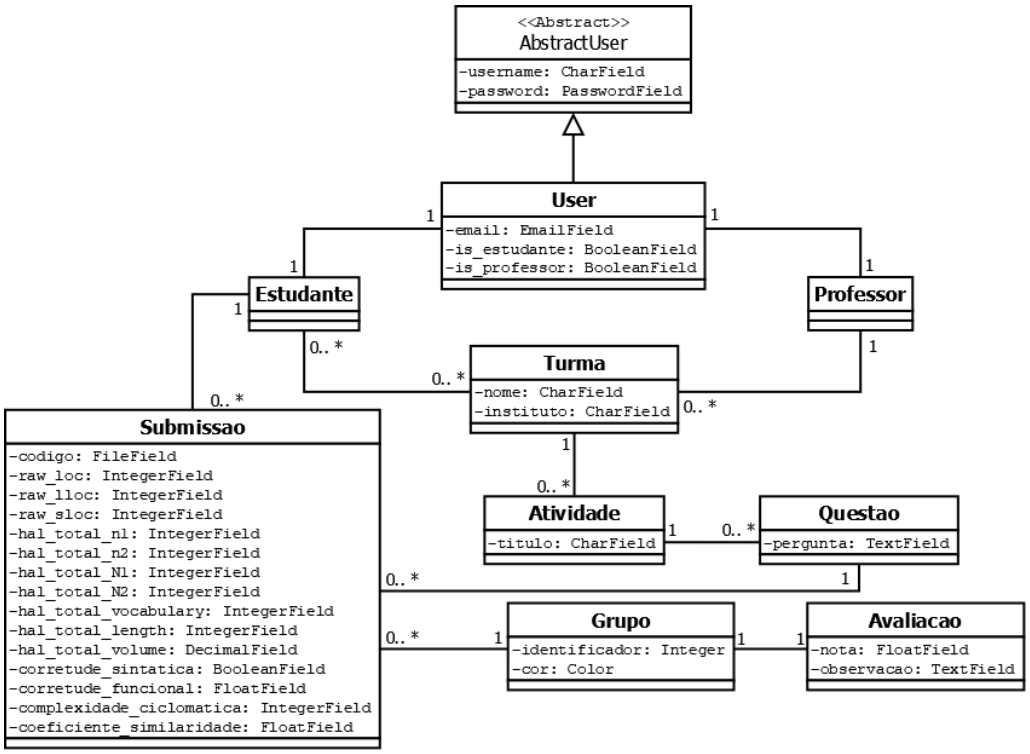

Figura 2. Diagrama de classes da ferramenta.

A codificação do sistema leva em consideração o padrão arquitetural de software MTV implementado pelo Django, o paradigma de programação orientado a objetos, assim, como o mapeamento objeto-relacional. Neste caso, a partir da construção dos *models* da aplicação é gerado pelo próprio *framework* o mapeamento entre as classes e as tabelas do banco de dados.

O Django implementa alguns *models* prontos para uso, como por exemplo, a classe '*AbstractUser*' adicionada ao escopo da aplicação para fins de autenticação (que verifica se um usuário é quem ele afirma ser) e autorização (que determina as funções que um usuário autenticado pode fazer no sistema) [Django 2020].

Desta forma, por meio do princípio de herança da orientação a objetos foi definida a classe '*User*', que herda os atributos e as funcionalidades de '*AbstractUser*', permitindo o desenvolvimento dos dois tipos de usuários (professor e estudante) e a separação do acesso as suas respectivas funcionalidades. Os demais *models* foram desenvolvidos seguindo o que foi definido do diagrama de classes (Figura 2).

Sobre os atores que compõem o diagrama de casos de uso na Figura 1, o 'Professor' é o ator responsável pelas ações de criar, recuperar, atualizar e remover (CRUD – *Create, Retrieve, Update and Delete*), relacionadas ao gerenciamento da turma, atividade

e questão. De modo que é realizado pelo professor: a adição ou remoção de estudantes na turma; a publicação de atividade com base nos temas trabalhados; a criação de um conjunto de questões na atividade, para que o estudante realize a submissão do código-fonte; e avaliação do código-fonte submetido pelo estudante. A Figura 3 ilustra algumas destas atividades do docente, com o acesso a partir da *dashboard* deste usuário.

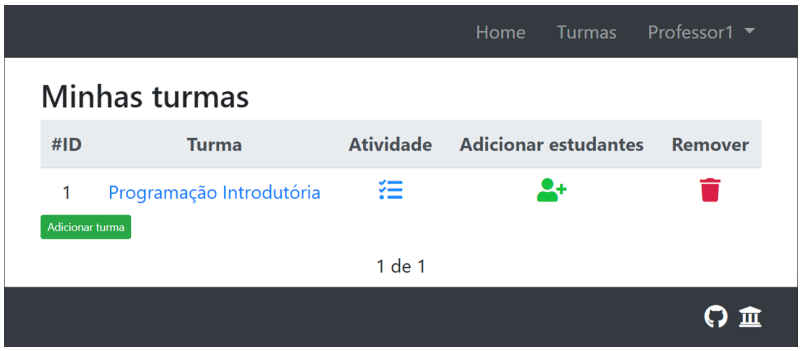

Figura 3. *Dashboard* do professor.

Dentre as interações do ator 'Professor' destaca-se 'Avaliar submissão' (Figura 1), onde ocorre o processo de agrupamento dos códigos a partir: das soluções submetidas pelos alunos referente uma determinada questão; dos critérios de agrupamento escolhidos pelo professor; e do algoritmo *K-means*. Então, de acordo com os grupos formados o professor pode realizar a avaliação em relação aos agrupamentos, generalizando o *feedback* avaliativo para as soluções de um mesmo grupo, adicionando uma nota e uma observação ao invés de avaliar individualmente cada código submetido (Figura 4).

Figura 4. Professor adiciona a avaliação do código-fonte submetido.

O ator 'Estudante' em contrapartida pode: visualizar informações da turma em que está inserido; acessar a atividade disponibilizada na turma e o conjunto de questões

que compõe a referida atividade; submeter a solução de uma questão; e visualizar a avaliação do professor referente ao código-fonte submetido (Figura 1). Na Figura 5 é apresentada a *dashboard* deste usuário, no qual é permitido através do nome da turma, o acesso a atividade publicada pelo professor, ao conjunto de questões e consequentemente a submissão da solução.

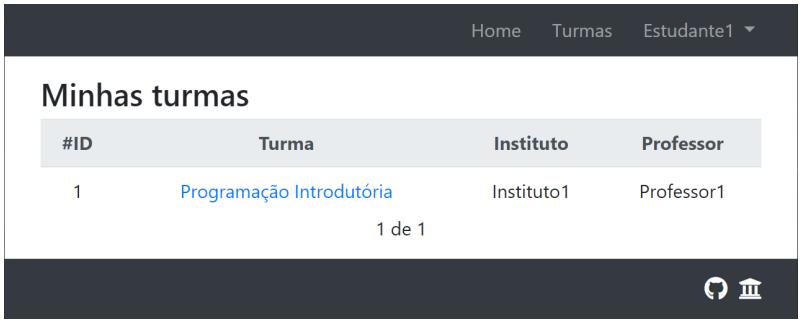

Figura 5. *Dashboard* do estudante.

Por fim, neste contexto do ator 'Estudante' (Figura 1), a interação 'Submeter solução' consiste no envio do código-fonte, no qual por meio deste são calculadas as métricas de software utilizadas na criação dos agrupamentos. Uma outra interação que merece destaque é 'Visualizar avaliação', nela o discente consegue recuperar a nota e a observação feita pelo professor mediante a análise da solução submetida (Figura 6).

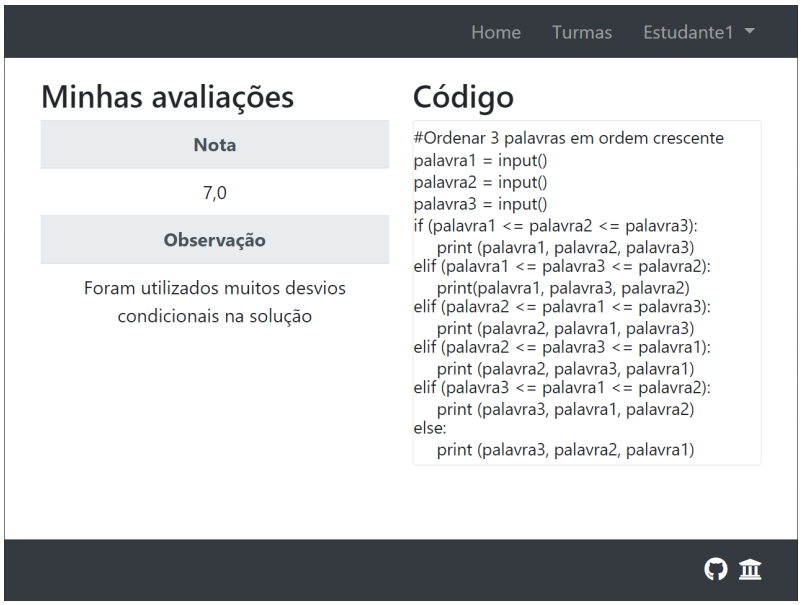

Figura 6. Estudante visualiza a avaliação do código-fonte submetido.

6. Considerações e próximas etapas

Neste artigo foi apresentada uma ferramenta voltada para o auxílio em disciplinas de programação introdutória, tendo em vista o apoio da generalização de *feedback* aos discentes a respeito das atividades de codificação, sendo esta proposta baseada na tese de [Barbosa 2018].

A ferramenta em seu estado atual é capaz de: realizar o cadastro de estudantes e professores, de modo que as funcionalidades respectivas a cada usuário estejam apenas dentro do escopo do perfil escolhido; extrair métricas de software das soluções submetidas; agrupar estas soluções para fins de avaliação; e generalizar o *feedback* avaliativo.

Embora a versão apresentada não seja a versão final, as seguintes atividades englobam o término do desenvolvimento da ferramenta:

- Implementação de novas métricas de software – métrica de complexidade, métricas de corretude (sintática e funcional) e as métricas de similaridade (coeficiente de Jaccard, distância de edição de texto e distância de edição de árvore), com o propósito de diversificar os critérios de agrupamentos.
- Modificação na interface da ferramenta para uma melhor experiência dos usuários.

Com a finalização do desenvolvimento, o sistema passará pela etapa de teste e validação, após sua conclusão, os resultados obtidos serão comparados com os resultados da investigação de [Barbosa 2018]. Diante disto, como discussão futura espera-se que a ferramenta auxilie no processo de ensino-aprendizagem em disciplinas de programação introdutória.

7. Agradecimentos

O desenvolvimento do projeto vem sendo realizado através do Edital PIBIC UFAL 2019-2020 no Campus Arapiraca, com a concessão de uma bolsa de iniciação científica.

Referências

Alves, F. P. and Jaques, P. (2014). Um ambiente virtual com feedback personalizado para apoio a disciplinas de programação. *Anais do XXV Simpósio Brasileiro de Informática na Educação (SBIE) 2014*, pages 1078–1082.

Amaral, E., Camargo, A., Gomes, M., Richa, C., and Becker, L. (2017). Algo+: uma ferramenta para o apoio ao ensino de algoritmos e programação para alunos iniciantes. *Anais do XXVIII SBIE 2017*, pages 1677–1686.

Barbosa, A. A. (2018). *Minimizando o esforço de avaliação em disciplinas de programação introdutória utilizando agrupamentos adaptáveis*. Tese (Doutorado em Ciência da Computação). Universidade Federal de Campina Grande, Campina Grande, 2018.

Barbosa, A. A., Costa, E. B., and Brito, P. H. S. (2017). Uma abordagem adaptativa para gerar agrupamento de códigos em disciplinas de programação introdutória. *Anais do XXVIII SBIE 2017*, pages 1427–1436.

Beck, K. (2004). *Programação extrema aplicada: acolha as mudanças*. Bookman.

Bootstrap (2020). *Bootstrap about*. Get Bootstrap. Disponível em: https://getbootstrap.com/docs/4.3/about/overview. Acesso em: 28 fev. 2020.

Django (2020). *Django documentation*. Django Software Foundation. Disponível em: https://www.djangoproject.com. Acesso em: 11 fev. 2020.

Halstead, M. H. (1977). *Elements of software science (Operating and programming systems series)*. Elsevier.

Honda, H. (2017). *Introdução básica à clusterização*. Laboratório de Aprendizado de Máquina em Finanças e Organizações (LAMFO). Disponível em: https://lamfo-unb.github.io/2017/10/05/Introducao_basica_a_clusterizacao. Acesso em: 10 ago. 2020.

Lacchia, M. (2020). *Radon 4.1.0 documentation*. Read the docs. Disponível em: https://radon.readthedocs.io/en/latest. Acesso em: 11 fev. 2020.

MDN (2019). *Django introduction*. MDN Web Docs. Disponível em: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. Acesso em: 11 fev. 2020.

Oliveira, M. G., Monroy, N. A. J., Zandonade, E., and Oliveira, E. (2014). Análise de componentes latentes da aprendizagem de programação para mapeamento e classificação de perfis. *Anais do XXV SBIE 2014*, pages 134–143.

Oracle (2020). *MySQL 8.0 reference manual*. Oracle Corporation. Disponível em: https://downloads.mysql.com/docs/refman-8.0-en.pdf. Acesso em: 28 fev. 2020.

Piech, C. (2013). *K-means*. Stanford. Disponível em: http://stanford.edu/ cpiech/cs221/handouts/kmeans. Acesso em: 28 fev. 2020.

Santos, A. G. S., Neves, W. R. M., and Lopes, F. A. (2019). Uma perspectiva das múltiplas inteligências nas tecnologias utilizadas para o ensino de programação. *Anais da XIX Escola Regional de Computação Bahia-Alagoas-Sergipe (ERBASE) 2019*.

Silva, M. T., Costa, E. B., Barbosa, P. H., and Cavalcante, J. C. (2014). Um arcabouço para construção de mecanismos de análise de códigos de programação introdutória. *Anais do XXV SBIE 2014*, pages 1083–1092.

Sommerville, I. (2011). *Engenharia de software*. Pearson Prentice Hall, 9^{th} edition.

Souza, M. S. C., Costa, F. A. M., Silva, V. L., and Terra, D. C. (2016). Lord of code: uma ferramenta de apoio ao ensino de programação. *Anais do XXVII SBIE 2016*, pages 1316–1320.

Teles, V. M. (2014). *Extreme programming*. Novatec, 2^{nd} edition.

Yin, H., Moghadam, J., and Fox, A. (2015). Clustering student programming assignments to multiply instructor leverage. In *Proceedings of the Second 2015*, page 367–372. ACM Conference on Learning @ Scale.

Yulianto, S. V. and Liem, I. (2014). Automatic grader for programming assignment using source code analyzer. In *International Conference on Data and Software Engineering (ICoDSE) 2014*.