Um mapeamento sistemático sobre analisadores de código em disciplinas de programação

Alexandre A. Barbosa[1,3], **Allan L. Correia**[1], **Danilo V. B. da Costa**[1], **Evandro B. Costa**[2,3]

[1]Universidade Federal de Alagoas - Campus Arapiraca (UFAL)
Caixa Postal 61 - Arapiraca - AL - Brasil

[2]Instituto de Computação - Universidade Federal de Alagoas (UFAL)
Caixa Postal 15.064 - 91.501-970 - Maceió - AL - Brasil

[3]Departamento de Sistemas e Computação
Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 - Campina Grande - PB - Brasil

{alexandre.barbosa, allan.coreia, danilo.costa}@arapiraca.ufal.br

ebc.academico@gmail.com

Abstract. *Traditionally, in programming disciplines practical coding activities are adopted. The assessment of the students' solutions is not trivial, instead it is time consuming, error-prone and biased by the evaluator. Thus, in many researches it was proposed the use of automatic mechanisms for code analysis in programming courses. The objectives are to reduce professor workload and the possibility of providing an agile feedback. In this paper, a systematic mapping about code analysis in programming courses is presented. This systematic mapping intended to identify the objectives, the techniques and educational environments where code analysis is adopted, another objective is also to determine whether there is or not a growing interest in research on this topic.*

Resumo. *Tradicionalmente, em disciplinas de programação são adotadas atividades práticas de codificação. A análise das soluções propostas pelos alunos nestas atividades não é trivial. Sendo essa avaliação demorada, ela é sujeita ao viés e aos erros do avaliador. Visto isso, muitas pesquisas propuseram o uso de mecanismos automáticos para análise de códigos em disciplinas de programação, tendo como objetivos a redução da carga de trabalho do professor e a possibilidade de fornecer feedback de maneira mais ágil. Neste trabalho é apresentado um mapeamento sistemático da literatura (MSL) sobre análise de códigos em disciplinas de programação. A realização deste MSL buscou identificar os objetivos, as técnicas e os ambientes educacionais onde a análise de código é adotada. Além disso, buscou-se verificar se há um interesse crescente ou decrescente na pesquisa sobre este tema.*

1. Introdução

A programação é uma das competências fundamentais na área da computação. Contudo, apesar de sua importância, as disciplinas de programação têm apresentado um alto índice de reprovações e desistências [Duarte et al. 2010] [Yadin 2011]. Para o cenário apresentado em [Barbosa et al. 2014] o percentual de reprovações nas disciplinas de programação

DOI: 10.5753/cbie.sbie.2015.1235

é de aproximadamente 50%, cenários similares são encontrados em outras instituições, tal como citado em [Barbosa et al. 2011] e [Noschang et al. 2014]. Além disso, McCracken, em [McCracken et al. 2001], descreve que mesmo os alunos aprovados em disciplinas de programação não possuem as competências necessárias para o decorrer do curso, e consequentemente para vida profissional.

Diversos pesquisadores descrevem que a impossibilidade de um acompanhamento adequado por parte do professor para cada um dos alunos nas disciplinas de programação, é um dos fatores motivadores dos diversos problemas existentes neste contexto [Mota et al. 2009] [Stegeman et al. 2014]. O feedback rápido é de extrema importância para possibilitar o aprendizado de qualquer conceito [Stegeman et al. 2014].

Tradicionalmente em disciplinas de programação são adotadas atividades práticas de codificação. A análise das soluções propostas pelos alunos nestas atividades não é trivial, sendo essa avaliação demorada, sujeita ao viés e aos erros do avaliador. Para avaliar um programa, além de identificar se este é capaz de gerar as saídas adequadas para um conjunto de entrada, diversos parâmetros podem ser observados, tais como, a estrutura da código, a documentação associada, a eficiência e a manutenabilidade. Visto isso, o professor, mesmo quando assessorado por monitores, não consegue fornecer um feedback adequado e rápido para cada uma das soluções propostas pelos alunos.

Além disso, as atividades de avaliação em disciplinas de programação tem registrado um alto índice de ocorrência de plágio. Quando existem muitos alunos matriculados, ou quando uma grande quantidade de exercícios deve ser realizada na disciplina, é impossível para um avaliador humano realizar a comparação de todas as soluções e identificar quais destas foram plagiadas.

Com isso, pesquisas têm sido desenvolvidas com o intuito de propor métodos ou ferramentas para facilitar o acompanhamento das atividades dos discentes em disciplinas de programação. Muitas destas pesquisas propuseram o uso de mecanismos automáticos para análise de códigos, tendo como objetivos a redução da carga de trabalho do professor e a possibilidade de fornecer feedback de maneira mais ágil.

Neste trabalho é apresentado um mapeamento sistemático da literatura (MSL) sobre análise de códigos em disciplinas de programação. Este MSL buscou identificar os objetivos, as técnicas e os ambientes educacionais onde a análise é adotada, além disso buscou-se verificar se o interesse na pesquisa sobre este tema é crescente ou decrescente.

Este artigo está estruturado da seguinte maneira, na Sessão 2 são descritos os conceitos relacionados com a análise de códigos em contextos de ensino e aprendizagem de programação. Na sessão seguinte a metodologia adotada na execução do mapeamento é apresentada. Na Sessão 4 é exibida a análise dos dados relacionados aos trabalhos e são discutidos os resultados. Finalmente na última sessão, são exibidas as conclusões.

2. Analisadores de código em disciplinas de programação

Analisadores de código são utilizados em diversas atividades dentro da área de computação. Na engenharia de software, por exemplo, analisadores podem ser utilizados para extrair métricas que auxiliem na identificação de *bad smells* [Hozano et al. 2015] para realizar refatoração do código [Fowler 1999]. Compiladores utilizam diversas formas de análise de código na tradução de uma linguagem de programação para linguagem

de máquina. Contudo, para o contexto desta pesquisa os analisadores de código deveriam ser utilizados em disciplinas de programação, ou seja, a análise de código deveria gerar algum conjunto de dados com utilidade para o aluno ou o professor da disciplina.

Duas formas de análise pode ser realizadas, sendo estas a análise estática e a análise dinâmica. A análise estática envolve a observação do código, possivelmente com a extração de dados deste código, sem que o mesmo seja executado [Yulianto and Liem 2014]. A análise dinâmica requer a execução do código para realizar algum tipo de medição sobre a operação do programa [Yulianto and Liem 2014].

Análises de código úteis ao contexto de disciplinas de programação incluem análise de códigos que visem compreender os mecanismos de codificação adotados pelos alunos, a identificação de plágio nas soluções dos exercícios, o possibilidade de prover feedback sobre a solução especificada, entre outros.

A compreenssão dos mecanismos de codificação adotados pelos alunos, visa possibilitar ao professor planejar ações mais efetivas ao aprendizado dos discentes. Como exemplo, ao perceber que os discentes executam o programa diversas vezes com o intuito de imprimir o estado das variáveis ao longo da execução, o professor pode descrever o uso de mecanismos de debug como um modo de facilitar esta visualização.

Detectores de plágio tem como objetivo identificar trechos de código que tenham sido copiados a partir de outros alunos, ou de fontes externas [D Souza et al. 2007]. Uma vez que o plágio é uma atividade condenada por grande parte dos professores, pois é muitas vezes considerado como um ação que não favorece o aprendizado, é necessário primeiramente identificar os casos onde este ocorre para que medidas punitivas ou corretivas sejam adotadas. A identificação manual de plágio é praticamente impossível em turmas que possuam muitos alunos, ou onde muitos exercícios sejam solicitados, isso por que será necessário comparar cada solução com todas as outras, sendo este processo extremamente demorado e sujeito a diversos tipos de erros.

Avaliadores de códigos foram criados com o intuito de reduzir a carga de trabalho do professor, possibilitando ainda que as avaliações ocorram de forma mais uniforme. Araujo e colaboradores, em [Araujo et al.], descrevem que tais ferramentas geram avaliações com alta correlação com as avaliações fornecidas por professores/especialistas. Desta forma, tais soluções tem sido amplamente utilizadas em disciplinas de programação, tendo um alto índice de aceitação por parte de discentes e docentes.

Diversos outros usos de análise de códigos podem ser de grande auxílio na condução de disciplinas de programação. Este MSL visa identificar quais tem sido as principais aplicações de tais soluções, bem como quais técnicas tem sido utilizadas.

3. Metodologia adotada

Nesta sessão é apresentada a metodologia adotada na condução do MSL. Este mapeamento segue as diretrizes propostas em [Petersen et al. 2008], sendo o processo dividido em cinco etapas, sendo elas: (i) definição das questões de pesquisa (escopo da pesquisa), (ii) condução da pesquisa dos estudos primários (todos os artigos), (iii) triagem dos artigos usando critérios de inclusão e exclusão (artigos relevantes), (iv) keywording dos resumos, e (v) extração de dados e processo de mapeamento.

O objetivo desta pesquisa foi identificar para quais objetivos e quais técnicas são

empregadas na realização de análise de códigos em ambientes de ensino e aprendizagem de programação. Além disso, era desejado identificar o grau de interesse da comunidade nas pesquisas sobre o tema e as experiências práticas em relação ao uso destas soluções.

Embora a análise de código seja adotada para diversos fins, tais como, detecção de bad smells, atividades de perfilamento, otimização de compiladores, entre outros, neste mapeamento o foco de pesquisa foram analisadores de código relacionados com finalidades educacionais no contexto de programação introdutória.

Para a realização deste MSL não foi estipulado um período limite, pois algumas pesquisas relacionados aos tópicos de interesse datam das décadas de 60 ou 70. Desta forma, caso um período limite fosse adotado, diversas pesquisas dentro do escopo de interesse poderiam ter sido ignoradas.

3.1. Questões de pesquisa

Para condução do MSL devem ser definidas questões de pesquisa que orientarão o processo de seleção dos trabalhos. desta forma, o mapeamento conduzido neste trabalho visa a responder às questões de pesquisa:

QP1 Para quais objetivos a análise de código é adotada?
QP2 Quais técnicas são empregadas na análise de código?
QP3 Existe um interesse crescente ou descrecente na pesquisa sobre análise de código?
QP4 Em quais ambientes e níveis educacionais a análise de código tem sido utilizada?

3.2. Bases de dados utilizadas

Para realizar as buscas foram selecionadas fontes nacionais e internacionais. As fontes nacionais selecionadas correspondem aos anais dos eventos e periódicos que tratam do tema informática na educação ou educação em informática, para estes foram utilizados apenas aqueles que estão classificados no Qualis/CAPES. Entre as bases internacionais foram selecionadas aquelas que indexam os principais eventos e periódicos de computação. Foram conduzidas buscas automáticas para todas as fontes utilizadas, exceto para o Workshop sobre Educação em Computação (WEI), onde foi realizada busca manual.

As seguintes fontes nacionais foram utilizadas: Simpósio Brasileiro de Informática e Educação (SBIE), Workshop de Informática na Escola (WIE), Workshops do Congresso Brasileiro de Informática na Educação (WCBIE), Revista Brasileira de Informática na Educação (RBIE), Revista de Novas Tecnologias na Educação (RENOTE) e Workshop sobre Educação em Computação (WEI). As seguintes bases internacionais foram utilizadas: ACM Digital Library, IEEE Xplore, Elsevier Science Direct, Scopus, Springer e Web of Science

3.3. Processo de busca, critérios de inclusão e exclusão

O processo de busca adotado neste MSL tem início a partir da atividade de definição das strings de busca utilizadas. Para construir estas strings foram observados os termos mais comuns a um conjunto inicial de artigos já conhecidos pelos autores. Desde o processo de identificação inicial dos termos de busca, foram percebidas algumas dificuldades relacionadas ao estabelecimento de uma única string de busca para as pesquisas nas fontes nacionais e internacionais.

A primeira dificuldade observada se refere a termos que são particularmente utilizados nos textos em uma língua, não sendo estes comuns aos textos outra língua. Para ilustrar citamos o termo "novice programmer" (programador novato ou iniciante), utilizado em textos na língua inglesa e incomum aos textos em português, onde é substituído pelos termos aluno, estudante ou discente.

Além dos diferentes termos empregados, uma grande quantidade de sinônimos foi identificada para o contexto da busca. Para ilustrar citamos os sinônimos exercício/atividade/tarefa, código/algoritmo/programa. O uso ou a exclusão de um dos sinônimos da string de busca acarretava em uma grande quantidade de elementos retornados ou na exclusão de trabalhos relacionados ao tema de pesquisa.

Associada a dificuldade de uso de uma grande quantidade de termos na busca, foi observado que os mecanismos de busca das fontes nacionais possuíam limitação de tamanho da string. Nas bases internacionais apenas a IEEE Xplore teve restrição similar.

Visto isso, foi decidido que duas strings de busca seriam utilizadas, uma para a busca nas fontes nacionais e outra para a busca nas bases internacionais. As strings estabelecidas para a busca são exibidas nas tabelas a seguir.

Tabela 1. String de busca utilizada nas fontes nacionais

(algoritmo OR programa* OR codi*) AND ((avaliação OR feedback OR "análise de código" OR "comparação de código" OR "detecção de plágio" OR "juiz online") AND (exercício* OR atividade* OR "programador novato" OR estudante* OR aluno* OR discente*)

Os mecanismos de busca nas fontes nacionais permitem o uso do caractere * como caracter coringa, desta forma, por exemplo, o termo "programa*" pode se referir a "programa", "programas", "programador" ou "programação". Foram estabelecidos parenteses para maximizar a quantidade de trabalhos especificamente relacionados com o contexto de ensino e aprendizagem de programação.

Tabela 2. String de busca utilizada nas bases internacionais

(algorithms OR programming OR coding) AND ((automatic OR automated) AND (grading OR marking OR assessment OR feedback OR "code analysis"OR "code comparison") OR ("plagiarism detection"OR "online judge")) AND ("programming assignments"OR "novice programmer"OR "student programs"OR "student code")

Diferentemente dos mecanismos de busca para as fontes nacionais, os mecanismos de busca internacionais, em geral, não permitiam o uso de caracteres coringa. Desta forma, para tentar maximizar a quantidade de resultados foram utilizados os termos de busca no singular. O termo "student code"foi removido da string de busca, somente ao realizar a pesquisa no IEEE Xplore, devido a limitação no tamanho da string de busca imposta pelo mecanismo. Sempre que possível, para as bases internacionais, os resultados retornados foram filtrados para a área de ciência da computação.

Fazendo uso das strings de busca apresentadas foram retornados inicialmente 2.598 artigos, os resultados das buscas foram realizadas em cada uma das fontes e bases pesquisadas são apresentados na Tabela 3.

Tabela 3. Resultados retornados na seleção dos artigos em cada etapas do MSL.

Base de dados	Resultados iniciais	1ª Seleção (título e resumo)	2ª Seleção (intro. e conc.)	Seleção final (completa)
SBIE	27	8	8	8
WIE	10	0	0	0
WCBIE	6	3	2	2
RBIE	26	3	1	1
RENOTE	261	8	3	3
WEI (manual)	87	22	9	9
ACM Digital Lib.	810	134	88	67
IEEE Xplore	41	30	23	20
Science Direct	221	12	10	7
Scopus	498	94	34	25
Springer	116	6	3	2
Web of science	81	35	15	14
Total	2184	355	196	158

Para efetuar a seleção dos estudos primários recuperados foram adotados critérios de inclusão e exclusão. Os critérios de inclusão adotados foram:

- artigos que relatam o uso ou desenvolvimento de analisador de códigos;
- analisador de códigos para paradigmas de programação imperativo/estruturado;
- analisador de códigos relacionados com objetivo ou contexto educacional.

Os critérios de exclusão adotados foram:

- analisadores de códigos para outros temas relacionadas a programação (ex. engenharia de software, compiladores, outros);
- artigos em outros idiomas que não os idiomas português ou inglês;
- relatórios técnicos, notas de aula, slides, posters, position papers e estudos secundários (ou seja, revisões e mapeamentos sistemáticos da literatura);
- artigos duplicados.

Para aplicar os critérios de inclusão e exclusão especificados, os autores procederam com a distribuição dos artigos de cada base de pesquisa. Desta forma, cada autor realizou a análise dos artigos presentes em um determinada base. Essa distribuição também buscou uma divisão igualitária sobre o total de artigos que cada um deveria observar.

Na primeira etapa, todos os estudos primários recuperados foram avaliados através da leitura do título e resumo, visando identificar apenas aqueles que eram relevantes para o escopo adotado neste MSL. Caso existissem alguma dúvida sobre a adequação do trabalho ao MSL, foi recomendada a permanência deste trabalho para a próxima etapa de seleção. O conjunto inicial de estudos foi então reduzido para 355 artigos relacionados ao tema de interesse. Durante esta triagem, foram aplicados critérios de inclusão e exclusão para cada estudo recuperado, o único critério de exclusão não adotado foi o de remoção de artigos duplicados.

Em uma segunda etapa, foram efetuadas as leituras da introdução e das conclusões de cada um dos 355 artigos selecionados, mais uma vez os critérios de inclusão e exclusão

foram aplicados, desta vez os artigos duplicados foram removidos. Com isso, um subconjunto de 196 artigos permaneceu para a próxima etapa de seleção.

Finalmente na terceira etapa, os 196 artigos foram lidos de forma completa, mais uma vez os critérios de inclusão e exclusão foram aplicados. Com isso, um subconjunto de 158 artigos permaneceu para a extração de dados e processo de mapeamento. Em [Barbosa 2015] é exibida a lista final de artigos do MSL.

4. Análise

Nesta seção são apresentados os detalhes das análises realizadas com o intuito de responder cada uma das questões de pesquisa apresentadas. Durante o processo de resposta a cada uma das questões de pesquisa foram criados resumos sobre as informações coletadas durante o processo de classificação destes estudos e associados gráficos que facilitam a compreensão destas informações.

4.1. Resultados e discussão

Para responder a QP1, 'Para quais objetivos a análise de código é adotada?', cada uma dos 158 artigos foi lido de forma completa, buscando identificar objetivo da análise. Diversas categorias foram identificadas, sendo aqueles com maior representatividade citados a seguir:

- avaliadores automáticos, 108 publicações relacionadas ao tema;
- detectores de plágio, 50 publicações relacionadas ao tema;
- analisadores de similaridades, 10 publicações relacionadas ao tema.

Vale salientar que diversos trabalhos foram classificados em mais de uma categoria, por exemplo, 11 artigos apresentavam ferramentas para avaliar o código e detectar plágio entre as submissões dos alunos.

Observando os resultados é possível perceber que a comunidade acadêmica tem como foco de pesquisa os avaliadores automáticos e os detectores de plágio. Sendo desta forma respondida a QP1.

Para responder a QP2, 'Quais técnicas são empregadas na análise de código?', cada uma dos 158 artigos foi lido de forma completa. Para cada um destes buscou-se identificar a técnica utilizada. Uma vez que os avaliadores automáticos e os detectores de plágio representam a grande maioria das pesquisas, apenas as técnicas utilizadas nestas categorias serão citadas.

Para o contexto de avaliação automática a principal técnica empregada nas soluções é a análise dinâmica, neste contexto o teste de software. Dentre os 108 avaliadores automáticos identificados, 79 fazem uso desta técnica. A análise estática, realizada através da obtenção de métricas relacionadas aos códigos, é utilizada em 22 avaliadores automáticos. Outras abordagens propostas fazem uso de diversas técnicas, entre estas citamos, o uso de grafos na análise estrutural dos códigos, o emprego de técnicas inteligência artificial na comparação das soluções propostas com uma solução de referência, a utilização de árvores sintáticas, entre outros. Observando os resultados é possível perceber que os testes de software são a técnica mais utilizada na construção de avaliadores automáticos. Sendo desta forma respondida a QP2 para o conjunto de soluções voltada a avaliação automática.

Para o contexto de detecção de plágio, em comparação com a quantidade de técnicas citadas para avaliação automática, uma maior diversidade de técnicas é utilizada. Dentre os 50 detectores de plágio identificados, tokenização com cálculos de similaridade são adotadas em 15 trabalhos, a extração de métricas de similaridade é usada em 10 trabalhos e análise estrutural e sintática são adotados em 8 trabalhos cada. Observando os resultados é possível perceber que a tokenização é a técnica mais utilizada na construção de detectores de plágio. Sendo desta forma respondida a QP2 para o conjunto de soluções voltada a detecção de plágio.

Para responder a QP3, 'Existe um interesse crescente ou decrescente na pesquisa sobre análise de código?', cada artigo foi classificado de acordo com seu ano de publicação. Desta forma, é possível observar se a quantidade de publicações relacionadas ao tema tem crescido ao longo dos anos. Observando a Figura 1 é possível perceber que existe uma tendência crescente na quantidade de publicações. Apenas nos anos de 2006 e 2010 ocorreu uma redução na quantidade de publicações envolvendo o tema de interesse. A redução observada em 2015 não deve ser considerada, pois diversas publicações ainda serão realizadas ao longo do corrente ano. Sendo desta forma respondida a QP3, o interesse da comunidade de pesquisa sobre o tema de análise de códigos em ambientes de programação é crescente.

Figura 1. Quantidade de artigos relacionados com análise de código em ambientes de ensino e aprendizagem de programação publicados ao longo do período de 1964 até 2015.

Para responder a QP4, 'Em quais ambientes e níveis educacionais a análise de código tem sido utilizada?', foi observado em cada artigo a descrição do curso onde a análise foi aplicada, avaliada ou para qual foi construída. Dentre trabalhos selecionados, 19 artigos não forneceram descrições sobre o ambiente em que foram utilizados. A imensa maioria das soluções propostas são utilizadas no ensino superior, foram contabilizadas 125 pesquisas voltadas para estes ambientes. Apenas um trabalho teve como foco análise de código em escolas de ensino básico ou médio. Alguns trabalhos já investigam o uso de tais soluções em *Massive Open Online Courses* (*MOOCs*). Observando os resultados é possível perceber que o ambiente de ensino superior é aquele onde a maior parte das soluções de análise de código para o ensino e aprendizagem de programação tem sido pesquisada. Sendo desta forma respondida a QP4.

O processo completo do MSL, iniciado com a construção das strings de busca e finalizado com a especificação das respostas para cada QP, durou aproximadamente três

meses. A quantificação da duração de cada etapa é complexa, pois muitas atividades foram executadas em paralelo por cada um dos autores.

4.2. Ameaças à validade

A condução do MSL visa um processo de seleção imparcial, desta forma as questões de pesquisa e os critérios de inclusão e exclusão foram estabelecidos antes do início das buscas. Contudo, existem ameaças à validade presentes neste MSL, sendo estas:

- Variação na interpretação dos critérios de inclusão e exclusão para seleção de artigos - a condução do processo de seleção dos estudos foi realizada em paralelo pelos autores. Desta forma, embora tenham sido realizados diversas discussões sobre as dúvidas que surgiram na seleção, alguma variação na interpretação dos critérios de inclusão e exclusão pode ter ocorrido;
- Indisponibilidade para download - durante o processo de busca, muitos artigos encontrados com a aplicação das strings de busca não estavam disponíveis para download. Desta forma, visto que alguns destes artigos poderiam estar no conjunto de artigos da seleção final, os resultados apresentados neste MSL poderiam ser ligeiramente diferentes;
- Falta de detalhamento de informações nos artigos - alguns dos trabalhos selecionados não mencionavam informações relevantes para pesquisa, como por exemplo, a técnica empregada na análise ou o ambiente em que o analisador é utilizado. Desta forma, os resultados gerados para responder as questões de pesquisa poderiam ser modificados, caso tais informações fossem conhecidas;
- Ameaça de constructo relacionada a string de busca - devido a grande quantidade de termos e sinônimos que poderiam ser utilizados nos artigos de interesse, e associado a isso, dada a limitação de tamanho da string de busca imposta por alguns mecanismos de pesquisa, é possível que termos que influenciariam a pesquisa não tenham sido empregados. Além disso, um conjunto limitado de base de dados foi utilizado, portanto é possível que estudos relevantes não tenham sido incluídos.

5. Conclusões

No presente artigo foi apresentado um MSL sobre o contexto de analisadores de código para ambientes de ensino e aprendizagem de programação. O principal objetivo desse mapeamento foi proporcionar uma visão geral do que tem sido investigado no contexto da análise de código aplicada em disciplinas relacionadas a programação.

Para cumprir esse objetivo foram definidas quatro questões de pesquisa a serem respondidas pelo mapeamento. QP1: Para quais objetivos a análise de código é adotada? QP2: Quais técnicas são empregadas na análise de código? QP3: Existe um interesse crescente ou descrecente na pesquisa sobre análise de código? QP4: Em quais ambientes e níveis educacionais a análise de código tem sido utilizada? Todas as questões de pesquisa foram respondidas neste MSL.

A análise de código, para o escopo de interesse da pesquisa, tem sido utilizada para possibilitar a avaliação automática de códigos e a detecção de plágio em exercícios de programação. A principal técnica utilizada na avaliação automática de códigos é o emprego de testes de software para comparar as saídas geradas. A principal técnica utilizada na detecção de plágio é a tokenização para comparação de similaridade entre dois códigos. A ampla maioria das pesquisas se concentra no ensino superior, e existe um interesse crescente na comunidade acadêmica na pesquisa de tais soluções.

6. Agradecimentos

Agradecemos o suporte financeiro, em forma de Bolsas de Desenvolvimento Acadêmico e Institucional (BDAI), fornecido pela Universidade Federal de Alagoas (UFAL).

Referências

Araujo, E. C., Gaudencio, M., Menezes, A., Ferreira, I., Ribeiro, I., Fagner, A., Ponciano, L., Morais, F., Guerrero, D. S., and Figueiredo, J. A. O papel do hábito de estudo no desempenho do aluno de programação.

Barbosa, A. (2015). Listagem de artigos. `sites.google.com/site/ufalalexandre/dadosdeartigos`. Último acesso em Julho de 2015.

Barbosa, A. d. A., Ferreira, D. Í., and Costa, E. B. (2014). Influência da linguagem no ensino introdutório de programação. In *Anais do CBIE/SBIE*, pages 612–621.

Barbosa, L. S., Fernandes, T. C., and Campos, A. M. (2011). Takkou: Uma ferramenta proposta ao ensino de algoritmos. In *Anais do CSBC/WEI*.

D Souza, D., Hamilton, M., and Harris, M. C. (2007). Software development marketplaces: Implications for plagiarism. In *Proc. of the 9th Australasian Conf. on Computing Education*, pages 27–33, Darlinghurst, Australia.

Duarte, A. N., Moreira, H., and Mello, T. S. (2010). Competitividade como fator motivacional para o estudo de computação. In *Anais SBIE*, João Pessoa, PB, Brasil.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Hozano, M., Ferreira, H., Fonseca, B., and de Barros Costa, E. (2015). Using developers feedback to improve code smell detection. In *Anais do ACM SAC*, Salamanca, Espanha.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working Group Reports from ITiCSE*, pages 125–180, New York, USA.

Mota, M. P., de Brito, S. R., Moreira, M. P., and Favero, E. L. (2009). Ambiente integrado a plataforma moodle para apoio ao desenvolvimento das habilidades iniciais de programação. In *Anais do SBIE*, Florianópolis, SC, Brasil.

Noschang, L. F., Fillipi Pelz, E. A., and Raabe, A. L. (2014). Portugol studio: Uma ide para iniciantes em programação. In *Anais do CSBC/WEI*, pages 535–545.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proc. of the 12th Int. Conf. on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK.

Stegeman, M., Barendsen, E., and Smetsers, S. (2014). Towards an empirically validated model for assessment of code quality. In *Proc. of the 14th Koli Calling Int. Conf. on Computing Education Research*, Koli Calling, pages 99–108, New York, USA.

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads*, 2(4):71–76.

Yulianto, S. and Liem, I. (2014). Automatic grader for programming assignment using source code analyzer. In *Int. Conf. on Data and Software Engineering (ICODSE)*.