Brazilian Journal of Development

Análise de aprendizagem a partir de códigos-fontes euma proposta de seleção automática de métricas de avaliação

Analysis of learning from source-codes and a proposal for automatic selection of evaluation metrics

Recebimento dos originais: 04/11/2018
Aceitação para publicação: 06/12/2018

Márcia Gonçalves de Oliveira
Doutora em Engenharia Elétrica pela Universidade Federal do Espírito Santo
Instituição: Centro de Referência em Formação e Educação a Distância (Cefor/Ifes)
Endereço: Rua - Bairro, Cidade – PR, Brasil
E-mail: marciaoliveira@ifes.edu.br

Ádler Oliveira Silva Neves
Bacharel em Sistemas de Informação pelo Instituto Federal do Espírito Santo
Instituição: Instituto Federal do Espírito Santo
Endereço: ES-010, Km-6,5 - Manguinhos, Serra - ES, Brasil
E-mail: adlerosn@gmail.com

Mônica Ferreira Silva Lopes
Estudante do Curso de Sistema de Informação do Ifes – Campus Serra
Instituição: Instituto Federal do Espírito Santo
Endereço: ES-010, Km-6,5 - Manguinhos, Serra - ES, 29173-087
E-mail: ferreirasilvalopes@gmail.com

Andreângelo da Paixão Patuzzo
Estudante de Sistemas de Informação
Instituição: Instituto Federal do Espírito Santo
Endereço: ES-010, Km 6,5 - Manguinhos, Serra - ES, Brasil
E-mail: andreangelo.pp@gmail.com

RESUMO

Este trabalho apresenta uma revisão de soluções automatizadas de análise da aprendizagem de programação a partir de códigos-fontes. Estendendo as propostas apresentadas nesse estudo, propomos uma estratégia tecnológica de análise quantitativa da aprendizagem de programação a partir de informações de códigos-fontes mapeadas em um cluster de métricas de software correlacionadas acima de um índice. A contribuição desta proposta para a aprendizagem de programação é auxiliar o trabalho de avaliação de professores de programação para que melhor compreendam as dificuldades de aprendizagem de seus alunos e reconheçam, a partir de um conjunto representativo de variáveis, as dificuldades de aprendizagem e habilidades de programação.

Palavras-chaves: Códigos de Programação, seleção de métricas, análise de aprendizagem

Brazilian Journal of Development

ABSTRACT

This paper presents a review of automated programming learning analysis solutions from source codes. Extending the proposals presented in this study, we propose a technological strategy for quantitative analysis of programming learning from information from source codes mapped in a cluster of correlated software metrics above an index. The main contribution of this proposal to Informatics Education is to help the evaluation of programming teachers so that they better understand the learning difficulties of their students and recognize, from a representative set of variables, difficulties and skills programming.

Keywords: Programming codes, selection of metrics, Learning Analysis

1 INTRODUÇÃO

O desenvolvimento de estratégias tecnológicas de apoio à avaliação que identifiquem dificuldades de aprendizagem e habilidades de programação a partir de informações de códigos-fontes escritos por alunos tem sido um verdadeiro desafio para a informática na educação e poucas soluções têm sido de fato desenvolvidas para esse fim.

Contemplando esse desafio, buscamos através deste trabalho responder às seguintes questões de pesquisa: existem possibilidades de avaliação em que, em vez de aplicar testes, analisem-se códigos de programação e, a partir deles, identifiquem-se dificuldades de aprendizagem, habilidades e até competências em programação? É possível ter um conjunto representativo desses indicadores para avaliação da aprendizagem?

Para responder a essas questões, este trabalho apresenta uma revisão de literatura de estratégias tecnológicas de avaliações da aprendizagem baseadas na análise de códigos-fontes, propomos e experimentamos uma estratégia de seleção automática de indicadores de aprendizagem a partir de 348 métricas de software do sistema de análise de aprendizagem *Pcodigo II* [Neves et al. 2017a, Neves et al. 2017b].

A contribuição desta pesquisa para a Informática na Educação é promover a discussão e incentivar o desenvolvimento de tecnologias para uma avaliação fina, clínica e multidimensional da aprendizagem de programação [Raphael and Carrara 2002]. Espera- se, portanto, que em um futuro próximo, essas tecnologias possam ser instrumentos de trabalho de professores auxiliando-os na compreensão das dificuldades de aprendizagem de seus alunos e na tomada de decisões de avaliação que reorientem as ações de ensino em favor da aprendizagem.

Para apresentar algumas tecnologias de análise da aprendizagem a partir de códigos-fontes e propor uma estratégia de seleção automática de variáveis de avaliação para a análise de aprendizagem de programação, este trabalho foi organizado conforme a ordem a seguir. Na Seção 2, apresentamos as estratégias de avaliação de programação baseadas em análise estática e como elas têm sido utilizadas na avaliação da aprendizagem de programação. Na Seção 3, propomos uma

sequência de ações para análise de aprendizagem de programação aplicando a abordagem de análise estática. Na Seção 4, a partir dessa proposta de análise de aprendizagem, desenvolvemos uma estratégia de sele- ção automática de métricas de software por clusterização em grafo para gerar indicadores de avaliação de programação. Na Seção 5, concluimos com as considerações finais.

2 ANÁLISE ESTÁTICA DA APRENDIZAGEM DE PROGRAMAÇÃO

A análise estática é uma abordagem de avaliação automática da aprendizagem de programação baseada na análise de códigos-fontes. Através da análise estática, é possível ana- lisar esforço, complexidade, eficiência e qualidade de programação [Curtis et al. 1979b, Berry and Meekings 1985, Rahman et al. 2008].

De acordo com [Oliveira and Oliveira 2015], as principais vantagens da análise estática são o menor custo, a menor dependência de um gabarito e a possibilidade de oferecer uma avaliação mais próxima da avaliação humana, embora muitos professores de programação priorizem a avaliação dinâmica, que é uma avaliação baseada na execução correta e eficiente de programas. A análise estática pode, portanto, ser utilizada na análise de códigos de programação para os seguintes propósitos:

- Medição de esforço e complexidade de codificação [Curtis et al. 1979b]
- Predição de desempenhos [Xu and Chee 2003, Naude et al. 2010]
- Análise de estilo de programação [Rees 1982, Berry and Meekings 1985]
- Avaliação por métricas de software [Hung et al. 1993, Neves et al. 2017b]
- Reconhecimento de indícios de plágios [Neves et al. 2017b]
- Reconhecimento de rubricas [Oliveira et al. 2018]
- Recomendação de Atividades [Oliveira et al. 2013]
- Análise de proficiência em programação [Kumar et al. 2014]
- Análise de dificuldades de aprendizagem [Neves et al. 2017b]

Entre as soluções tecnológicas de análise de aprendizagem de programação baseadas em análise estática já propostas, destacamos: as métricas de *Halstead* e *McCabe* [Halstead 1977, Curtis et al. 1979b, Berry and Meekings 1985], a avaliação de habilidades de programação por métricas de software de [Hung et al. 1993], o sistema de recomendação de atividades conforme dificuldades de aprendizagem de [Oliveira et al. 2013], a análise de dificuldades por métricas de software de [Oliveira et al. 2017b, Neves et al. 2017b], a avaliação de como estudantes de programa- ção aprendem a partir da análise de seus códigos de programação [Blikstein et al. 2014] e a análise de proficiência em programação realizada pelo sistema SCALE de [Kumar et al. 2014].

2.1 A EVOLUÇÃO DAS ESTRATÉGIAS DE ANÁLISE DE APRENDIZAGEM DE PROGRAMAÇÃO

As tabelas 1, 2 e 3 apresentam exemplos de estratégias de análise de aprendizagem baseadas na análise de códigos-fontes e como elas evoluíram desde a década de 70 até os dias atuais. Essas estratégias se destacam principalmente pelos objetivos de identificar dificuldades de aprendizagem, habilidades e até competências.

Na Tabela 1, apresentamos três exemplos de trabalhos que utilizaram métricas de software para analisar códigos e avaliar esforço, complexidade e estilo de programação. Esses trabalhos associam o processo de programação à complexidade psicológica para avaliar performance em programação sem necessariamente ter a preocupação de melhorar a aprendizagem daqueles que tinham mais dificuldades.

Avaliação da aprendizagem de programação por análise de c ódigo - 1970 – 1990

Propostas	Estratégias	Características de Avaliação
[Curtis et al. 1979b]	Métricas de Software de *McCabe* e de *Halstead*	As métricas são utilizadas para medir complexidade psicológica e para predizer desempenhos. O estudo evidencia as dificuldades de programadores em compreender e modificar software.
[Curtis et al. 1979a]	Métricas de Halstead, Métricas de McCabe, Linhas de código	Demonstra-se que as métricas de Halstead são os me- lhores preditores de performance seguidos de MCCabe e linhas de código.
[Rees 1982]	Métricas de Estilo	Avalia estilo de programação por contagens de atributos de código-fonte.

Tabela 1. Estratégias de avaliação por análise de código - 1970 a 1990

A Tabela 2 apresenta exemplos de trabalhos dos anos 90 até o ano de 2010. Nesses trabalhos, ainda utilizam-se métricas para análise de aprendizagem, mas em tempos em que estavam em alta os Sistemas Tutores Inteligentes (STI), buscava-se representar o modelo ou perfil do aluno atentando-se mais para a aprendizagem.

Na Tabela 3, destacamos os trabalhos mais recentes sobre análise da aprendizagem de programação. Em muitos desses trabalhos, além de atentar-se para a aprendizagem dos alunos, buscava-se remediar as dificuldades de aprendizagem [Oliveira et al. 2013]. Outras tendências são

Brazilian Journal of Development

a avaliação de proficiência em programação [Kumar et al. 2014], a previsão de desempenhos [Oliveira 2016] e a classificação de perfis por níveis de aprendizagem [Novais et al. 2016].

3 AÇÕES PARA ANÁLISE DE APRENDIZAGEM A PARTIR DE CÓDIGOS-FONTES

Uma proposta de sequência de ações para análise quantitativa de aprendizagem a partir de códigos-fontes consiste nos seguintes passos:

Análise da aprendizagem de programação por análise de código - 1990 - 2010

Propostas	Estratégia	Características de Avaliação
[Hung et al. 1993]	Métricas de Avaliação Automática	Avalia habilidades de programar, complexidade, estilo de programação e eficiência
[Benford et al. 1995] - *Ceilidh*	Métricas de Análise Dinâmica e Estática	Analisa características de código-fonte, corretude funcional e eficiência. São computadas estatísticas como, por exemplo, média de caracteres por linha, média de espaços por linha, número de comentários e variáveis globais. Na complexidade estrutural são avaliados itens como número de palavras reservadas, número de estruturas de controle condicional e de repetição e número de chamadas de funções.
[Mengel and Ulans 1998]	Analisador estático LOGISCOPE	Possui várias métricas de análise de programas e avalia complexidade e qualidade de programação.
[Truong et al. 2004]	Análise de Similaridade estrutural e Métricas da Engenharia de Software	Avalia qualidade da solução e similaridade estrutural entre um programa Java e a solução modelo. A estratégia usa métricas de engenharia de software e comparação relativa para avaliar a qualidade dos programas de alunos e indicar como as soluções podem ser melhoradas.
[Wu et al. 2007] - AnalyseC	Métricas de software, otimização de compilador e técnicas de visualização	Análise de programas em Linguagem C em nível estrutural e semântico
[Binkley 2007]	Analisadores de código	O estudo não é sobre avaliação de aprendizagem, mas aponta muitos métodos e aplicações da análise de códigos-fontes na avaliação da aprendizagem.
[Gerdes et al. 2010]	Estratégias de padronização de código	Analisa corretude e boas práticas de programação

Tabela 2. Estratégias de avaliação por análise de código - 1990 a 2010

- Coletar via submissão de exercícios por uma interface *web* códigos de programação [Neves et al. 2017a].
- Estabelecer mecanismos de execução em massa e segura dos códigos de programação submetidos para avaliação [Oliveira et al. 2015].
- Estabelecer um amplo conjunto de métricas de análise de códigos-fontes para quantificar esforço, estilo e eficiência de programação [Neves et al. 2017b].
- Para cada solução de programação, criar uma representação vetorial em que cada dimensão é o valor de uma métrica de análise de código [Neves et al. 2017b].
- Gerar uma matriz cognitiva *M* para cada exercício de programação reunindo os vetores das soluções em métricas de software [Neves et al. 2017b].
- Remover da matriz *M* as métricas que estão zeradas em todas as dimensões.

- Gerar uma matriz cognitiva M^j selecionando apenas as colunas das métricas mais relevantes para a avaliação de uma atividade.
- Para cada matriz cognitiva M^j, identicar classes de soluções e as métricas que mais as caracterizam.
- Gerar visualizações gráficas multidimensionais da matriz M^j para informar dificuldades, habilidades e competências em programação [Oliveira et al. 2017a].
- Sintetizar métricas relacionadas em fatores para obter indicadores de classificação de aprendizes como noviços, experientes e proficientes em programação.
- Gerar relatórios de avaliação diagnóstica [Oliveira et al. 2017a, Neves et al. 2017a].
- Criar um mecanismo de análise de aprendizagem baseado em sistema de recomendação de filtragem colaborativa em que a nota de uma questão é predita a partir dos perfis mapeados em métricas de um histórico de aprendizagem.

Um sistema desenvolvido recentemente para a análise multidimensional e quantitativa da aprendizagem de programação a partir de códigos-fontes por 348 métricas de software é o *PCodigo II* [Neves et al. 2017a].

O PCodigo II estende PCodigo original de [Oliveira et al. 2015] na representação de perfis de estudantes pelas métricas de análise de códigos-fontes em Linguagem C [Berry and Meekings 1985], pelas métricas de Halstead e McCabe [Curtis et al. 1979b] e pelos indicadores de execução compila e executa de [Oliveira et al. 2015]. Ao todo, o PCodigo II implementa 348 métricas de representação de códigos-fontes para quantificar esforço e qualidade de programação [Neves et al. 2017b].

Embora o PCodigo II apresente um amplo leque de variáveis de avaliação da aprendizagem de programação, para um professor nem sempre é fácil visualizar e compreender o que todas essas variáveis informam sobre a aprendizagem de programação dos alunos. Além disso, para realizar análises estatísticas e previsões de desempenhos, torna-se necessário um número bem menor de variáveis.

Dessa forma, estendendo a proposta do PCodigo II, propomos uma estratégia de seleção automática de métricas de software mais relevantes para auxiliar professores em suas tomadas de decisões de avaliação. Essa estratégia contempla as atividades 6, 7 e 8 da sequência de ações de análise da aprendizagem de programação proposta.

Análise da aprendizagem de programação por análise de código - 2011 - 2017

Propostas	Estratégia	Características de Análise
[Oliveira et al. 2013]	Algoritmos e métricas de avaliação de classificação *multilabel*	Representa vetorialmente perfis de estudantes por componentes de habilidades mapeadas em contagens de *tokens*, símbolos e operadores presentes em códigos de programação escritos em Linguagem C. A re- comendação de atividades é uma estratégia de avalia- ção formativa que sugere atividades conforme difícul- dades de aprendizagem reconhecidas nas representa- ções de perfis dos estudantes. As métricas de avali- ação de classificação *multi-label* também são utiliza- das como instrumentos de avaliação diagnóstica para identificação de habilidades e dificuldades em progra- mação.
[Blikstein et al. 2014]	Métodos de Aprendizagem de Máquina e métrica baseada em processo	Tem como objetivos descobrir padrões em códigos de programação, mapear processos e trajetórias de aprendizagem, identificar situações de produtividade e improdutividade e prever desempenhos.
[Kumar et al. 2014]	Framework chamado SCALE (*Smart Competence Analytics in LEarning - Análise inteligente de competências na aprendizagem*)	O SCALE possui camadas de detecção, análise, com- petência e visualização. A camada de detecção visa capturar dados para consciência de contexto. A camada de análise obtém métricas de desempenho pela análise de códigos-fontes. A camada de competência identifica níveis de competências em programação. A camada de visualização permite a interação entre mé- tricas de desempenho e de competência.
[Oliveira 2016]	Análise de aprendizagem preditiva	Apresenta uma revisão de literatura das tecnologias de análise de aprendizagem aplicadas no domínio da programação e em outros domínios e propõe um *framework* para previsão de desempenhos em avaliação somativa a partir de um histórico de códigos são ana- lisados e mapeados em componentes de habilidades.
[Novais et al. 2016]	Reconhecimento de perfis por Métricas de análise estática de códigos	Tem como objetivos inferir perfis de programado- res a partir da análise de seus códigos em Java, classificá-los conforme habilidades e avaliar continu- amente seus progressos na prática da programação em um curso. Os perfis detectados são noviço, iniciante avançado, proficiente e especialista. Algumas das mé- tricas utilizadas são: número de sentenças, de estrutu- ras de controle condicional e de repetição, de tipos de dados, de classes, de operadores, de linhas de código e outras informações de código
[Neves et al. 2017b]	Combinação de Métricas de Software	Reúne um conjunto de 348 métricas de sofware [Curtis et al. 1979b, Berry and Meekings 1985, Oliveira et al. 2015] para avaliação diagnóstica da aprendizagem de programação visando quantificar informações de esforço e qualidade de programação para reconhecimento automático de dificuldades de aprendizagem de programação e análise de plágios.

Tabela 3. Estratégias de avaliação por análise de código - 2011 a 2017

4 SELEÇÃO DE MÉTRICAS POR CLUSTERIZAÇÃO DE CARACTERÍSTICAS CORRELACIONADAS

Uma proposta inicial para selecionar um conjunto mais representativo das 348 métricas do *PCodigo II* é utilizar um algoritmo de clusterização em grafo [Schaeffer 2007] para formar dois *clusters* de métricas de software mapeadas no número de alunos de uma turma de programação. A ideia é separar no primeiro *cluster*, chamado *Cluster -1*, métricas que têm baixa ou nenhuma correlação com outras métricas e, no segundo *cluster*, chamado de *Cluster 0*, as métricas mais correlacionadas. Dessa forma, ficariam no *Cluster -1* as métricas que não aparecem ou aparecem

poucas vezes nos códigos de soluções de um exercício de programação. Já as métricas que aparecessem mais nos códigos de soluções de um exercício seriam agrupadas no *Cluster 0*.

Para realizar esse experimento, utilizamos o *PCodigo II* para criar uma matriz cognitiva *M* , cujas linhas são 100 alunos de uma turma de programação e as colunas, as 348 métricas de softwares normalizadas em valores de 0 a 1 para cada aluno. Para cada métrica, essa normalização foi realizada dividindo-se o valor de métrica de cada aluno pelo maior valor da métrica obtido por um aluno.

Para realizar a clusterização de métricas, transpomos a matriz *M* em uma matriz M^j, cujas linhas são as métricas de software e as colunas, os alunos. A clusterização foi realizada pelo software *Cluto 2.1.2*.

Os parâmetros da clusterização foram configurados da seguinte forma: *method=graph; sim=corr; vtxprune=0.3*. *Method* é o algoritmo de clusterização em grafo. O índice de similaridade de *sim=corr* é o coeficiente de correlação. O parâmetro *vtxprune=0.3* indica que as métricas com correlação abaixo de 0.3 serão inseridas no *Cluster -1* e as com correlação acima de 0.3, no *Cluster 0*.

A Figura 1 é uma representação das matrizes transpostas do *Cluster 0 (A)* com as métricas selecionadas e do *Cluster -1 (B)* com as métricas não selecionadas e métricas para análise de relevância.

Nos gráficos da Figura 1, as linhas são as notas de 100 alunos e as colunas, as métricas de software. Quanto mais vermelho for o valor de uma métrica maior é o seu valor para um aluno. A cor branca indica valores iguais a zero nas métricas.

Nesse primeiro experimento reduzimos o número de métricas em 78.7%, isto é, de 348 para 74 métricas. Essa redução foi realizada extraindo as métricas com valores iguais a 0 em todas as 100 soluções de programação e as métricas com correlação abaixo de 0.3 com outras métricas.

No Gráfico (A) da Figura 1, que é o *Cluster 0*, observamos pela predominância da cor vermelha que as métricas selecionadas aparecem praticamente em todas as soluções desenvolvidas pelos alunos, o que indica que a maior parte dessas métricas são relevantes para comparar as soluções dos alunos. Entre essas métricas relevantes destacamos as métricas de *Halstead*, de complexidade ciclomática, de contagem de operadores relacionais (de comparação), o número de linhas de código e uso do *if* e do *for*.

Nesse caso, para finalizar o processo de redução e de seleção de métricas de análise de aprendizagem, é necessária uma tomada de decisão do professor. No exemplo da Figura 1 (C), o professor considerou importantes apenas as duas primeiras métricas, que indicam se um programa compila e se executa, tendo essas métricas apenas dois valores: *0 (Falso)* ou *1 (Verdadeiro)*.

Brazilian Journal of Development

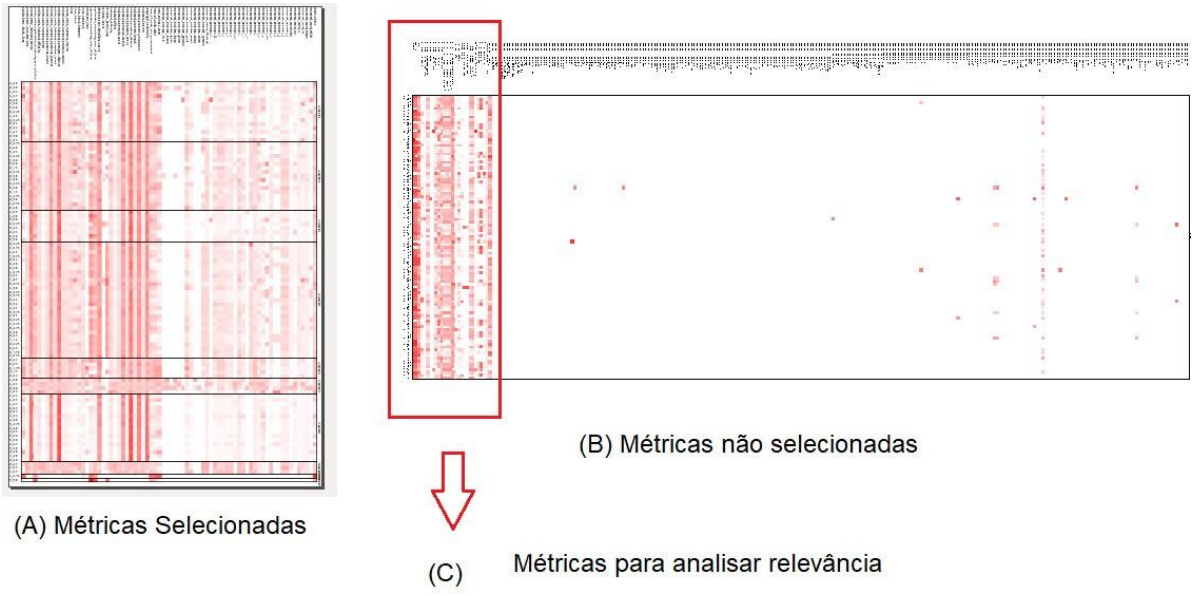

Figura 1. Seleção de Métricas de Software

A Figura 2 apresenta os resultados da redução de características por clusterização em grafo do *Cluster 0* com as métricas selecionadas. Uma importante observação nos alunos que descrevem o *cluster 0*, identificados por *classe_nota*, (em *Descriptive*) representam as notas das classes 3 (de 2.5 a 3.4) a 5 (de 4.5 a 5.4) em uma escala de 0 a 5. Isso indica que o *Cluster 0* contém uma representação das melhores soluções de programação para um exercício, o que reforça que as métricas selecionadas são os indicadores mais prováveis para a avaliação desse exercício conforme os resultados de seleção automática.

```
1-way clustering: [Cut=7.69e+02] [74 of 348]
------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  |
------------------------------------------------------------------------
  0    74 +0.285 +0.147  +nan  +nan |
------------------------------------------------------------------------
------------------------------------------------------------------------------
1-way clustering solution - Descriptive & Discriminating Features...
------------------------------------------------------------------------------
Cluster   0, Size:    74, ISim: 0.470, ESim: -inf
      Descriptive:  3_3.0 24.1%, 4_3.5 12.2%, 4_3.7 11.0%, 3_2.7  7.2%, 4_4.0  6.6%, 4_3.5  6.0%, 4_3.75  2.9%, 5_5.0  2.2%, 5_5.0
1.2%, 4_3.5  1.2%
```

Figura 2. Seleção representativa da diversidade de notas

Os resultados desse primeiro experimento mostram, portanto, que podemos automaticamente selecionar uma melhor representação das métricas de software para ser utilizada nas decisões de avaliação de professores. Mas, embora tenhamos uma redução de quase 80% das métricas, um novo passo pode ser dado para redução de dimensionalidade, que é a síntese de

métricas relacionadas em fatores através da análise fatorial. Recomendamos esse passo para um trabalho futuro.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou um resumo do estado da arte de soluções de análise da aprendizagem de programação por análise de códigos-fontes e os desafios a serem vencidos. Além disso, propomos um *framework* para análise da aprendizagem da programação por métricas de software [Curtis et al. 1979b, Berry and Meekings 1985, Oliveira et al. 2015, Neves et al. 2017b] e uma estratégia de seleção automática de métricas de software para melhor representar o que um professor quer avaliar.

Como trabalhos futuros a partir deste, sugerimos estender a proposta deste trabalho aplicando a extração de características por meio da técnica estatística de análise fatorial para melhor redução de dimensionalidade.

Nossas expectativas a partir deste estudo são, portanto, incentivar o desenvolvimento de novas estratégias de seleção e de extração de características de forma a ampliar as possibilidades de análise de aprendizagem de programação para reconhecimento auto- mático mais preciso de dificuldades, habilidades e competências em programação.

REFERÊNCIAS

Benford, S. D., Burke, E. K., Foxley, E., and Higgins, C. A. (1995). The Ceilidh system for the automatic grading of students on programming courses. In *Proceedings of the 33rd annual on Southeast regional conference*, ACM-SE 33, pages 176–182, New York, NY, USA. ACM.

Berry, R. E. and Meekings, B. A. (1985). A style analysis of c programs. C*ommunications of the ACM*, 28(1):80–88.

Binkley, D. (2007). Source code analysis: A road map. In *Future of Software Engineering, 2007. FOSE'07*, pages 104–119. IEEE.

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., and Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4):561–599.

Curtis, B., Sheppard, S. B., and Milliman, P. (1979a). Third time charm: Stronger predic- tion of programmer performance by software complexity metrics. In *Proceedings of the 4th International Conference on Software Engineering*, ICSE '79, pages 356–360, Piscataway, NJ, USA. IEEE Press.

Curtis, B., Sheppard, S. B., Milliman, P., Borst, M., and Love, T. (1979b). Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Transactions on software engineering*, (2):96–104.

Gerdes, A., Jeuring, J. T., and Heeren, B. J. (2010). Using strategies for assessment of programming exercises. In *Proceedings of the 41st ACM technical symposium on Com- puter science education*, SIGCSE '10, pages 441–445, New York, NY, USA. ACM.

Halstead, M. H. (1977). *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, USA.

Hung, S.-l., Kwok, L.-f., and Chung, A. (1993). New metrics for automated programming assessment. In *Proceedings of the IFIP WG3.4/SEARCC (SRIG on Education and Training) Working Conference on Software Engineering Education*, pages 233–243, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

Kumar, V., Boulanger, D., Seanosky, J., Panneerselvam, K., Somasundaram, T. S., et al. (2014). Competence analytics. *Journal of Computers in Education*, 1(4):251–270.

Mengel, S. A. and Ulans, J. (1998). Using verilog logiscope to analyze student programs. In *Frontiers in Education Conference, 1998. FIE'98. 28th Annual*, volume 3, pages 1213–1218. IEEE.
Naude, K. A., Greyling, J. H., and Vogts, D. (2010). Marking student programs using graph similarity. *Computers & Education*, 54(2):545 – 561.

Neves, A., Oliveira, M., França, H., Lopes, M., Reblin, L., and Oliveira, E. (2017a). Pcodigo II: O sistema de diagnóstico da aprendizagem de programação por métricas de software. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educa- ção*, volume 6, page 339.

Neves, A., Reblin, L., França, H., Lopes, M., Oliveira, M., and Oliveira, E. (2017b). Mapeamento automático de perfis de estudantes em métricas de software para análise de aprendizagem de

programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 28, page 1337.

Novais, D., Pereira, M. J., and Henriques, P. R. (2016). Profile detection through source code static analysis. In *5th Symposium on Languages, Applications and Technologies (SLATE'16)*, volume 51, pages 1–13. OASICS.

Oliveira, M. (2016). As tecnologias de análise de aprendizagem e os desafios de prever desempenhos de estudantes de programação. In *XI^{O} DesafIE – Workshop de Desafios da Computação Aplicada à Educação*, Porto Alegre - RS. CSBC 2016.

Oliveira, M., França, H., Neves, A., Lopes, M., and Silva, A. C. (2017a). Instrumentos de visualização da informação para avaliação diagnóstica em curso de programação a distância. In *Anais do Workshop de Informática na Escola*, volume 23, page 452.

Oliveira, M., Neves, A., Lopes, M. F. S., Medeiros, H., Andrade, M. B., and Reblin, L. (2017b). Um curso de programação a distância com metodologias ativas e análise de aprendizagem por métricas de software. *RENOTE*, 15(1).

Oliveira, M., Nogueira, Araújo, M., and Oliveira, E. (2015). Sistema de apoio à prática assistida de programação por execução em massa e análise de programas. In *CSBC 2015-Workshop de Educação em Informática (WEI), Recife-PE*.

Oliveira, M., Souza, M., Reblin, L., and Oliveira, E. (2018). Reconhecimento automático de representações de rubricas em agrupamentos de soluções de exercícios de programação. *Revista Brasileira de Informática na Educação*, 26(2).

Oliveira, M. G., Ciarelli, P. M., and Oliveira, E. (2013). Recommendation of program- ming activities by multi-label classification for a formative assessment of students. *Expert Systems with Applications*, 40(16):6641–6651.

Oliveira, M. G. and Oliveira, E. (2015). Abordagens, práticas e desafios da avaliação automática de exerc'icios de programação. In *4^{O} DesafIE – Workshop de Desafios da Computação Aplicada à Educação*, Recife. CSBC 2015.

Rahman, K. A., Ahmad, S., Nordin, M. J., and Maklumat, F. T. D. S. (2008). The design of an automated c programming assessment using pseudo-code comparison technique.

Raphael, H. and Carrara, K. (2002). *Avaliação sob exame*. Autores Associados, Campinas.

Rees, M. J. (1982). Automatic assessment aids for pascal programs. *SIGPLAN Not.*, 17:33–42.

Schaeffer, S. E. (2007). Graph clustering. *Computer science review*, 1(1):27–64. Truong, N., Roe, P., and Bancroft, P. (2004). Static analysis of students' java programs.

In *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30*, pages 317–325. Australian Computer Society, Inc.

Wu, W., Li, G., Sun, Y., Wang, J., and Lai, T. (2007). AnalyseC: A Framework for Assessing Students' Programs at Structural and Semantic Level. In *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, pages 742 –747.

Xu, S. and Chee, Y. S. (2003). Transformation-based diagnosis of student programs for programming tutoring systems. *IEEE Transactions on Software Engineering*, 29:360– 384.