Abordagem para seleção de exemplos trabalhados para engenharia de software do domínio de sistemas distribuídos

Breno Farias da Silva, Igor Scaliante Wiese, Rodrigo Campiolo, Marco Aurélio Graciotto Silva
brenofarias@alunos.utfpr.edu.br,{igor,rcampiolo,magsilva}@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Campo Mourão, Paraná, Brasil

Um exemplo trabalhado é definido como um trabalho cognitivo e experimental com o intuito de fornecer uma solução ideal, todavia próxima ao praticável, para um problema específico, no qual uma pessoa com escasso ou nenhum conhecimento acerca do tema possa examinar e aprender com a solução proposta [4].

Exemplos trabalhados promovem o engajamento e retenção de conhecimento [5, 13, 14], reduzindo a carga cognitiva e melhorando a compreensão dos tópicos abordados [4, 10]. Na educação em Computação, existem diversos trabalhos sobre uso de exemplos trabalhados sobre conceitos introdutórios de programação [7, 10] e engenharia de software [5, 13, 14]. No entanto, não foi possível identificar na literatura estudos abrangendo outros tópicos de currículos em Computação, como Sistemas Distribuídos (SD).

Este trabalho explora a seleção de exemplos trabalhados para uso em atividades de aprendizagem em SD. Para esta seleção, estão sendo considerados o alinhamento com currículos sobre SD [8, 9] e a qualidade da implementação de algoritmos relacionados ao tema.

Quanto ao contexto de SD, foram considerados projetos de software livre extensivamente empregados no mercado de trabalho para o desenvolvimento de aplicações distribuídas e selecionados com base na orientação de um professor especialista no domínio: Apache Kafka [3] e *ZooKeeper* [2]. O Kafka é um sistema de mensagens distribuídas fundada no modelo *publish-subscribe* e o *Zookeeper* é um sistema de coordenação distribuída.

Quanto à qualidade de implementação, foi considerada a evolução de métricas de software relacionadas à qualidade do desenho. Para isso, este estudo propõe uma heurística baseada em métricas para examinar a evolução da qualidade do código em SD, com o intuito de identificar exemplos de código que demonstrem melhorias significativas. Foram utilizadas as ferramentas de análise de código *CK* [1] para obter as métricas e o *RefactoringMiner* [15–17] para identificar refatorações associadas com mudanças de desenho de todas as configurações (*commits*) de cada um dos projetos analisados com a ferramenta *PyDriller* [11, 12]. Com esses dados, foram geradas visualizações para auxiliar na identificação e seleção de alterações de código candidatas para a elaboração de exemplos trabalhados.

Para a aplicação do método proposto, foi criada a ferramenta Worked Example Miner [6], que integra as ferramentas citadas no método. A primeira etapa da abordagem consiste em extrair as métricas de cada configuração do projeto sob análise (*commit*). Para isso, combina-se o *PyDriller*, que percorre cada *commit*, com o *CK* para extrair as métricas dos arquivos contendo código Java e o *RefactoringMiner* para extrair as refatorações. Concluída essa extração, a ferramenta analisa os dados e gera, para cada classe e método: descrições estatísticas; regressões lineares; refatorações.

Quanto às descrições estatísticas das métricas, são gerados arquivos no formato *CSV* que sintetizam o estado inicial, final, média, mediana e terceiro quartil de cada métrica para cada classe e método. Nesse passo também podem ser identificadas mudanças bruscas das métricas, que podem ser indicativos de mudanças significativas na qualidade da classe ou método sob análise.

Ainda a partir dos dados extraídos pelo *CK*, são elaboradas regressões lineares, com o intuito de evidenciar tendências nas métricas para métodos e classes dos repositórios analisados. Considerando a regressão linear e os valores extraídos, são gerados gráficos de cada classe ou método com relação a cada métrica, permitindo a análise da evolução da qualidade desses elementos.

Das refatorações identificadas pelo RefactoringMiner, são selecionadas aqueles do tipo *Extract Method*, *Extract Class*, *Pull Up Method*, *Push Down Method* e *Extract Superclass*, visto que refletem melhorias na qualidade do desenho do código. Esse resultado é armazenado em arquivos no formato *JSON*.

Atualmente a análise dos resultados gerados pela ferramenta proposta é realizada manualmente, de forma exploratória, para entender a evolução da qualidade e a pertinência das classes e métodos quanto aos tópicos de SD, e para identificar alterações de código candidatas para a seleção de exemplos trabalhados. Por exemplo, para o Zookeeper, a análise da classe *org.apache.zookeeper.server.quorum.Follower* pelas regressões lineares revelou tendências decrescentes nas métricas *Coupling Between Objects* (CBO), *Response for a Class* (RFC) e *Weight Method Class* (WMC), sugerindo melhorias na modularidade e coesão. Esta classe é central para o mecanismo de consenso, em que *followers* concordam em votações. Além da regressão linear, a classe foi também objeto de uma refatoração do tipo *Pull Up Method* para reduzir a redundância e promover a reusabilidade do código. Dessa forma, é possível selecionar esta classe e as alterações de código associadas com as melhorias de qualidade como candidatas para um exemplo trabalhado sobre SD.

Com base nesses resultados preliminares, percebe-se que a abordagem proposta pode contribuir para a seleção de exemplos trabalhados para SD. Como trabalhos futuros, considera-se a integração de métricas avaliando aspectos dinâmicos do software e a aplicação da metodologia em uma variedade maior de projetos.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Processo

EduComp24, Abril 22-27, 2024, São Paulo, São Paulo, Brasil (On-line)

408812/2021-4 - e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

REFERÊNCIAS

[1] Maurício Aniche et al. 2015. Java code metrics calculator (CK). Programa de computador. https://github.com/mauricioaniche/ck

[2] Apache Software Foundation. 2008. Zookeeper. Programa de computador. https://zookeeper.apache.org/

[3] Apache Software Foundation. 2011. Kafka. Programa de computador. https://kafka.apache.org/

[4] Robert K. Atkinson, Sharon J. Derry, Alexander Renkl, and Donald Wortham. 2000. Learning from Examples: Instructional Principles from the Worked Examples Research. *Review of Educational Research* 70, 2 (June 2000), 181–214.

[5] Tiago Piperno Bonetti, Matheus Molina Dias, Williamson Silva, and Thelma Elita Colanzi. 2023. Students' Perception of Example-Based Learning in Software Modeling Education. In *XXXVII Brazilian Symposium on Software Engineering (SBES 2023)* (37 ed.) (Campo Grande, MS, Brasil). ACM, New York, NY, EUA, 67–76.

[6] Breno Farias da Silva. 2023. Worked Example Miner. Programa de computador. https://github.com/BrenoFariasdaSilva/Worked-Example-Miner

[7] Kasia Muldner, Jay Jennings, and Veronica Chiarelli. 2023. A Review of Worked Examples in Programming Activities. *Transactions on Computing Education* 23, 1 (March 2023), 13:1–13:35.

[8] Sushil K. Prasad, T. Estrada, S. Ghafoor, A. Gupta, K. Kant, C. Stunkel, A. Sussman, R. Vaidyanathan, C. Weems, K. Agrawal, M. Barnas, D. W. Brown, R. Bryant, D. P. Bunde, C. Busch, D. Deb, E. Freudenthal, J. Jaja, M. Parashar, C. Phillips, B. Robey, A. Rosenberg, E. Saule, and Chi Shen. 2020. *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing – Core Topics for Undergraduates*. Technical Report. NSF/IEEE-TCPP. 53 pages. http://tcpp.cs.gsu.edu/curriculum v. 2.0 beta.

[9] Rajendra K. Raj and Amruth N. Kumar. 2022. Toward Computer Science Curricular Guidelines 2023 (CS2023). *ACM Inroads* 13, 4 (nov 2022), 22–25.

[10] Ben Skudder and Andrew Luxton-Reilly. 2014. Worked Examples in Computer Science. In *Proceedings of the Sixteenth Australasian Computing Education Conference* (6 ed.) (Auckland, Nova Zelandia), Vol. 148. Australian Computer Society, Darlinghurst, Austrália, 59–64.

[11] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller. Programa de computador. https://github.com/ishepard/pydriller

[12] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (26 ed.) (Lake Buena Vista, FL, EUA). ACM, New York, NY, EUA, 908–911.

[13] Simone Tonhão, Thelma Colanzi, and Igor Steinmacher. 2021. Using Real Worked Examples to Aid Software Engineering Teaching. In *35th Brazilian Symposium onSoftware Engineering (SBES 2021)* (35 ed.) (Joinville, SC, Brasil), Márcio Ribeiro, Marco Túlio Valente, Christina von Flach G. Chavez, Elisa Yumi Nakagawa, Kiev Santos da Gama, Simone do Rocio Senger de Souza, Ingrid Nunes, and Rohit Gheyi (Eds.). ACM, New York, NY, EUA, 133–142.

[14] Simone Tonhão, Williamson Silva, Thelma Colanzi, and Igor Steinmacher. 2022. Uma plataforma gamificada de desafios baseados em worked examples extraídos de projetos de Software Livre para o ensino de Engenharia de Software. In *XVII Simpósio Brasileiro de Sistemas Colaborativos* (17 ed.) (Rio de Janeiro, RJ, Brasil). SBC, Porto Alegre, RS, Brasil, 33–38.

[15] Nikolaos Tsantalis et al. 2014. RefactoringMiner. Programa de computador. https://github.com/tsantalis/RefactoringMiner

[16] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. 2022. RefactoringMiner 2.0. *Transactions on Software Engineering* 48, 3 (March 2022), 930–950.

[17] Nikolaos Tsantalis, Matin Mansouri, Laleh M. Eshkevari, Davood Mazinanian, and Danny Dig. 2018. Accurate and Efficient Refactoring Detection in Commit History. In *40th International Conference on Software Engineering* (40 ed.) (Gothenburg, Suécia). ACM, New York, NY, EUA, 483–494.