Bringing Industry-Grade Code Quality and Practices into Software Engineering Education (Doctoral Consortium)

Anastasiia Birillo
JetBrains Research
Belgrade, Serbia
Utrecht University
Utrecht, The Netherlands
anastasia.birillo@jetbrains.com

Abstract

Using professional development tools and practices is an essential part of being a programmer. However, beginners often struggle with professional tools. In this work, we ask the question: **"How can we adapt professional programming tools to improve software engineering education?"** and aim to find efficient ways to solve this problem.

CCS Concepts

• **Social and professional topics** → **Student assessment**; **Software engineering education**; • **Computing methodologies** → **Artificial intelligence**; • **Human-centered computing** → *Interactive systems and tools.*

Keywords

Code Quality Assessment, Code Formatting, LLMs, Generative AI, Next-Step Hints

ACM Reference Format:
Anastasiia Birillo. 2024. Bringing Industry-Grade Code Quality and Practices into Software Engineering Education (Doctoral Consortium). In . ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Nowadays, students often focus only on solving coding tasks while learning programming. However, to become a programmer, it is not enough to learn how to code, it is also important to learn how to use professional tools and apply industry practices. The main research question of this work is: ***"How can we adapt professional programming tools to improve software engineering education?"***. This work is focused on two main aspects: (1) helping students to write high-quality code by using professional code quality checkers and (2) improving the in-IDE experience of learning computer science by providing personalized help in programming tasks.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Related Work

Code Quality Assessment Tools. The issue of assessing code quality in education has been actively studied before [7, 10, 11]. Several tools have been created that partially reimplement code quality checks from professional solutions [6, 8, 12, 21]. However, these tools are focused on a single programming language, often handle only basic errors typical of novice programmers, and do not teach students how to handle errors encountered in industry coding scenarios.

Personalized Feedback in Education. One of the possible ways to provide personalized help in education is *next-step hint generation*, *i.e.*, showing a student what specific small step they need to do next. There are many ways to generate such hints [13], from predefined rules or templates [9] to using systems based on previous student submissions [17, 18]. The state-of-the-art works use LLMs to provide hints to students [14, 16, 19]. However, these works face a significant challenge to reduce the number of hallucinations and to get LLMs to follow the prompt instructions strictly [15].

3 Current Work

RQ#1. How can professional code quality checkers help students learn to write high-quality code? To introduce code quality assessment into the educational process, the *Hyperstyle* tool [5] was developed, which is built on top of several existing professional code quality checkers. To make help suitable for students, it adapts the error messages, groups code quality issues into categories, and provides a mechanism for summarizing the feedback into a final code quality score on a four-point scale. The impact of the tool was evaluated with 594 students from the popular MOOC platform *Hyperskill* [1] by comparing the median number of code quality issues for students before and after integrating the tool. The results showed circumstantial evidence that the tool contributed to improving the students' code quality.

RQ#2. What is the behavior and performance of students working with professional code quality checkers? Then, to confirm our findings from the first study, we conducted a large-scale analysis of student submissions from the *Hyperskill* platform to investigate the influence of the developed *Hyperstyle* tool on the code quality of student submissions. First, an algorithm for detecting code quality issues in the templates of programming tasks was developed to remove irrelevant code quality issues that students did not make [3], and it was then validated on a dataset of 415 student solutions. This algorithm was used to remove all non-student code quality issues from the student submissions. Then, we carried out a code quality analysis of 2.3 million submissions from

the *Hyperskill* platform, where we examined both the most common types of code quality issues and the dynamics of how students fix them [20]. The findings confirmed that the developed approach helps students write high-quality code that does not contain code quality issues.

RQ#3. What aspects of industry-grade IDEs can be adapted to enhance the student learning experience? Recently, an in-IDE learning approach was introduced [4], which is implemented as a plugin [2] to the JetBrains IDEs, customizing the IDEs for students. The approach allows students to conduct courses entirely within this realistic professional environment. *Hyperskill* uses this plugin when students switch to the IDE while solving more complex programming exercises. To define how this format can be adapted to improve the learning experience, we conducted the first exploratory study with eight one-hour interviews with students who completed at least one course within the new in-IDE format. The findings suggest that overall, students welcome learning within the IDE, as it allows them to learn in a more realistic scenario. However, one of the biggest problems for students is the lack of personalized help during learning. The in-IDE learning format opens up the possibility of combining static analysis with state-of-the-art approaches that use LLMs to solve this problem.

RQ#4. How can static analysis and LLMs be combined to improve the quality of automated hints? Then, to improve students' in-IDE experience, a next-step hints system was designed that combines LLMs and static analysis to provide both textual and code hints for programming tasks. Based on prior work and our experience, a list of validation criteria was proposed to evaluate the approach, and two rounds of expert validation were conducted to improve the overall quality of the hint system. Finally, the next-step hints were evaluated in a classroom with 14 students from two universities. The results show that both forms of the hints — textual and code — were helpful for the students, and the proposed system helped them to proceed with the coding tasks better.

4 Planned Work

The final part of this thesis focuses on the exploratory evaluation of the proposed AI next-step hints system design, comparing it with the hints provided by other state-of-the-art approaches and analyzing students' behavior when working in the system and receiving hints. This study will open up possibilities for new directions in this area, *e.g.*, when exactly we need to show such hints, how we can adapt hints to the student's background, etc.

References

[1] 2024. *Hyperskill platform.* Retrieved October 5, 2024 from https://hyperskill.org/

[2] 2024. *JetBrains Academy Plugin.* Retrieved October 5, 2024 from https://plugins.jetbrains.com/plugin/10081-jetbrains-academy

[3] Anastasiia Birillo, Elizaveta Artser, Yaroslav Golubev, Maria Tigina, Hieke Keuning, Nikolay Vyahhi, and Timofey Bryksin. 2023. Detecting Code Quality Issues in Pre-written Templates of Programming Tasks in Online Courses. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1.* 152–158.

[4] Anastasiia Birillo, Mariia Tigina, Zarina Kurbatova, Anna Potriasaeva, Ilya Vlasov, Valerii Ovchinnikov, and Igor Gerasimov. 2024. Bridging Education and Development: IDEs as Interactive Learning Platforms. In *Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments.* 53–58.

[5] Anastasiia Birillo, Ilya Vlasov, Artyom Burylov, Vitalii Selishchev, Artyom Goncharov, Elena Tikhomirova, Nikolay Vyahhi, and Timofey Bryksin. 2022. Hyperstyle: A tool for assessing the code quality of solutions to programming assignments. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1.* 307–313.

[6] Hannah Blau and J Eliot B Moss. 2015. FrenchPress gives students automated feedback on Java program flaws. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education.* 15–20.

[7] Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle Van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. 2018. "I know it when I see it" – Perceptions of Code Quality: ITiCSE'17 Working Group Report. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports.* 70–85.

[8] Rohan Roy Choudhury, Hezheng Yin, and Armando Fox. 2016. Scale-driven automatic hint generation for coding style. In *International Conference on Intelligent Tutoring Systems.* Springer, 122–132.

[9] Alex Gerdes, Bastiaan Heeren, Johan Jeuring, and L. Thomas van Binsbergen. 2016. Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback. *International Journal of Artificial Intelligence in Education* 27 (02 2016).

[10] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2017. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education.* 110–115.

[11] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2019. How teachers would help students to improve their code. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education.* 119–125.

[12] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2021. A tutoring system to learn code refactoring. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.* 562–568.

[13] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43.

[14] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research.* 1–11.

[15] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, and Li Zhang. 2024. Exploring and evaluating hallucinations in LLM-powered code generation. *arXiv preprint arXiv:2404.00971* (2024).

[16] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 2024. Teaching CS50 with AI: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1.* 750–756.

[17] Thomas W. Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education.* 483–488.

[18] Kelly Rivers and Kenneth R Koedinger. 2013. Automatic generation of programming feedback: A data-driven approach. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, Vol. 50. 50–59.

[19] Lianne Roest, Hieke Keuning, and Johan Jeuring. 2024. Next-Step Hint Generation for Introductory Programming Using Large Language Models. In *Proceedings of the 26th Australasian Computing Education Conference.* 144–153.

[20] Maria Tigina, Anastasiia Birillo, Yaroslav Golubev, Hieke Keuning, Nikolay Vyahhi, and Timofey Bryksin. 2023. Analyzing the quality of submissions in online programming courses. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET).* IEEE, 271–282.

[21] Leo C Ureel II and Charles Wallace. 2019. Automated critique of early programming antipatterns. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* 738–744.