Using Notional Machines to Automatically Assess Students' Comprehension of Their Own Code

Joey Bevilacqua
joey.bevilacqua@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Luca Chiodini
luca.chiodini@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Igor Moreno Santos
igor.moreno.santos@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Matthias Hauswirth
matthias.hauswirth@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

ABSTRACT

Code comprehension has been shown to be challenging and important for a positive learning outcome. Students don't always understand the code they write. This has been exacerbated by the advent of large language models that automatically generate code that may or may not be correct. Now students don't just have to understand their own code, but they have to be able to critically analyze automatically generated code as well.

To help students with code comprehension, instructors often use notional machines. Notional machines are used not only by instructors to explain code, but also in activities or exam questions given to students. Traditionally, these questions involve code that was not written by students. However, asking questions to students about their own code (Questions on Learners' Code, QLCs) has been shown to strengthen their code comprehension.

This poster presents an approach to combine notional machines and QLCs to automatically generate personalized questions about learners' code based on notional machines. Our aim is to understand whether notional machine–based QLCs are effective. We conducted a pilot study with 67 students to test our approach, and we plan to conduct a comprehensive empirical evaluation to study its effectiveness.

ACM Reference Format:
Joey Bevilacqua, Luca Chiodini, Igor Moreno Santos, and Matthias Hauswirth. 2024. Using Notional Machines to Automatically Assess Students' Comprehension of Their Own Code. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626253.3635524

1 INTRODUCTION

Code comprehension is an important skill to acquire and is considered important for a positive learning outcome [3, 9, 10]. Learners often struggle to understand their code, which may be a mix of their own production, code from textbooks, and code found online. With the increasing capabilities and popularity of large language models, which are able to generate code from a user–supplied prompt, this problem becomes more and more relevant. The bar for *producing code* gets lowered even more. Empowering learners with such tools might have the consequence of decreasing the comprehension of their own code, given that there is less need to reason about the various aspects that pertain to the implementation. Moreover, these models provide no guarantee that the generated code is correct: code comprehension as a skill has never been more crucial.

To help students with code comprehension, instructors often use *Notional Machines* [11], "pedagogic devices used to assist the understanding of some aspect of programs or programming" [4]. Notional machines are used not only by instructors to explain code, but also in activities or exams given to students. Traditionally, the code used in these tasks is not student code. However, asking questions to students about *their own* code has been shown to strengthen their code comprehension [5]. Recent work on *Questions on Learners' Code* (QLCs) [7] explored approaches to automatically generate questions about the code written by students. QLCs are usually multiple–choice questions and can thus be easily graded automatically. Multiple studies [6, 8] have shown that bad scores on QLCs correlate with bad performance in courses.

Our approach combines QLCs and notional machines to automatically generate and assess notional machine–based questions about learners' code. Our aim is to understand the effectiveness of notional machine–based QLCs.

2 INSTANTIATING THE APPROACH

To study our approach we need to focus on a specific notional machine. We picked *Expression as Tree* [4] because expressions have been shown to be prevalent in the largest repository of student code (Blackbox [1]) but neglected in textbooks [2].

We implemented a platform that instantiates our approach for teaching expressions in Java using the *Expression as Tree* notional machine. In our platform, the instructor selects expression–related constructs such as arithmetic operators, method invocations, and array accesses by specifying a query in a domain–specific language. Using the code already written by learners in their programming assignments, our platform finds expressions that include the instructor–specified constructs. Based on those expressions, it generates personalized tasks for students to solve. Specifically, students are

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0424-6/24/03.
https://doi.org/10.1145/3626253.3635524

asked to translate source code into expression trees and to specify the types of sub–expressions.

```
class LinAlgebra {
  double[] scale(double a, double[] x) { ... }
  double scaledAt(double a, double[] x,
                  int oneBasedPos) {
    return scale(a, x)[oneBasedPos - 1];
  }
}
```

Listing 1: An expression (yellow) containing a method invocation and an array access, selected from student code.

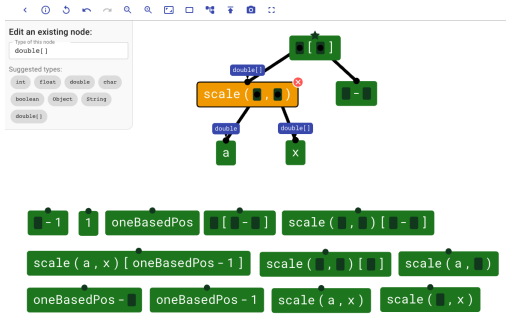

Figure 1: An expression tree (top) being constructed using automatically generated nodes (bottom) which include distractors. Nodes can be annotated (left) with types (blue labels).

Figure 1 illustrates an activity automatically generated from the expression highlighted in Listing 1. The task of the student is to construct the tree corresponding to the given expression. They are provided with a set of disconnected nodes. Some nodes are necessary for constructing the correct tree, other nodes are distractors our platform automatically generates. Students also need to specify the type of each sub-expression by attaching labels to each node.

The platform also generates the reference solution: the correct expression tree with the correct types specified for all nodes. It compares the reference solution with the student's submission and produces feedback. The feedback for students highlights incorrectly represented constructs and reports errors in the tree structure. The feedback for the instructor includes aggregated results and summarizes the observed mistakes tallying incorrectly represented constructs.

3 PILOT STUDY AND FUTURE WORK

We performed a pilot study in a university–level Java programming course with 93 students, of which 67 gave consent for their data to be included in our study. We used the platform for two programming assignments. For each assignment, each student received a personalized task, automatically constructed from the code they submitted as a solution to the assignment. Solving these tasks was not mandatory for students; in the end, we received a total of 52 solutions to the notional machine-based questions. The platform identified 25 incorrect tree nodes, which it reported in feedback for the students and the course instructor.

Our preliminary results show that our approach is practical, with the tool being used in classes of a university–level course on object–oriented programming. Our next step is to conduct a comprehensive empirical evaluation. Our evaluation will include the quality of the automatically generated distractor nodes, the effectiveness, and the accuracy of the automatic feedback generation for both instructors and students, as well as the correlation of the performance of learners on the tasks with their performance in other assessments.

ACKNOWLEDGMENTS

This work was partially funded by the Swiss National Science Foundation project 200021_184689.

REFERENCES

[1] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 223–228. https://doi.org/10.1145/2538862.2538924

[2] Luca Chiodini, Igor Moreno Santos, and Matthias Hauswirth. 2022. Expressions in Java: Essential, Prevalent, Neglected?. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on SPLASH-E (SPLASH-E 2022)*. Association for Computing Machinery, New York, NY, USA, 41–51. https://doi.org/10.1145/3563767.3568131

[3] Peter Donaldson and Quintin Cutts. 2018. Flexible Low-Cost Activities to Develop Novice Code Comprehension Skills in Schools. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education (WiPSCE '18)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/3265757.3265776

[4] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. 2020. Notional Machines in Computing Education: The Education of Attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '20)*. Association for Computing Machinery, New York, NY, USA, 21–50. https://doi.org/10.1145/3437800.3439202

[5] Rita Garcia, Katrina Falkner, and Rebecca Vivian. 2019. Instructional Framework for CS1 Question Activities. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 189–195. https://doi.org/10.1145/3304221.3319732

[6] Teemu Lehtinen, Lassi Haaranen, and Juho Leinonen. 2023. Automated Questionnaires About Students' JavaScript Programs: Towards Gauging Novice Programming Processes. In *Australasian Computing Education Conference*. ACM, Melbourne VIC Australia, 49–58. https://doi.org/10.1145/3576123.3576129

[7] Teemu Lehtinen, Andre L. Santos, and Juha Sorva. 2021. Let's Ask Students About Their Programs, Automatically. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, Madrid, Spain, 467–475. https://doi.org/10.1109/ICPC52881.2021.00054

[8] Teemu Lehtinen, Otto Seppälä, and Ari Korhonen. 2023. Automated Questions About Learners' Own Code Help to Detect Fragile Prerequisite Knowledge. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, Turku Finland, 505–511. https://doi.org/10.1145/3587102.3588787

[9] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further Evidence of a Relationship between Explaining, Tracing and Writing Skills in Introductory Programming. *ACM SIGCSE Bulletin* 41, 3 (July 2009), 161–165. https://doi.org/10.1145/1595496.1562930

[10] Greg L. Nelson, Benjamin Xie, and Andrew J. Ko. 2017. Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research - ICER '17*. ACM Press, Tacoma, Washington, USA, 2–11. https://doi.org/10.1145/3105726.3106178

[11] Juha Sorva. 2013. Notional Machines and Introductory Programming Education. *ACM Transactions on Computing Education* 13, 2 (June 2013), 1–31. https://doi.org/10.1145/2483710.2483713