Shifting Programming Education Assessment from Exercise Outputs Toward Deeper Comprehension

André L. Santos
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Portugal

Abstract

Practice and assessment in introductory programming courses are typically centered on problems that require students to write code to produce specific outputs. While these exercises are necessary and useful for providing practice and mastering syntax, their solutions may not effectively measure the learners' real understanding of programming concepts. Misconceptions and knowledge gaps may be hidden under an exercise solution with correct outputs. Furthermore, obtaining answers has never been so easy in the present era of chatbots, so why should we care (much) about the solutions? Learning a skill is a process that requires iteration and failing, where feedback is of utmost importance. A programming exercise is a means to build up reasoning capabilities and strategic knowledge, not an end in itself. It is the process that matters most, not the exercise solution. Assessing if the learning process was effective requires much more than checking outputs.

I advocate that introductory programming learning could benefit from placing more emphasis on assessing learner comprehension, over checking outputs. Does this mean that we should not check if the results are correct? Certainly not, but a significant part of the learning process would focus on assessing and providing feedback regarding the comprehension of the written code and underlying concepts. Automated assessment systems would reflect this shift by comprising evaluation items for such a purpose, with adequate feedback. Achieving this involves numerous challenges and innovative technical approaches. In this talk, I present an overview of past and future work on tools that integrate code comprehension aspects in the process of solving programming exercises.

2012 ACM Subject Classification Social and professional topics → Computer science education; Applied computing → Computer-assisted instruction

Keywords and phrases Introductory programming, assessment, comprehension

Digital Object Identifier 10.4230/OASIcs.ICPEC.2023.1

Category Invited Talk

Acknowledgements I thank the ICPEC organizing committee for this Invited Talk.

1 Do programming learners fully understand their code?

Studies have shown that programming assignments that are successfully solved do not necessarily have a matching learner confidence [13, 11], while a significant number of students may struggle to explain their own code [15]. In other words, a learner's ability to write a correct solution to a problem does not imply mastery of the underlying concepts, algorithms, and programming primitives. Despite reaching solutions that work, the learner may hold misconceptions [7, 18] about the written code. Even if a learner did not cheat and actually wrote the code, the latter could have been obtained through tinkering and trial-and-error until reaching a working solution. In my experience as a programming instructor, I often get surprised when a third-year student almost graduating cannot interpret rather elementary aspects of program execution and errors. (I ask myself: *How did the student reach this point without understanding these matters? Systematically hammering out programs until they work as expected?*)

These days we have just reached the era of chatbots capable of correctly solving most introductory programming assignments [8]. These systems, such as the popular ChatGPT[1], not only solve problems but also have a generative nature that allows them to output different solutions for those, with accompanying detailed explanations. Until now one could easily find code snippets for typical programming problems by browsing the Web. However, the learner had to make at least a minimal effort regarding the interpretation and integration of the search. With the advent of the latest chatbots, that effort is reduced to the minimum possible – copy and paste the problem statement. Furthermore, there are barely any constraints regarding the written or programming language. This implies that one may easily pass an online programming exam blindly, just using the outputs of chatbots. Some solutions will fail or not be optimal, but the overall score is likely to be positive. Therefore, obtaining solutions has never been so easy, and this is likely to remain as such.

If just reading solutions would be adequate to learn to program, chatbots would not even be necessary, and students simply would be provided with a bundle of problem statements and corresponding solutions.[2] Most programming exercises are "classical", for which thousands of solutions exist out there. The ease of obtaining solutions implies that they lose their value. However, the real benefit of solving a programming exercise is not reaching a solution, but rather the process of developing it – the problem interpretation, the strategy to approach it, how to express it, handling common errors and bugs that will occur, and in turn, their interpretation and strategy to overcome those. If a learner obtains immediate working solutions without going through this process and just focuses on checking whether they meet the desired outcome, will certainly acquire weaker skills. In analogy with natural languages, it is like training to write sentences by modifying existing ones, but not being able to express oneself by writing on a blank sheet.

Even in a pre-chatbot era, overreliance on automatic assessment systems could lead to an "autograder insanity" phenomenon [5], where students replace the task of testing their own solutions with highly frequent resubmissions to the system until they obtain a high score. Not imposing limits and/or penalties for overusing the submission system may incentivize a trial-and-error and tinkering approach to problem-solving. I believe this is a poor educational practice, because one may reach a solution with high quality, but not fully understand all aspects that characterize it as such. While in a small programming exercise against an autograder a trial-and-error tactic may suffice to reach a working solution, in real settings that strategy may be highly inefficient due to aspects pertaining to state space size, system complexity, and development settings.

Access to (reliable) information is generally perceived as beneficial. Still, I argue that the absence of learning strategies that are adapted to this new reality may hinder programming learning processes (and other subjects). Struggling students will likely use the means at their disposal to overcome their difficulties, especially when they are considered legitimate. I believe that if the activities for fulfilling programming course requirements remain focused on obtaining solutions, the widespread use of chatbots might lead to a shallow acquisition of programming skills.

---

[1] `https://chat.openai.com/`

[2] Worked examples [4] are an effective learning means, but their aim is not to replace deliberate practice.

2 How can courseware help to improve program comprehension?

I believe that the instructional design of programming courses could place more emphasis on assessing the learner's comprehension of programming concepts, algorithms, and code understanding while downplaying the accomplishment of reaching a solution that produces the expected outputs. In this perspective, an exercise would not be completed until some assessment of program comprehension is carried out. My hypothesis is that deemphasizing solution outputs will, to some degree, shift the learners' attention and learning time to understanding.

A possible approach to assess code understanding is to pose questions about learners' code [16]. Given that having a human tutor to carry out this role systematically for each individual learner is likely not to be feasible in practice due to instructor availability and cost, such an approach would better scale using automated assessment systems. A recent survey [17] concluded that such systems generally do not comprise this sort of meta-cognition feedback on the submitted code solutions. Jask [21] is a research prototype capable of generating question-answer pairs about Java code against methods. Our early experiment with introductory programming students revealed a high failure rate ($> 60\%$) on questions involving program dynamics (e.g., variable tracing, call stack) [21], while their solutions were producing the expected outputs. Another study with questions on JavaScript [14] has revealed similar failure rates and found that students that repeatedly fail these questions are more likely to drop out. The results confirm that correct exercise solutions do not imply an understanding of the inner workings of programs. This fragility may become evident as problems and algorithms get more complex, but that was not yet evaluated.

Questions about learners' code could be applied in a post-submission fashion in an automated assessment system, where a learner would face questions about the submitted code solution. Upon submitting a solution, the system could also ask questions about other solutions for the same exercise from other students. This would lead the learner to carry out code comprehension tasks related to the same matter, possibly strengthening the related skills by having to interpret similar or different solutions.

Questions about learners' code may be posed at different moments and target different concerns. For example, an inquisitive code editor [10] may prompt questions to learners when the written code reveals a hypothetical misconception, fostering users to reflect on their code. Another possibility for asking questions is during debugging (when errors occur) [1], leading users to reflect on the cause of the errors (instead of trying something else straight away).

Another form of assisting programming learners during exercise solving is by providing feedback and appropriate hints. When a learner cannot progress in an exercise the first temptation might be to look for the solution somewhere else, such as a chatbot. However, in their current form, chatbots may straightly provide a complete solution. As discussed earlier, this may cause the learner to go through the exercise with minimal reflection, despite that chatbots are also capable of outputting detailed explanations of the provided solutions. Jinter [9] is a system to provide fine-grained hints for progressing and receiving feedback in programming exercises. Instead of providing straight answers, the hints attempt at leading the learner to a viable path, while demanding some reflection. Other approaches have focused on providing feedback for improving the code, for instance, with respect to code quality [2] and refactoring [12].

Following a different research line – educational programming environments (e.g., Ville [19], BlueJ [6], Thonny [3], PandionJ [20]) – I postulate that the programming environments themselves could improve their role in program comprehension. Namely, by providing

additional insights that would be otherwise unnoticeable or of difficult access, such as providing facilities for users to: see execution history, ask questions about program behavior, trace output to program statements, detail error explanations and location, and present information about execution performance (time and memory). Hypothetically, the absence of available information to help understand what went wrong in a program incentivizes a learner to search for other forms of overcoming the problem.

To conclude, chatbots have the potential of being a fabulous aid to programming learners that are stuck in their progress. The great novelty of chatbots relates to their immediacy and conversational nature, while they do not provide anything that is not explained elsewhere. Programming education will have to adapt to this new reality, but I argue that educators should not overvalue chatbots as if they were a silver bullet – in the end, one still has to understand how programs work. Time will tell how chatbots affect the learning processes of programming.

References

1 Fatima Abu Deeb and Timothy Hickey. Reflective debugging in Spinoza V3.0. In *Australasian Computing Education Conference*, ACE '21, pages 125–130, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3441636.3442313.

2 Francisco Alfredo, André L. Santos, and Nuno Garrido. Sprinter: A didactic linter for structured programming. In Alberto Simões and João Carlos Silva, editors, *Third International Computer Programming Education Conference, ICPEC 2022, June 2-3, 2022, Polytechnic Institute of Cávado and Ave (IPCA), Barcelos, Portugal*, volume 102 of *OASIcs*, pages 2:1–2:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASIcs.ICPEC.2022.2.

3 Aivar Annamaa. Introducing Thonny, a Python IDE for learning programming. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 117–121, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2828959.2828969.

4 Robert K. Atkinson, Sharon J. Derry, Alexander Renkl, and Donald Wortham. Learning from examples: Instructional principles from the worked examples research. *Review of Educational Research*, 70(2):181–214, 2000. doi:10.3102/00346543070002181.

5 Elisa Baniassad, Lucas Zamprogno, Braxton Hall, and Reid Holmes. Stop the (autograder) insanity: Regression penalties to deter autograder overreliance. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 1062–1068, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3408877.3432430.

6 Jens Bennedsen and Carsten Schulte. BlueJ visual debugger for learning the execution of object-oriented programs? *ACM Transactions on Computing Education*, 10(2):8:1–8:22, June 2010. doi:10.1145/1789934.1789938.

7 Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. A curated inventory of programming language misconceptions. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 380–386, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3430665.3456343.

8 James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, pages 10–19, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3511861.3511863.

9 Jorge Gonçalves and André L. Santos. Jinter: a hint generation system for Java exercises. In *28th annual ACM conference on Innovation and Technology in Computer Science Education (ITiCSE) (to appear)*, 2023.

10 Austin Z. Henley, Julian Ball, Benjamin Klein, Aiden Rutter, and Dylan Lee. An inquisitive code editor for addressing novice programmers' misconceptions of program behavior. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2021, Madrid, Spain, May 25-28, 2021*, pages 165–170. IEEE, 2021. doi:10.1109/ICSE-SEET52601.2021.00026.

11 Cazembe Kennedy and Eileen T. Kraemer. Qualitative observations of student reasoning. In *The 24th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 224–230. ACM, 2019. doi:10.1145/3304221.3319751.

12 Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. A tutoring system to learn code refactoring. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 562–568, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3408877.3432526.

13 Päivi Kinnunen and Beth Simon. My program is ok – am I? computing freshmen's experiences of doing programming assignments. *Computer Science Education*, 22(1):1–28, 2012. doi:10.1080/08993408.2012.655091.

14 Teemu Lehtinen, Lassi Haaranen, and Juho Leinonen. Automated questionnaires about students' JavaScript programs: Towards gauging novice programming processes. In *Proceedings of the 25th Australasian Computing Education Conference, ACE 2023, Melbourne, VIC, Australia, 30 January 2023 - 3 February 2023*, pages 49–58. ACM, 2023. doi:10.1145/3576123.3576129.

15 Teemu Lehtinen, Aleksi Lukkarinen, and Lassi Haaranen. Students struggle to explain their own program code. In Carsten Schulte, Brett A. Becker, Monica Divitini, and Erik Barendsen, editors, *ITiCSE '21: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V.1, Virtual Event, Germany, June 26 - July 1, 2021*, pages 206–212. ACM, 2021. doi:10.1145/3430665.3456322.

16 Teemu Lehtinen, André L. Santos, and Juha Sorva. Let's ask students about their programs, automatically. In *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*, pages 467–475. IEEE, 2021. doi:10.1109/ICPC52881.2021.00054.

17 José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ.*, 22(3), June 2022. doi:10.1145/3513140.

18 Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), October 2017. doi:10.1145/3077618.

19 Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. Ville: A language-independent program visualization tool. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, Koli Calling '07, pages 151–159, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc. URL: http://dl.acm.org/citation.cfm?id=2449323.2449340.

20 André L. Santos. Enhancing visualizations in pedagogical debuggers by leveraging on code analysis. In Mike Joy and Petri Ihantola, editors, *Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 22-25, 2018*, pages 11:1–11:9. ACM, 2018. doi:10.1145/3279720.3279732.

21 André L. Santos, Tiago Soares, Nuno Garrido, and Teemu Lehtinen. Jask: Generation of questions about learners' code in Java. In Brett A. Becker, Keith Quille, Mikko-Jussi Laakso, Erik Barendsen, and Simon, editors, *ITiCSE 2022: Innovation and Technology in Computer Science Education, Dublin, Ireland, July 8 - 13, 2022, Volume 1*, pages 117–123. ACM, 2022. doi:10.1145/3502718.3524761.