Automatic feedback and hints on steps students take when learning how to program

Johan Jeuring [1]

Abstract: Every year, millions of students learn how to write programs. Learning activities for beginners almost always include programming tasks that require a student to write a program to solve a particular problem. When learning how to solve such a task, many students need feedback on their previous actions, and hints on how to proceed. For tasks such as programming, which are most often solved stepwise, the feedback should take the steps a student has taken towards implementing a solution into account, and the hints should help a student to complete or improve a possibly partial solution. In this talk I will give an overview of the approaches to automatic feedback and hints on programming steps and discuss our research on how to evaluate the quality of feedback and hints. I will also take the opportunity to involve the audience in some of the dilemmas we are facing.

Keywords: Programming education, feedback, automatic feedback and hints, stepwise solutions

Introduction

Every year, millions of students study some form of computing. In many countries, computer science is part of the obligatory part of the secondary school curriculum, many universities offer computing programs, and at quite a few universities it is one of the largest programs. A computer science program consists of many components, but every program includes at least one module on learning to program.

Learning to program can be done in many ways. It involves amongst others understanding and decomposing a problem, and planning, implementing, and evaluating a solution [RRR03]. When learning to program students may solve Parsons problems, trace code, complete a program with a hole, etc. At some stage in their learning, a student needs to write (part of) a program.

When learning, a student needs feedback [Ra05]. Feedback can be defined as information provided to a learner relating to their skills or understanding as demonstrated on a task or in the completion of a task; usually after instruction [HT07]. Hattie and Timperley propose that effective feedback answers three key questions: “Where am I going?” (Feed-up), “How am I going?” (Feed-back), and “Where to next?” (Feed-forward). Feed-up is about the reason why a student should complete a task and is related to the learning goals of the

[1] Utrecht University, department of Information and Computing Sciences, Utrecht, NL, j.t.jeuring@uu.nl, https://orcid.org/0000-0001-5645-7681

task. Feed-back analyses and gives information about a learner's progress on a task. Feed-forward, finally, consists of help to move students from their current level of understanding towards task mastery.

The influence of feedback on student achievement is well established, with the potential to lead to significant learning gains [KD96]. The effects of feedback vary a lot, depending on the kind of feedback that is provided. Feedback can be on the level of self ("Well done!"), task ("The input-output behavior of your solution is not expected behavior"), process ("First write some test cases before you start on the implementation of the solution"), and self-regulation ("Did you watch the video about testing?"). These levels all vary in their influence on student outcomes, but there is some proof that the latter three categories lead to better results.

Computing education research has studied the potential of providing automatic grading and feedback [Al05,KJH18,Me23] to both realise the potential advantages of providing feedback, and address the large numbers of students taking programming courses and the lack of computer science teachers in many countries. There are many environments that (may) support beginners learning how to write a program, including intelligent tutoring systems [CLW18], online coding environments (Codecademy, Datacamp, Khan academy, Code.org, and many more), and educational games [GX20]. In addition, LLM-based tools such as ChatGPT and Github Copilot may also be helpful in providing feedback to beginners [He23]. Some of these learning environments give automatic feedback on (sometimes partially finished) student solutions, and hints on how to proceed.

Feedback and hints need to be of good quality to support learning. But when do students need feedback and hints when learning how to program, how should it be given, and how can it be automatically generated? How do the general principles for feedback described above translate to the situation in which a student is writing a program? Designers of learning environments make different choices here. How can we evaluate the quality of the feedback and hints provided by the different learning environments?

An ITiCSE Working Group tried to answer the above questions by collecting datasets of steps students take when solving programming problems and annotating these datasets with feedback [Je22]. It turned out that there was quite some disagreement among different experts on providing feedback on student programs. Together with several colleagues, I'm currently working on trying to gain more insight into why experts disagree on giving feedback.

In this talk I will give an overview of the approaches to automatic feedback and hints on programming steps and discuss our research on how to evaluate the quality of feedback and hints. I will also take the opportunity to involve the audience in some of the dilemmas we are facing.

Acknowledgements

This work builds upon the work of the 2022 ITiCSE working group on “Steps learners take when solving programming tasks, and how learning environments (should) respond to them”. Together with Hieke Keuning, Nathalie Kiesler, and Dominic Lohr, I am looking further into how experts would give feedback on programming steps of students learning to program.

Bibliography

[Al05] Ala-Mutka, K. M.: A survey of automated assessment approaches for programming assignments. Computer science education, 15(2), 83-102, 2005.

[CLW18] Crow, T.; Luxton-Reilly, A.; Wuensche, B.: Intelligent tutoring systems for programming education: a systematic review. In Proceedings of the 20th Australasian Computing Education Conference (pp. 53-62), 2018.

[GX20] Giannakoulas, A.; Xinogalos, S.: A review of educational games for teaching programming to primary school students. In Handbook of Research on Tools for Teaching Computational Thinking in P-12 Education, 2020.

[HT07] Hattie, J.; Timperley, H.: The power of feedback. Review of educational research 77.1, 81-112, 2007.

[He23] Hellas, A.; Leinonen, J.; Sarsa, S.; Koutcheme, C.; Kujanpää, L.; Sorva, J.: Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. arXiv preprint arXiv:2306.05715, 2023. To appear in ICER 2023.

[Je22] Jeuring, J.; Keuning, H.; Marwan, S.; Bouvier, D.; Izu, C.; Kiesler, N.; Lehtinen, T.; Lohr, D.; Peterson, A.; Sarsa, S.: Towards Giving Timely Formative Feedback and Hints to Novice Programmers. In Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education, 95-115, 2022.

[KJH18] Keuning, H.; Jeuring, J.; Heeren, B.: A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education (TOCE), 19(1), 1-43, 2018.

[KD96] Kluger, A. N.; DeNisi, A.: The effects of feedback interventions on performance: a historical review, a meta-analysis, and a preliminary feedback intervention theory. Psychological bulletin, 119(2), 254, 1996.

[Me23] Messer, M.; Brown, N. C.; Kölling, M.; Shi, M.: Automated Grading and Feedback Tools for Programming Education: A Systematic Review. arXiv preprint arXiv:2306.11722, 2023.

[Ra05] Race, P.: Making learning happen – A guide for post-compulsory education. Sage, 2005.

[RRR03] Robins, A.; Rountree, J.; Rountree, N.: Learning and teaching programming: A review and discussion. Computer science education, 13(2), 137-172, 20