Analysing Programming Misconceptions

Julien Lienard

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Analysing Programming Misconceptions

Julien Liénard[0009−0009−2966−7179]

Second year PhD Student, UCLouvain, ICTEAM, Louvain-la-Neuve, Belgium

Keywords: CS Education · Pattern Mining · Automatic Feedback on Autograders · Misconceptions · Programming Flaws

1 Goals of the research

For programming education to become truly accessible to any student, whether they be primary, mid school or university students, students from less privileged socio-economic or cultural backgrounds, female students (an underrepresented audience for computer science studies), adults returning to study, or future informatics teachers themselves, any tool that can help students and their teachers to lower the bar to access and complete these studies is worth exploring. Of particular interest are automated teaching tools adapted to the particular needs, skills and pace of each individual student. Ensuring a good informatics background to students also depends a lot on the motivation, quality and background of the teachers; having tools that can help less skilled teachers provide accurate automated feedback to their students can be very helpful [13]. Such automated tools are also beneficial to allow scaling up to larger student audiences, as is increasingly the case for first year computer science courses, while still being able to guarantee an individualised quality feedback.

Teaching something new to students is at least partly a matter of informing them that some of what they think they know just isn't so [34]. Identifying and addressing students' misconceptions should be a key part of computer science teachers competences [12][30]. Students experience many difficulties for various reasons including unfamiliarity of syntax, incomplete prior knowledge, misconceptions, and lack of problem-solving strategies. Knowing what misconceptions students hold in programming helps educators offer appropriate teaching methods to correct those erroneous conceptions. In this project we will study not only how to discover and encode these typical errors and what misconceptions underly them, but also how to extend existing autograders to provide automated feedback to students on their false conceptions and how to reduce, overcome and correct them.

Existing classifications of common misconceptions by novice programmers (cf. §2) serve as a starting point to build supportive tools that use this information to react to student's (wrong) behaviours and provide feedback tailored to address suspected misconceptions. Exploring the technology, building prototypes of and evaluating such advanced automated teaching tools will be the main goal of this research project. We will first complete and improve upon existing classifications, by applying pattern mining and program analysis to large code

repositories of student submissions we have at our disposition [23], as well as by collecting new data from new generations of students. This classification has been used in a first tool that we developed [20]. Our tools creates unit test to detect those coding flaws classified before. It help create more advanced feedback for the student that make common mistakes in the code submitted on the autograder.

2 State of the art

Many authors have studied students' misconceptions and other difficulties in introductory programming [30]. Chiodini proposes a curated inventory of programming language misconceptions [5]; Caceffo proposes the creation of a concept inventory based on such misconceptions [2]; Sorva presents a catalogue of misconceptions, drawn through exploratory research from actual novice programmers' misconceptions [32]; and Grover also categorises common mistakes and misconceptions made by novice programmers [12]. Several other authors have studied possible misconceptions and types of errors made by novice students in introductory programming education as well [1][31][17][6][18][35][19][33]. Such taxonomies will serve as our basis to build supportive tools that use this information to react to student's (wrong) behaviours and provide feedback tailored to address suspected misconceptions.

Research on source code mining [24][29] has been explored to discover interesting structural regularities [21], coding idioms [28], API usage patterns [36], code clones [8], bugs [4], crosscutting concerns [3], systematic changes [11][26]. refactoring opportunities, etc.

In a recent article [23] which formed the seed for the current research project, we conducted an initial experiment where we used a frequent subtree pattern mining algorithm to mine for interesting good, bad or ugly coding idioms made by over 500 undergraduate students from their exam submissions to an introductory programming course. We did so by looking for patterns that distinguished positive examples, corresponding to the more correct answers to each question, from negative examples, corresponding to solutions that failed the question. That paper was a first promising exploratory step to prove the potential of using this technique to find possible misconceptions (corresponding to the bad coding idioms that were discovered).

Many automatic graders [14][16], autograders in short, have been proposed to grade programs submitted by students, either to scale up to larger audiences through automation [25][7][10] or to provide alternative forms of feedback [27][22][9][15]. In this project we will explore how extend the INGInious autograder [10] with information on mined misconceptions to tell students not only where their code is wrong and what tests it doesn't pass, but also why it is wrong and how it could be improved. Even students who solved a question correctly could receive recommendations on how to improve their coding skills.

3 Research project

In this research project we will investigate the following research questions:

1. What are the typical programming mistakes made by novice programmers?
2. What misconceptions do these typical mistakes correspond to?
3. What recommendations can help students overcome such misconceptions?
4. How to automate the detection of such misconceptions and their remediation?
5. How do such automated tools help students and their teachers?

To answer the first research questions, during my first year as a PhD student,, we used a pattern miner to detect common mistakes in first year computer science exam.

The repository we used is the set of programs submitted by students following the first-year introductory programming course for computer scientists and engineers at UCLouvain. This course is followed by over 700 students yearly and makes use of an autograder which collects and corrects students' programs throughout the year and during exam sessions. The course has been running for 4 years using Python as programming language and before that using the Java language. All this data is available for analysis. For all the submissions we even have a trace of all subsequent submissions each student made per question. As outcome of this analysis, we found recurring errors and misconceptions.

For the second research question, research literature contains a vast body of knowledge on programming misconceptions, including several classifications and taxonomies of typical programming errors and their corresponding misconceptions. However, many of those are still incomplete or cover only some of the programming concepts for some programming languages. We used these classifications as a starting point on which to build our own.

Research questions 3 and 4 are dedicated to the development of our automated tool support. For this we will build upon and extend the existing INGInious autograder that was developed at our university. Since it was developed in house, the necessary expertise is available to extend it. As the autograder is already used by our students, the developed tool support will be integrated in an environment that they are already familiar with.

I have integrated a new feature into this autograders to identify a wider range of errors beyond just syntax and input-output issues. This includes detecting conceptual errors, semantic errors, and poor coding practices [20].

Following this initial investigation, during this second year, we have recognized the potential benefit of introducing a user-friendly language for defining code patterns that can be identified within students' work. This development aims to assist educators who may not possess an extensive background in computer science. Our ongoing research on this topic will be the subject of a forthcoming paper.

A quantitative evaluations has been conducted by comparing the performance of students using our tools support with those that are not. We also want to evaluate their performance when using our tools as compared to using similar

tools if they exist. We will also assess whether our automated tool support is helpful for students and their teachers is by evaluating it through them. Qualitative evaluations will be conducted through interviews with volunteering tutors and students involved in our first-year programming course. Their feedback will help us understand whether the tool meets its objectives and steer its further development and improvement if not. Such evaluations will happen at regular intervals throughout the project rather than only at the end.

We also want to try to use more variety of techniques (such as formal concept analysis, association rule mining, frequent subtree mining and clustering) to mine for good or bad coding idioms in student code, in order to reveal potential misconceptions.

Another outcome will be a comparative study of what techniques or combinations of techniques prove most useful to mine for such errors and idioms, thus contributing to the growing research domain of mining software repositories.

This entails conducting iterative research in which we aim to uncover increasing numbers of misconceptions through techniques like pattern mining and others. Subsequently, we will employ these methods to develop tools and evaluate their effectiveness with our students on an annual basis.

References

1. Bayman, P., Mayer, R.E.: A diagnosis of beginning programmers' misconceptions of BASIC programming statements. Communications of the ACM **26**(9), 677–679 (Sep 1983). https://doi.org/10.1145/358172.358408, https://doi.org/10.1145/358172.358408
2. Caceffo, R., Frank-Bolton, P., Souza, R., Azevedo, R.: Identifying and Validating Java Misconceptions Toward a CS1 Concept Inventory. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education. pp. 23–29. ITiCSE '19, Association for Computing Machinery, New York, NY, USA (Jul 2019). https://doi.org/10.1145/3304221.3319771, https://doi.org/10.1145/3304221.3319771
3. Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., Tourwe, T.: A qualitative comparison of three aspect mining techniques. In: 13th International Workshop on Program Comprehension (IWPC'05). pp. 13–22 (May 2005). https://doi.org/10.1109/WPC.2005.2, iSSN: 1092-8138
4. ChandraYadav, D., Pal, S.: Software Bug Detection using Data Mining. International Journal of Computer Applications **115**(15), 21–25 (Apr 2015). https://doi.org/10.5120/20228-2513, http://research.ijcaonline.org/volume115/number15/pxc3902513.pdf
5. Chiodini, L., Moreno Santos, I., Gallidabino, A., Tafliovich, A., Santos, A.L., Hauswirth, M.: A Curated Inventory of Programming Language Misconceptions. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. pp. 380–386. ITiCSE '21, Association for Computing Machinery, New York, NY, USA (Jun 2021). https://doi.org/10.1145/3430665.3456343, https://doi.org/10.1145/3430665.3456343

6. Danielsiek, H., Paul, W., Vahrenhold, J.: Detecting and understanding students' misconceptions related to algorithms and data structures. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education. pp. 21–26. SIGCSE '12, Association for Computing Machinery, New York, NY, USA (Feb 2012). https://doi.org/10.1145/2157136.2157148, https://doi.org/10.1145/2157136.2157148
7. Danutama, K., Liem, I.: Scalable Autograder and LMS Integration. Procedia Technology **11**, 388–395 (Jan 2013). https://doi.org/10.1016/j.protcy.2013.12.207, https://www.sciencedirect.com/science/article/pii/S2212017313003617
8. Deknop, C., Mens, K., Baars, S., Oprescu, A.: Clone Detection vs. Pattern Mining: The Battle p. 7
9. DeNero, J., Sridhara, S., Pérez-Quiñones, M., Nayak, A., Leong, B.: Beyond Autograding: Advances in Student Feedback Platforms. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 651–652. SIGCSE '17, Association for Computing Machinery, New York, NY, USA (Mar 2017). https://doi.org/10.1145/3017680.3017686, https://doi.org/10.1145/3017680.3017686
10. Derval, G., Gego, A., Reinbold, P., Frantzen, B., Van Roy, P.: Automatic grading of programming exercises in a mooc using the inginious platform. European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15) pp. 86–91 (2015)
11. Gerlec, , Krajnc, A., Heričko, M., Božnik, J.: Mining source code changes from software repositories. In: 2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR). pp. 1–5 (Oct 2011). https://doi.org/10.1109/CEE-SECR.2011.6188468
12. Grover, S., Basu, S.: Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 267–272. ACM, Seattle Washington USA (Mar 2017). https://doi.org/10.1145/3017680.3017723, https://dl.acm.org/doi/10.1145/3017680.3017723
13. Henry, J., Joris, N.: Informatics at secondary schools in the french-speaking region of belgium: myth or reality. ISSEP 2016 **13** (2016)
14. Hollingsworth, J.: Automatic graders for programming classes. Communications of the ACM **3**(10), 528–529 (Oct 1960). https://doi.org/10.1145/367415.367422, https://doi.org/10.1145/367415.367422
15. Iosup, A., Epema, D.: An experience report on using gamification in technical higher education. In: Proceedings of the 45th ACM technical symposium on Computer science education. pp. 27–32. SIGCSE '14, Association for Computing Machinery, New York, NY, USA (Mar 2014). https://doi.org/10.1145/2538862.2538899, https://doi.org/10.1145/2538862.2538899
16. Jackson, D., Usher, M.: Grading student programs using ASSYST. In: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education. pp. 335–339. SIGCSE '97, Association for Computing Machinery, New York, NY, USA (Mar 1997). https://doi.org/10.1145/268084.268210, https://doi.org/10.1145/268084.268210
17. Kaczmarczyk, L.C., Petrick, E.R., East, J.P., Herman, G.L.: Identifying student misconceptions of programming. In: Proceedings of the 41st ACM technical symposium on Computer science education. pp.

107–111. SIGCSE '10, Association for Computing Machinery, New York, NY, USA (Mar 2010). https://doi.org/10.1145/1734263.1734299, https://doi.org/10.1145/1734263.1734299
18. Karpierz, K., Wolfman, S.A.: Misconceptions and concept inventory questions for binary search trees and hash tables. In: Proceedings of the 45th ACM technical symposium on Computer science education. pp. 109–114. SIGCSE '14, Association for Computing Machinery, New York, NY, USA (Mar 2014). https://doi.org/10.1145/2538862.2538902, https://doi.org/10.1145/2538862.2538902
19. Kohn, T.: Variable Evaluation: an Exploration of Novice Programmers' Understanding and Common Misconceptions. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 345–350. SIGCSE '17, Association for Computing Machinery, New York, NY, USA (Mar 2017). https://doi.org/10.1145/3017680.3017724, https://doi.org/10.1145/3017680.3017724
20. Lienard, J., Mens, K., Nijssen, S.: Extracting unit tests from patterns mined in student code to provide improved feedback in autograders. In: Seminar Series on Advanced Techniques & Tools for Software Evolution (SATToSE) (2023)
21. Lozano, A., Kellens, A., Mens, K., Arevalo, G.: Mining Source Code for Structural Regularities. In: 2010 17th Working Conference on Reverse Engineering. pp. 22–31 (Oct 2010). https://doi.org/10.1109/WCRE.2010.12, iSSN: 2375-5369
22. MacWilliam, T., Malan, D.J.: Streamlining grading toward better feedback. In: Proceedings of the 18th ACM conference on Innovation and technology in computer science education. pp. 147–152. ITiCSE '13, Association for Computing Machinery, New York, NY, USA (Jul 2013). https://doi.org/10.1145/2462476.2462506, https://doi.org/10.1145/2462476.2462506
23. Mens, K., Nijssen, S., Pham, H.S.: The good, the bad, and the ugly: mining for patterns in student source code. In: Proceedings of the 3rd International Workshop on Education through Advanced Software Engineering and Artificial Intelligence. pp. 1–8. EASEAI 2021, Association for Computing Machinery, New York, NY, USA (Aug 2021). https://doi.org/10.1145/3472673.3473958, https://doi.org/10.1145/3472673.3473958
24. Mens, K., Tourwé, T.: Delving source code with formal concept analysis. Computer Languages, Systems & Structures **31**(3), 183–197 (Oct 2005). https://doi.org/10.1016/j.cl.2004.11.004, https://www.sciencedirect.com/science/article/pii/S1477842405000059
25. Milojicic, D.: Autograding in the Cloud: Interview with David O'Hallaron. IEEE Internet Computing **15**(1), 9–12 (Jan 2011). https://doi.org/10.1109/MIC.2011.2, conference Name: IEEE Internet Computing
26. Molderez, T., Stevens, R., De Roover, C.: Mining change histories for unknown systematic edits. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). pp. 248–256. IEEE (2017)
27. Nicol, D.: From monologue to dialogue: improving written feedback processes in mass higher education. Assessment & Evaluation in Higher Education **35**(5), 501–517 (Aug 2010). https://doi.org/10.1080/02602931003786559, https://doi.org/10.1080/02602931003786559, publisher: Routledge _eprint: https://doi.org/10.1080/02602931003786559
28. Nucci, D.D., Pham, H.S., Fabry, J., Mens, K., Molderez, T., Nijssen, S., Roover, C.D., Zaytsev, V.: Language-Parametric Modular Framework for Mining Idiomatic Code Patterns p. 7

29. Pham, H.S., Nijssen, S., Mens, K., Di Nucci, D., Molderez, T., De Roover, C., Fabry, J., Zaytsev, V.: Mining Patterns in Source Code Using Tree Mining Algorithms. In: Kralj Novak, P., Šmuc, T., Džeroski, S. (eds.) Discovery Science. pp. 471–480. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-33778-0$_3$5
30. Qian, Y., Lehman, J.: Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Transactions on Computing Education **18**(1), 1:1–1:24 (Oct 2017). https://doi.org/10.1145/3077618, https://doi.org/10.1145/3077618
31. Seppälä, O., Korhonen, A., Malmi, L.: Observations on student errors in algorithm simulation exercises. Koli Calling, 17-20.11.2005, Koli, Suomi pp. 81–86 (2005), https://research.aalto.fi/en/publications/observations-on-student-errors-in-algorithm-simulation-exercises, publisher: TURKU CENTRE FOR COMPUTER SCIENCE
32. Sorva, J.: Visual program simulation in introductory programming education. Aalto University (2012), https://aaltodoc.aalto.fi:443/handle/123456789/3534, accepted: 2012-05-31T07:29:53Z ISSN: 1799-4942 (electronic)
33. Swidan, A., Hermans, F., Smit, M.: Programming Misconceptions for School Students. In: Proceedings of the 2018 ACM Conference on International Computing Education Research. pp. 151–159. ICER '18, Association for Computing Machinery, New York, NY, USA (Aug 2018). https://doi.org/10.1145/3230977.3230995, https://doi.org/10.1145/3230977.3230995
34. Taylor, A.K., Kowalski, P.: Student misconceptions: Where do they come from and what can we do? In: Applying science of learning in education: Infusing psychological science into the curriculum, pp. 259–273. Society for the Teaching of Psychology, Washington, DC, US (2014)
35. Veerasamy, A.K., D'Souza, D., Laakso, M.J.: Identifying Novice Student Programming Misconceptions and Errors From Summative Assessments. Journal of Educational Technology Systems **45**(1), 50–73 (Sep 2016). https://doi.org/10.1177/0047239515627263, https://doi.org/10.1177/0047239515627263, publisher: SAGE Publications Inc
36. Zhong, H., Xie, T., Zhang, L., Pei, J., Mei, H.: MAPO: Mining and Recommending API Usage Patterns. In: Drossopoulou, S. (ed.) ECOOP 2009 – Object-Oriented Programming. pp. 318–343. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03013-0$_1$5