Department of Electrical, Computer, and Software Engineering

Part IV Research Project

Literature Review and

Statement of Research Intent

Project Number: 127

A Pedagogic IDE

Yulia Pechorina

Keith Anderson

Paul Denny

14/04/2022

Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

Name: Yulia Pechorina

ABSTRACT: Many of the students currently taking an introductory programming course will go on to have successful careers, helping to shape the world that we live in. However, these courses are notorious for their high attrition rates. One of the factors in these high attrition rates is a lack of problem-solving skills, which are some of the most important skills for a programmer. This is likely because most introductory programming courses do not explicitly teach problem-solving skills, and instead, explicitly teach the syntax and semantics of programming. Explicitly teaching problem-solving skills can be difficult as it requires students to develop metacognitive awareness. Existing problem-solving frameworks seek to solve this problem by formalizing the problem-solving process into a model which can be taught to students and used by students to track their progress while solving a problem. In this report, we review the literature on problem-solving frameworks and approaches that have been used to develop problem-solving skills in novices.

1. Introduction

Problem-solving is an essential skill for all programmers. Without it, programmers would be unable to turn a complex problem into an elegant solution. Despite this, an ITiCSE working group study found that many students in introductory programming courses lacked problem-solving skills [1]. Further, a lack of problem-solving skills is a factor in high attrition rates in programming courses [2].

Poor problem-solving skills in novices could be attributed to the curriculum of introductory programming courses. Programming skills can be divided into programming knowledge and programming problem-solving strategies. Programming knowledge refer to the syntax and semantics of programming, whereas programming problem-solving strategies refers to how one applies programming knowledge to solve a problem [3]. In a typical course curriculum, programming knowledge is explicitly taught, whereas problem-solving strategies are often implicitly taught [4].

Even when explicitly taught problem-solving strategies in a course, students can have difficulty learning problem-solving, as it requires them to develop metacognitive skills, which are hard to teach and take time to develop. Metacognition is defined as "awareness or analysis of one's own learning or thinking processes" [5]. When programming, a student needs metacognitive skills to identify where they are in the problem-solving process.

This paper will investigate the current state of strategies, frameworks, and interventions designed to aid novices in developing problem-solving skills. Section 2.1 investigates multiple problem-solving frameworks specific to programming. Sections 2.2 through 2.5 explore interventions in the problem-solving process categorized by the stages in Loksa et al.'s framework. Finally, Section 3 provides a conclusion of the literature review.

2. Related works

2.1. Problem-solving frameworks

Loksa et al. [6] developed a programming-specific problem-solving framework consisting of the following stages: (1) reinterpreting the problem prompt, (2) searching for analogous problems, (3) searching for solutions, (4) evaluating a potential solution, (5) implementing a solution, and (6) evaluating an implemented solution.

A controlled experiment with 48 students was conducted to measure the effect of teaching Loksa et al.'s framework along with interventions to motivate students to think about their progress along the stages of the problem-solving framework. The experimental group and control group participated in separate coding camps, and the experimental group had multiple interventions in their camp schedule. Intervention (1) was a 1-hour problem-solving lecture. Intervention (2) was a prompt given when a student requested help from an instructor, which asked them to identify their current problem-solving stage. Intervention (3) was a physical handout of the problem-solving stages. Intervention (4) was hints categorized under the problem-solving stages added to the student's IDE. They found that students in the experimental group had more detailed explanations of their problem-solving strategies, required help with design and selection rather than debugging, relied less on the instructors, and had greater productivity and scores. While these interventions were successful, they can be difficult to implement at scale due to no automation for interventions 1 to 3. Intervention 2 had especially low scalability, as it required 1:1 contact between the instructor and student.

Hilton et al. developed a framework called "The Seven Steps" [7]. This consisted of the following steps: (1) work an example yourself, (2) write down exactly what you just did, (3) generalize, (4) test your algorithm, (5) translate to code, and (6) test. These steps are designed to help a student design and implement an algorithm to solve a solution. They taught this framework to students in two introductory programming courses – one for graduate engineering students, and another for undergraduate computer science students. Lab sessions were held to implement steps (1) to (4). The effectiveness of this framework was measured through student surveys. They found that the graduate students had a much higher rate of use of "The Seven Steps" framework than undergraduate students and that there is no statistically significant relationship between the rate of use of the framework and how confident students were in their programming ability.

When compared to the framework given by Loksa et al., this framework is much more algorithmically focused, and therefore is not as generally applicable. Steps (1) and (2) would be particularly hard to implement for a larger problem, such as creating a web application. However, for a small algorithmic problem such as the "closest point" problem as seen in [7, Fig. 2], the "The Seven Steps" framework may be better suited than Loksa et al.'s abstracted framework. While an

algorithmically focused framework is likely useful for an introductory programming course, we feel that it is important to teach frameworks that a student can apply in future years.

Another framework by the ITiCSE working group [8], consisted of the following steps: (1) abstract the problem from its description, (2) generate sub-problems, (3) transform sub-problems into sub-solutions, (4) re-compose, and (5) evaluate and iterate. They conducted an experiment using this framework across several universities, to measure the performance of students in introductory programming courses. No data was collected about the effectiveness of this framework in teaching problem-solving. However, they did find that novices often skip early stages in the problem-solving process. The ITiCSE working group suggested that this could be due to the heavy focus of the taught framework on later stages of the problem-solving process. In comparison, Loksa et. al.'s framework has a greater focus on the early stages, as stage (1) relates to getting the student to understand the problem.

The problem-solving framework by Loksa et al. was chosen to structure the various interventions covered in this literature review, due to its unique focus on metacognition, greater generalizability compared to Hilton et al.'s framework, and a greater focus on the early stages of the problem-solving process compared to the ITiCSE working group's framework. Studies that build on Loksa et al.'s framework typically involve automated interventions to increase scalability and target individual stages of the framework rather than all stages. Currently, there are studies that investigate interventions related to Stages (1), (2), (3), and (5), covered in sections 2.2 through 2.3. A summary of these existing studies is presented in Table 1.

Table 1: Summary of Existing Studies

Author	Paper Title	Stage in Loksa et al.'s Framework	Intervention	Number of Students Involved in Experiment	Year
Prather et al.	First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts	1	Solving test cases	36	2019
Denny et al.	A Closer Look at Metacognitive Scaffolding: Solving Test Cases Before Programming	1	Solving test cases	976	2019
Craig et al.	Answering the Correct Question	1	Solving test cases	831	2019
Janzen et al.	Test-driven learning in early programming courses	1	Writing tests cases	140	2008
Weinman et al.	Improving Instruction of Programming Patterns with Faded Parsons Problems	2	Faded Parsons problems	237	2021
Garcia	Evaluating Parsons Problems as a Design-Based Intervention	3	Design-level Parsons problems	10	2018
Prather et al.	On Novices' Interaction with Compiler Error Messages: A Human Factors Approach	5	Enhanced compiler error messages	31	2017

2.2. Interventions related to "Reinterpreting the problem prompt"

Stage (1) of Loksa et al.'s problem-solving framework corresponds to problem comprehension. Studies of interventions that aim to increase problem comprehension in students typically involve test cases. These interventions can be divided into solving test cases (Section A) and writing test cases (Section B).

A. Solving test cases

A study by Prather et al. involved a controlled experiment to measure the effect of solving test cases before coding on student metacognitive development, using an automated assessment tool (AAT) [9]. Students in the experimental group were required to solve and pass a test case quiz before coding, requiring them to reflect on their interpretation of the problem prompt. Students in the experimental group completed the task faster, at a higher completion rate, and required fewer attempts before completion. However, Prather et al. were not able to conclude the effect on the likeliness of solving the problem. They acknowledged that this was likely due to the small scale of the experiment (36 students).

Denny et al. conducted a similar study on a larger scale, with 976 students in a first-year programming course [10]. Students in each group used an online tool to solve a programming exercise and were shown a problem statement. Students in the control group were immediately shown the code editor on the same page of the online tool. Students in the experimental group were prompted to solve questions before the code editor was shown, though they did not have to answer correctly to proceed. Denny et al. did not find any differences in completion time, number of submissions, or the total error number between the control and experimental groups. However, they did find that the experimental group made fewer errors caused by an incorrect mental model of the problem. This indicates that solving test cases is successful in helping students' problem comprehension.

Craig et al. conducted a study on the effect of solving test cases before solving programming problems on submission counts [11]. Students were split into two treatment groups and received different programming exercises to solve. The treatment groups were then further split into control and experimental groups. Students in the experimental group were required to solve three test cases before being allowed to code, which acted as the problem-solving intervention. Craig et al.'s students were required to correctly answer test cases before they were shown the code editor. In contrast, Denny et al.'s students only needed to submit an answer to view the code editor; the answer did not necessarily have to be correct. They found successful results in one treatment group, where the correct problem solution relied on a solid understanding of boundary conditions. In this treatment group, the students who received the intervention required fewer submissions to correctly solve the problem compared to the control group. This indicates that solving test cases is beneficial for

problems where the difficulty lies in understanding rather than coding. Problems where programming language or library understanding is the greatest challenge, are not well suited to the test-case solving intervention.

B. Writing test cases

A study by Janzen et al. investigated the effect of requiring students to write test cases before they wrote a solution to a problem [12]. This study involved students in CS1 and CS2 courses who were taught Test-Driven Development (TDD). Students were given two programming projects for which they were required to write tests. Students were required to adhere to a TDD approach for any one of the two projects. For the CS1 group, they found no difference in grades or productivity between students who wrote tests before they wrote solutions and students who wrote tests afterward. However, for the CS2 group, they did find a difference, where students who wrote tests first achieved higher grades and spent less time on the solution code. The higher grades and faster development time indicate greater problem comprehension. Janzen et al. concluded that requiring students to write their own tests is an effective intervention to increase problem comprehension for more mature students, however, may not be suitable for novice students.

Writing test cases, as in Janzen et al.'s study, can directly assist students in stage (1) of Loksa et al.'s problem-solving framework. However, it is also important to note that these test cases can also later be used by the student in stage (6) – "Evaluating an implemented solution".

2.3. Interventions related to "Searching for analogous solutions"

Weinman et al. [13] investigated the use of Faded Parsons problems as a method to teach reusable programming patterns. Faded Parsons problems are where students are given a block of code, with lines in the wrong order and containing blank spaces. The students rearrange these lines of code and fill in the blank spaces with expressions. The addition of the blank spaces in lines is what distinguishes Faded Parsons problems from Parsons problems. This study involved three experiments in an introductory programming course, which involved solving Faded Parsons problems, code writing problems, and code tracing problems. Students' improvement in learning reusable patterns was measured through three metrics. Pattern Exposure measured whether students were first exposed to programming patterns through viewing instructor solutions (Faded Parsons problems or code-tracing exercises) or through writing their own code. Active Pattern Exposure measured whether students' solutions adhered to a programming pattern. Pattern Acquisition measured whether students applied programming patterns they had learned in previous exercises to subsequent code-writing exercises. The authors found that Faded Parsons problems have a higher rate of Pattern Exposure, Active Pattern Exposure, and Pattern Acquisition than code-writing problems. This indicates that Faded Parsons problems are effective in teaching students reusable programming patterns and how to apply them to code-writing exercises. Further, Faded

Parsons problems are more effective in teaching these patterns than traditional code-writing exercises and code-tracing problems and are preferred by students.

Though the aim of this study was not specifically related to increasing problem-solving skills in novices, it is most relevant to the "Searching for analogous solutions" stage of Loksa et al.'s framework. Faded Parsons problems could be used as an intervention by providing students with problems where the pattern is analogous to the model solution.

2.4. Interventions related to "Searching for solutions"

Garcia investigated the use of Parsons problems as an intervention to assist students with designing solutions [14]. Parsons problems are where students are given fragments of code in the wrong order and asked to arrange them to form a solution. In this experiment, rather than being provided fragments of code, students were given fragments of the algorithm's strategy as sentences. Arranging these fragments prompted students to design a solution. Data was collected through an initial self-assessment, think-aloud sessions for 3 assignments, and a final interview per assignment. The students solved the Parsons problems in the think-aloud sessions a lab-based environment and were given the fragments on pieces of paper. They found that Parsons problems were an effective intervention to help students develop higher-quality solutions. However, it is important to note that the scale of this experiment was very small. Another limitation of this intervention is the low scalability, as physical fragments were provided, rather than online or through an integrated development environment (IDE).

2.5. Interventions related to "Implementing a solution"

Prather et al. conducted a study on the helpfulness of enhanced compiler error messages in an automated assessment tool (AAT) which involved 2 experiments [15]. The first experiment was a controlled experiment that involved daily quizzes. These daily quizzes contained a program with errors and compiler error messages, and students were required to solve the error. The students in the experimental group received enhanced compiler error messages, where error messages had wording and structure better suited to novices. They found that students in the experimental group had significantly fewer incorrect answers. The last experiment was a think-aloud session where students solved a programming problem under time constraints. They found that students performed worse on the given problem than students in other semesters who did not receive enhanced compiler error messages. They suggested that these results could be due to students being able to solve this problem offline in previous semesters. Students' verbalizations indicated that they believed the enhanced error messages were useful. The results of these experiments indicate that enhanced compiler error messages are an effective intervention for stage (5) of Loksa et al.'s problem-solving framework.

3. Conclusion

All programmers require problem-solving skills to apply their programming knowledge to solve a given problem. Despite this, many students in introductory programming courses lack problem-solving skills. This could be a consequence of problem-solving strategies not being explicitly taught in introductory programming courses. Though even when explicitly taught, problem-solving can be hard for students to learn, as it requires metacognitive awareness.

This literature review investigated multiple problem-solving frameworks specific to programming, and interventions used to teach stages of problem-solving to novice students. Loksa et al.'s problem-solving framework was used to classify the reviewed literature around problem-solving interventions, due to its' focus on metacognition. Overall, there have been studies of interventions related to stages (1), (2), (3), and (5) of Loksa et al's framework, of which most were successful. However, there is a lack of work surrounding interventions related to stage (4) – "Evaluating a potential solution", and stage (6) – "Evaluating a solution". Additionally, all studies focus on one stage of the framework, rather than multiple.

4. Research Intent

The conducted literature review highlighted that all existing work surrounding automated interventions to teach problem-solving skills to novices is limited to a single stage of a problem-solving framework at a time. It is also highlighted that these interventions individually are effective at teaching students how to problem solve, increasing their metacognitive awareness and performance in their courses. Therefore, we feel that there is an opportunity to build a tool that can cater to multiple stages of a problem-solving framework at once. There is the potential that combining multiple problem-solving stages into an IDE will make it a more effective tool for teaching problem-solving and metacognitive awareness than a tool that only integrates one. Through this project we will seek to build a pedagogic IDE with problem-solving and metacognitive interventions, to answer the following research questions:

RQ1: *To what extent can a pedagogic IDE offer features that explicitly target distinct steps in a problem-solving process?*

RQ2: *Will building metacognitive and problem-solving interventions into a pedagogic IDE lead to students developing their metacognitive and problem-solving skills?*

As part of this project, we will build an IDE with relevant problem-solving and metacognitive interventions and conduct an experiment on students in an introductory programming course to answer the research questions above. Fig. 1 shows a timeline including all key deliverables which we will need to meet throughout the project.

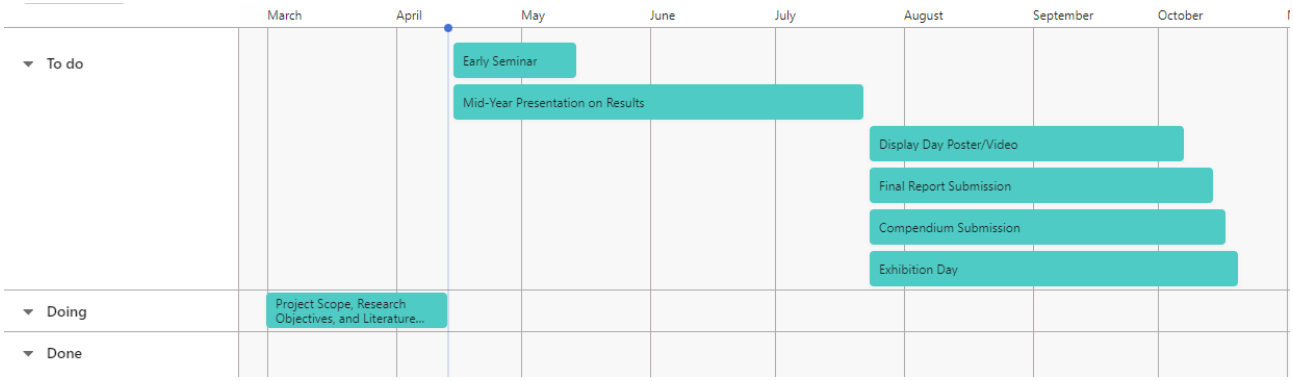

Fig. 1. Timeline of deliverables.

References

[1] R. Lister *et al.*, "A multi-national study of reading and tracing skills in novice programmers," *SIGCSE Bull.*, vol. 36, no. 4, pp. 119–150, 2004, doi: 10.1145/1041624.1041673.

[2] T. Beaubouef and J. Mason, "Why the high attrition rate for computer science students: some thoughts and observations," *SIGCSE Bull.*, vol. 37, no. 2, pp. 103–106, 2005, doi: 10.1145/1083431.1083474.

[3] M. d. Raadt, R. Watson, and M. Toleman, "Teaching and assessing programming strategies explicitly," presented at the Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95, Wellington, New Zealand, 2009.

[4] O. Muller, D. Ginat, and B. Haberman, "Pattern-oriented instruction and its influence on problem decomposition and solution construction," presented at the Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, Dundee, Scotland, 2007. [Online]. Available: https://doi.org/10.1145/1268784.1268830.

[5] Merriam-Webster, "Metacognition.," in *Metacognition.*, ed.

[6] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance," presented at the Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, California, USA, 2016. [Online]. Available: https://doi.org/10.1145/2858036.2858252.

[7] A. D. Hilton, G. M. Lipp, and S. H. Rodger, "Translation from Problem to Code in Seven Steps," presented at the Proceedings of the ACM Conference on Global Computing Education, Chengdu,Sichuan, China, 2019. [Online]. Available: https://doi.org/10.1145/3300115.3309508.

[8] M. McCracken *et al.*, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," presented at the Working group reports from ITiCSE on Innovation and technology in computer science education, Canterbury, UK, 2001. [Online]. Available: https://doi.org/10.1145/572133.572137.

[9] J. Prather *et al.*, "First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts," presented at the Proceedings of the 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, USA, 2019. [Online]. Available: https://doi.org/10.1145/3287324.3287374

[10] P. Denny, J. Prather, B. A. Becker, Z. Albrecht, D. Loksa, and R. Pettit, "A Closer Look at Metacognitive Scaffolding: Solving Test Cases Before Programming," presented at the Proceedings of the 19th Koli Calling International Conference on Computing Education Research, Koli, Finland, 2019. [Online]. Available: https://doi.org/10.1145/3364510.3366170.

[11] M. Craig, A. Petersen, and J. Campbell, "Answering the Correct Question," presented at the Proceedings of the ACM Conference on Global Computing Education, Chengdu,Sichuan, China, 2019. [Online]. Available: https://doi.org/10.1145/3300115.3309529.

[12] D. Janzen and H. Saiedian, "Test-driven learning in early programming courses," presented at the Proceedings of the 39th SIGCSE technical symposium on Computer science education, Portland, OR, USA, 2008. [Online]. Available: https://doi.org/10.1145/1352135.1352315.

[13] N. Weinman, A. Fox, and M. A. Hearst, "Improving Instruction of Programming Patterns with Faded Parsons Problems," presented at the Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Yokohama, Japan, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445228.

[14] R. Garcia, "Evaluating Parsons Problems as a Design-Based Intervention," presented at the *2021 IEEE Frontiers in Education Conference (FIE)* , 2021, Lincoln, NE, USA, 2021.

[15] J. Prather *et al.*, "On Novices' Interaction with Compiler Error Messages: A Human Factors Approach," presented at the Proceedings of the 2017 ACM Conference on International Computing Education Research, Tacoma, Washington, USA, 2017. [Online]. Available: https://doi.org/10.1145/3105726.3106169.