Program Comprehension: Identifying Learning Trajectories for Novice Programmers

Cruz Izu
The University of Adelaide
Adelaide, Australia
cruz.izu@adelaide.edu.au

Carsten Schulte
Paderborn University
Paderborn, Germany
carsten.schulte@uni-paderborn.de

Ashish Aggarwal
University of Florida
Florida, USA
ashishjuit@ufl.edu

Quintin Cutts
University of Glasgow
Glasgow, UK
quintin.cutts@glasgow.ac.uk

Rodrigo Duran
Aalto University
Helsinki, Finland
rodrigo.duran@aalto.fi

Mirela Gutica
British Columbia Institute of Technology
Burnaby, Canada
mirela_gutica@bcit.ca

Birte Heinemann
Paderborn University
Paderborn, Germany
birte.heinemann@uni-paderborn.de

Eileen Kraemer
Clemson University
Clemson, USA
etkraem@clemson.edu

Violetta Lonati
University of Milan
Milan, Italy
lonati@di.unimi.it

Claudio Mirolo
University of Udine
Udine, Italy
claudio.mirolo@uniud.it

Renske Weeda
Radboud University
Nijmegen, Netherlands
renske.smetsers@science.ru.nl

ABSTRACT

This working group asserts that Program Comprehension (PC) plays a critical part in the writing process. For example, this abstract is written from a basic draft that we have edited and revised until it clearly presents our idea. Similarly, a program is written in an incremental manner, with each step being tested, debugged and extended until the program achieves its goal.

Novice programmers should develop their program comprehension as they learn to code, so that they are able to read and reason about code while they are writing it. To foster such competencies our group has identified two main goals: (1) to collect and define learning activities that explicitly cover key components of program comprehension and (2) to define possible learning trajectories that will guide teachers using those learning activities in their CS0/CS1 or K-12 courses.

We plan to achieve these goals as follows:

Step 1 – Review the current state of research and development by analyzing literature on classroom activities that improve program comprehension.
Step 2 – Concurrently, survey lecturers at various institutions on their use of workshop activities to foster PC.
Step 3 – Use the outputs from both activities to define and conceptualize what is meant by PC in the context of novice programmers.
Step 4 – Catalog learning activities with regard to their prerequisites, intended learning outcomes and additional special characteristics.
Step 5 – Catalog learning activities with regard to their prerequisites, intended learning outcomes and additional special characteristics.
Step 6 – Develop a map of learning activities and thereby also models of probable learning trajectories.

KEYWORDS

program comprehension; learning trajectories; CS1;

ACM Reference Format:
Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, and Renske Weeda. 2019. Program Comprehension: Identifying Learning Trajectories for Novice Programmers. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland UK*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3304221.3325531

Related Work

Learning to program is not just about mastering the syntax and semantics of each construct of a programming language. From the outset, Soloway identified two key issues for learning to program: the ability to identify *chunks* (he renamed them plans) and the understanding of the way "the computer turns a static program written on a piece of paper into a dynamic entity that exists over time. In this dynamic representation, notions such as the causal relationship between the statements become very important and must be used in describing how a program works"[13, p. 854]. The latter issue is usually referred to in the literature as the notional machine [2].

Comprehension is usually conceptualized as a process in which an individual constructs his or her own mental representation of the program. Most novice programmer's misconceptions and logical errors are caused by poor understanding of that notional machine. Multiple models of program comprehension have been proposed in the literature [8, 16]. The 2010 WG report [11] compared and contrasted the way in which those models conceptualize program comprehension. Although this comparison was the main focus of the report, it also provided some insights into learning concepts and obstacles, effective learning tasks and teaching methods. Thus, this working group is picking up the baton by planning to collect and organize learning tasks that will develop program comprehension.

In the last ten years there has been a rising awareness and focus on program comprehension as part of learning to program. Assessment components have been proposed to include or target some aspects of program comprehension like reading [1], tracing [7], explaining [6], or reversing [3, 15]. Sudol et al. [14] vindicates using code comprehension questions as learning events instead of as assessment items. On a similar line, [12] selected 14 program comprehension tasks and survey practitioners to rank them in terms of perceived effectiveness in developing the novices' program comprehension.

Learning trajectories (LT) have garnered the attention of math and science educators [4], because their ability to model how the student's thinking about a specific topic evolves over time, and hence supporting (in the end) research based curriculum development. This is possible due to increased empirical knowledge about this progression - a knowledge currently lacking in computer science education. One reason is that there are seldom efforts to collect and systematize about such progressions in a domain; Rich et al [9] is an example of such approach in K-6 education.

The WG aims to extend our practical knowledge in the domain of program comprehension by collecting instructors' views of learning PC. We are therefore building on what Lobato and Walters have named the *hypothetical learning trajectory*; because the trajectory is seen through the eyes of the instructor [4, p. 84]:

> the starting point in teacher planning is the creation of conjectures regarding what students understand initially and what they may be able to learn next. Instructional tasks are selected, not only on the basis of generic task features, such as high cognitive demand or student interest, but also because of an inferred quality of being able to engender the next level of sophistication of student thinking.

The focus on how different tasks organize the thinking process of learners, and how tasks can and should be ordered to allow effective learning progressions is what distinguishes our WG from other approaches that have aimed at collecting useful examples and tasks (e.g. [10], [5]). In other words, we are not focusing on how to assess but looking at how to develop program comprehension. As mentioned before, assessment questions that target program comprehension can be extended to become learning activities. Thus, those previous collections of tasks will be analyzed to identify both tasks that require program comprehension and assessments that evaluate it.

REFERENCES

[1] Teresa Busjahn and Carsten Schulte. 2013. The Use of Code Reading in Teaching Programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13)*. ACM, New York, NY, USA, 3–11. https://doi.org/10.1145/2526968.2526969

[2] Benedict du Boulay. 1986. Some Difficulties of Learning to Program. *J. of Educational Comput. Research* 2, 1 (1986), 57–73.

[3] Cruz Izu, Claudio Mirolo, and Amali Weerasinghe. 2018. Novice Programmers' Reasoning About Reversing Conditional Statements. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, USA, 646–651. https://doi.org/10.1145/3159450.3159499

[4] Joanne Lobato and C David Walters. 2017. *A Taxonomy of Approaches to Learning Trajectories and Progressions.* NCTM, 74–101.

[5] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2017. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports (ITiCSE-WGR '17)*. ACM, New York, NY, USA, 47–69. https://doi.org/10.1145/3174781.3174784

[6] Laurie Murphy, Renée McCauley, and Sue Fitzgerald. 2012. 'Explain in Plain English' Questions: Implications for Teaching. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, USA, 385–390. https://doi.org/10.1145/2157136.2157249

[7] 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, New York, NY, USA, 2–11. https://doi.org/10.1145/3105726.3106178

[8] Nancy Pennington. 1987. Comprehension Strategies in Programming. In *Empirical Studies of Programmers: Second Workshop*, Gary M. Olson, Sylvia Sheppard, and Elliot Soloway (Eds.). Ablex Publishing Corp., Norwood, NJ, USA, 100–113.

[9] Kathryn M. Rich, Carla Strickland, T. Andrew Binkowski, Cheryl Moran, and Diana Franklin. 2017. K-8 Learning Trajectories Derived from Research Literature: Sequence, Repetition, Conditionals. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 182–190. https://doi.org/10.1145/3105726.3106166

[10] Kate Sanders, Marzieh Ahmadzadeh, Tony Clear, Stephen H. Edwards, Mikey Goldweber, Chris Johnson, Raymond Lister, Robert McCartney, Elizabeth Patitsas, and Jaime Spacco. 2013. The Canterbury QuestionBank: Building a Repository of Multiple-choice CS1 and CS2 Questions. In *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports (ITiCSE -WGR '13)*. ACM, New York, NY, USA, 33–52. https://doi.org/10.1145/2543882.2543885

[11] Carsten Schulte, Tony Clear, Ahmad Taherkhani, Teresa Busjahn, and James H. Paterson. 2010. An Introduction to Program Comprehension for Computer Science Educators. In *Proceedings of the 2010 ITiCSE Working Group Reports (ITiCSE-WGR '10)*. ACM, New York, NY, USA, 65–86. https://doi.org/10.1145/1971681.1971687

[12] A. Shargabi, S. A. Aljunid, M. Annamalai, S. M. Shuhidan, and A. M. Zin. 2015. Tasks that can improve novices' program comprehension. In *2015 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)*. 32–37. https://doi.org/10.1109/IC3e.2015.7403482

[13] E. Soloway. 1986. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (Sept. 1986), 850–858. https://doi.org/10.1145/6592.6594

[14] Leigh Ann Sudol-DeLyser, Mark Stehlik, and Sharon Carver. 2012. Code Comprehension Problems As Learning Events. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 81–86. https://doi.org/10.1145/2325296.2325319

[15] Donna Teague and Raymond Lister. 2014. Programming: Reading, Writing and Reversing. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, USA, 285–290. https://doi.org/10.1145/2591708.2591712

[16] Susan Wiedenbeck and Vennila Ramaligan. 1999. Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies* 51, 1 (1999), 71 – 87. https://doi.org/10.1006/ijhc.1999.0269