Empirical Assessment of Program Comprehension Styles in Programming Language Paradigms

Niloofar Mansoor
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska, USA
niloofar@huskers.unl.edu

I. Introduction

Developers work with different programming languages and tools throughout their careers. It is a critical skill to be able to build on existing skills and knowledge and learn new programming languages as needed. This makes exploring how developers learn and comprehend different types of programming languages an interesting problem. *The research question I plan to address with my research is: how do developers' mental model change when they learn and understand code written in different families of programming languages?* My research goals are to leverage empirical software engineering and cognitive sciences to understand learning and program comprehension in developers for answering this research question. I do this by conducting empirical studies on comprehension patterns of developers of varying skill levels using different language paradigms (imperative, declarative, and functional) while they work on a varied set of software tasks such as bug fixes, verification of static analysis alarms, adding new features, code refactoring, and code summarization. The proposed empirical studies are designed using a combination of online questionnaires and biometric equipment (eye trackers) and are performed on both program comprehension and a set of established cognitive tasks with the aim of determining whether there is indeed a relationship between these different tasks and domains on performance. The eye tracking biometric measures provide fine grained details on what tokens/words in code/text developers look at as they work. This better explains the thought process and mental models developers use to solve tasks. I propose multiple studies that will use both students and professional developers in order to understand the strategies of different levels of expertise. In addition, various other factors such as native language, reading speed, years of experience, programming expertise, cognitive expertise (among others) will be used to further describe the data collected on tasks.

II. Limitations of the Current State of the Art in Program Comprehension Studies

Besides the recently published eye movements in programming (EMIP) dataset [1], which is one of the largest eye movement datasets on programming, there are not a lot of publicly available datasets to work with to build theories on programmer expertise or comprehension strategies. The EMIP dataset uses small snippets of code that are not realistic and pose a threat to external validity. The proposed research will directly address this problem by providing additional datasets of fine grained programmer behavior on realistic open source applications within realistic settings such as Integrated Development Environments. In addition, the proposed research will test a variety of programming paradigms to provide evidence of any differences in cognitive load via objective biometric measures.

Software developers work with many source code files and each of these files can contain thousands of lines of code, and usually, developers flip through different files while working on code. State-of-the-art eye tracking software is only capable of tracking programmers gazes on short code snippets that are usually presented as images fixed on the screen - an unrealistic setting. iTrace [2], is the eye tracking infrastructure developed by our research team that solves these issues by connecting to IDEs and/or the Google Chrome browser to map the coordinates of eye gaze to the elements of the code and other software artifacts. iTrace maintains accurate gaze mapping even when the developer scrolls through the pages or changes the file they are reading through. Using iTrace in our empirical studies will help us better identify comprehension patterns and will give us a lot of useful data to explore developers' behaviors using a variety of metrics while they work in realistic settings.

III. Overview of Proposed Work

Each designed study considers different sets of research questions all linked under the common theme of understanding how developers work in a variety of tasks. I plan on conducting these studies using a mixed methods design. I will use quantitative and qualitative data analysis methods and data triangulation techniques to confirm hypotheses.

These proposed research projects are designed to help understand cognitive patterns and abilities of developers, and their program comprehension in different types of programming languages while working on different programming tasks.

All of the following studies are also accompanied by cognitive tasks for the developers to measure their cognitive skills and abilities such as working memory, attention and executive functioning, pattern recognition, spatial ability, etc. There is recent work that links some of these abilities to specific skills in programming and problem solving [3], and the results of these cognitive tasks will give us a chance to further explore these relationships and give us an insight into the developers' problem solving and cognitive abilities.

IV. Preliminary Study: An Exploratory Study Assessing the Alloy Specification Language

This study's goal is to explore developers' comprehension patterns not only in prevalent imperative programming languages such as Java [4], but also on declarative or domain specific languages. Studying the comprehension of different types of languages helps us determine strong and weak features and will ultimately improve programming language design. One of the languages I chose for investigating developers' comprehension patterns is the Alloy specification language. Alloy is a declarative language designed for expressing structural and behavioral constraints in software systems. Alloy is an example of a declarative language that is mostly used in academia and studying the learning patterns of developers when they are getting familiar with a language similar to Alloy gives us a lot of insight about the comprehension of the language itself. I conducted this study via an online questionnaire done by 25 participants. They were asked to find and fix syntactic and semantic bugs in Alloy models. We aimed to find out how novice and non-novice users of Alloy work with the language while solving the tasks. We found that overall, non-novices find and fix Alloy specification issues with 58% more accuracy compared to novices, and on average solve the tasks 27 minutes faster compared to novices. Our results also show that on average, non-novices who performed better on the Alloy tasks had higher mental rotation scores, which indicates the importance of spatial cognition ability in solving Alloy tasks.

V. Future Work

A. Program Comprehension in the Functional Programming Paradigm

This empirical study (via online surveys and in-lab eye tracking equipment) will assess comprehension of functional programming languages (such as Haskell), which are a specific subset of declarative languages. Investigating how developers understand and use functional languages and comparing the patterns with other declarative and imperative languages can give us more insight into general programming language comprehension, and how the structure and concepts of the language affect comprehension in people with different levels of expertise. Studying functional languages is particularly interesting because of their unique characteristics, such as being designed on mathematical concepts, using expressions and recursive functions, and not using constructs such as for loops and if statements. Lack of these widely used constructs might greatly affect program comprehension. Furthermore, the lack of side effects in functional programming will also be an interesting concept to study when developers are looking to fix bugs in functional programs.

B. Assessing Static Alarm Warning Messages

The second study I plan on conducting is on assessing static alarms repositioning in the C++ language. This is a collaborative project with two external sites. Static analysis tools generate a large number of alarms, and they need to be manually inspected by the users to separate false alarms from real errors. A considerable amount of manual effort is spent on this process. Automatic repositioning of these alarms has been proposed [5] to reduce the number of alarms and the manual effort of inspecting them. The reduction in manual effort is expected due to repositioning the alarms closer to their cause points. This empirical study of the realistic effects of this proposed method will be conducted in two stages. The online stage where a large number of participants can work on software tasks and use the static analysis tool that shows the alarms on various tasks in different positions (whether in a list or in a position close to their cause point), which will help us empirically determine if repositioning helps the users' spend less time and put less effort on finding real errors and false positives. Subsequently, a smaller group of developers and students will be recruited to perform the same tasks on computers that are equipped with an eye tracker on which we will use iTrace [2] to determine the elements they are examining while solving the tasks. This step of the project will help us better understand the developers' gaze and their cognitive effort and compare the proposed new method of repositioning with the traditional method of showing static alarms.

VI. Summary

The proposed empirical studies on program comprehension are to understand the comprehension patterns and cognitive abilities of developers at different skill levels working on different types of programming languages. The results of the proposed studies in this proposal will advance the program comprehension sub-field of software engineering by providing evidence-based results for future tools. They also pave the way towards understanding how theories of cognitive psychology apply to software engineering.

References

[1] R. Bednarik, T. Busjahn, A. Gibaldi, A. Ahadi, M. Bielikova, M. Crosby, K. Essig, F. Fagerholm, A. Jbara, R. Lister *et al.*, "Emip: The eye movements in programming dataset," *Science of Computer Programming*, vol. 198, p. 102520, 2020.

[2] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, "itrace: Eye tracking infrastructure for development environments," in *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, 2018, pp. 1–3.

[3] T. Baum, K. Schneider, and A. Bacchelli, "Associating working memory capacity and code change ordering with code review performance," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1762–1798, 2019.

[4] N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, and J. I. Maletic, "Developer reading behavior while summarizing java methods: Size and context matters," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 384–395.

[5] T. Muske, R. Talluri, and A. Serebrenik, "Repositioning of static analysis alarms," in *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*, 2018, pp. 187–197.