Minhyuk Ko | Research Statement

(703) 507-1177 • minhyukko@vt.edu • people.cs.vt.edu/minhyukko

The rapid development of Artificial Intelligence (AI) has transformed the way programmers write code. As AI programming tools become more proficient, it is estimated that approximately 80% of code will be written with AI in the future. While this will help democratize software development, it also comes with unprecedented challenges. For example, because developers frequently do not write code on their own, they spend more time on code review than on code implementation, such as bug fixing.

My research exists at the intersection of Software Engineering (SE), Computer Science Education, and Human-Computer Interaction, where **I develop tools that assist students with debugging.** Though generative AI is capable of generating code, debugging AI-generated code is still largely performed by humans. My tools aim to lower barriers to effective debugging by making program execution and behavior more accessible and intuitive to novice programmers, without requiring them to navigate the complexities of traditional debugging tools.

My work has been published in a wide range of conferences, including CHASE, EASE, ICSE, SERS, and VL/HCC. I received the **Best Short Paper Honorable Mention** at VL/HCC 2023 and the **Distinguished Paper Award** at ICSE 2023. I have 6 first-author papers and 2 co-authored works. My research has been partially supported by the Alfred P. Sloan Foundation, Dr. Richard Nance Graduate Fellowship Award, and Preston and Catharine White Family Graduate Fellowship. The following sections outline my past, ongoing, and future research.

Survey Study on Debuggers

To develop tools that assist students with debugging, it was critical to understand their debugging behaviors. To understand factors that influence decisions to use debugging tools or manual print statements, I conducted a survey and received responses from 73 software engineering students with varying degrees of programming experience. The results suggest that effort and lack of education are the main barriers to debugger usage, despite success and positive experiences when using debuggers to resolve bugs. This work contributes a detailed analysis of students' debugging behaviors and decisions, and the findings provide implications for improving debugging tool development and programming education to increase knowledge and usage of debuggers for automating debugging tasks. The results from this study motivate the design of my follow-up work, *AutoPrint*.

AutoPrint: An Automatic Print Statement Insertion Tool

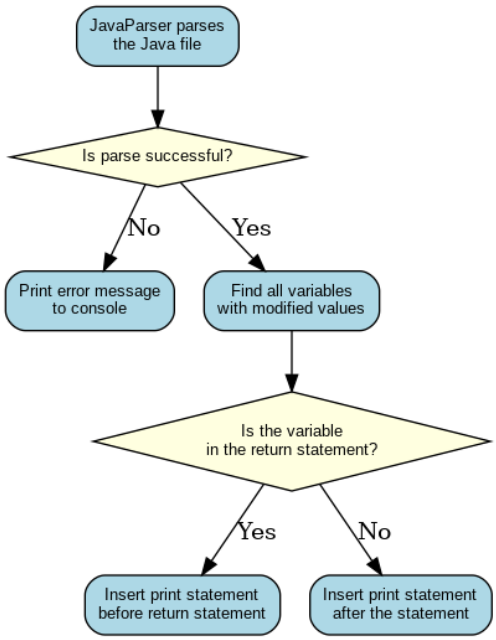


Figure 1: Improved AutoPrint Diagram

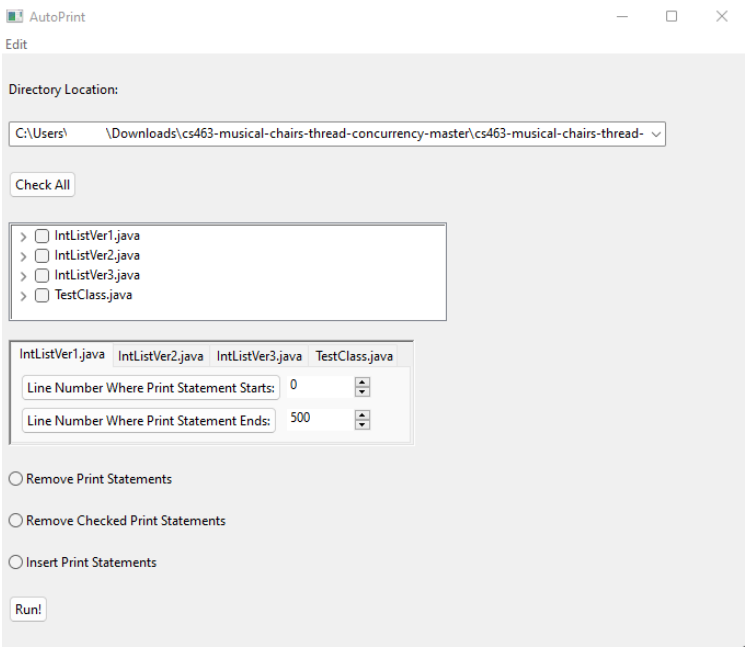


Figure 2: AutoPrint Image

In this work, I propose a novel debugging tool that automates print statement debugging called *AutoPrint* [1]. My tool automatically inserts and removes debugging-related print statements in code. *AutoPrint* uses a program's abstract syntax tree (AST) to detect Java variables and automatically insert a print statement to display the variable's value on the following line of code. An overview of the *AutoPrint* workflow is provided in Figure 1. *AutoPrint* asks users for information needed to automate print statement debugging in our tool, as depicted in Figure 2. In addition, users can specify where to insert print statements and select which variables to print.

To evaluate the effectiveness of *AutoPrint*, I conducted a judgment study through which 23 participants with Java programming experience—students and industry practitioners—used *AutoPrint* for practical debugging tasks in diverse real-world projects. I elicited feedback through a post-survey and semi-structured interviews, and the results indicate that *AutoPrint* was able to help programmers debug Java source code. Participants praised *AutoPrint*'s simplicity, speed, and flexibility, overall finding it effective for debugging code in practical settings.

The findings provide insight into debugging practices and enhancing debugging tools. *AutoPrint* is not intended to replace debuggers, but to help programmers who use print statement debugging. The main contributions of this project are as follows: (a) I propose a new debugging tool *AutoPrint* that automates inserting or removing print statements for debugging; and (b) I evaluated the tool among students and industry practitioners in real-world software projects.

Designing Tools to Enhance Best Practices in Research Software Engineering (RSE)

I recruited Research Software Engineers (RSEs) [2] and conducted two participatory design (PD) workshops to identify challenges and co-design solutions to support developing and maintaining research software [3]. This provided me with the opportunity to engage with RSEs, who develop and maintain research software without prior knowledge or formal SE training, to investigate human-centric aspects of research software development within a specific community. The PD workshop approach helped foster creative thinking and deeper insights by enabling participants to critically engage with one another and refine their understanding of RSE challenges.

The findings reveal that RSEs face common development challenges in implementing and debugging code, yet also face unique issues navigating different programming languages and adapting developers' mental models [4], such as typical development tools, activities, and practices. To overcome development challenges, participants proposed a variety of tools to increase awareness and adoption of SE practices. Based on the results, I motivate future work to innovate automated solutions for supporting the development and maintenance of research software by providing hands-on learning opportunities and enhancing the utilization of generative artificial intelligence (AI).

Helping RSEs Craft Better Prompts for Requirement Engineering Tasks

Based on the findings from the PD workshop study, I designed *PromptRSE*, a tool designed to help RSEs craft effective prompts that enable LLM to generate contextually tailored responses for their research software development tasks. In the current prototype implementation, *PromptRSE* incorporates empirically grounded guidelines for prompting LLMs during requirements engineering (RE) activities, as outlined by Arvidsson and Axell [5]. The tool provides feedback on prompt formulation, assisting users in refining their inputs to obtain results that better align with their intended goals.

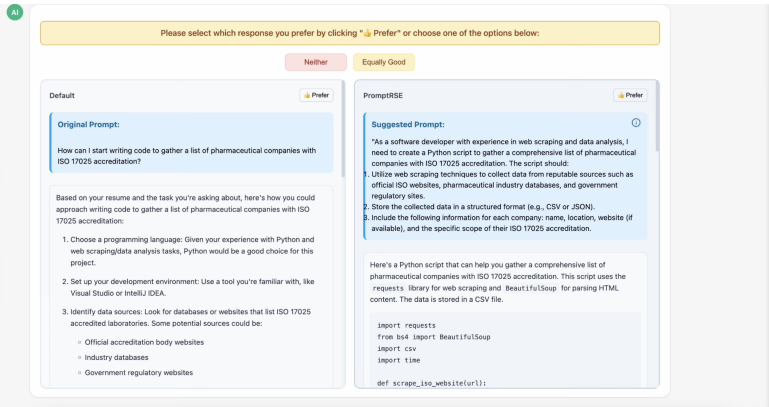


Figure 3: PromptRSE User Interface

I conducted a chatbot arena study, where 17 RSEs from diverse fields used *PromptRSE* to complete research SE tasks in their respective domains. I found participants preferred responses to prompts generated by *PromptRSE*, and post-survey results showed the tool was usable and easy to use, with participants expressing willingness to adopt it into their workflows. My research on *PromptRSE* provides encouraging evidence that prompt engineering tools can assist with RSEs' software engineering tasks. Based on the findings, I provide implications for future techniques to enhance LLM-assisted SE for programmers from diverse disciplines.

Future Work: Agentic Debugging for Novice Programmers

My research at Virginia Tech has focused on studying novice programmers' programming behavior. As a future faculty member, I envision myself working on this line of research with passionate undergraduate students who bring the perspective of a novice programmer, while I bring research expertise that I gained through my PhD. I expect the advisor-advisee relationship to complement each other in mutually enriching ways. I also envision applying the research findings directly into the classrooms. Here are a few possible directions of research that I am interested in pursuing:

AI Agents That Autonomously Support Student Debugging

Agentic AI systems are capable of assisting programmers in various software development tasks, such as interpreting error messages, generating code documentation, and repairing code. The high level of autonomy that agentic AI systems provide presents both opportunity and risk. For my future research, I plan to study how agentic debugging assistants can be designed in a way that is calibrated to the user's programming proficiency. For example, when the user is a novice programmer, the assistant should intervene just enough so they do not give up on debugging, but not so much that it deprives them of the productive struggles essential to skill development. My goal is to build an agentic debugging tool that calibrates its level of autonomy based on user interaction behavior, such as the difficulty of the questions they ask or the types of bugs they encounter. This tool will effectively reduce the workload of professors, especially at primarily undergraduate institutions where professors tend to teach 4-5 classes per year.

Guiding Novices Through Debugging Step-by-Step

I am interested in developing agentic debugging tools that guide students through the debugging process while helping them develop a developer's mental model. Although existing AI-powered debugging tools can identify bugs and suggest fixes, my envisioned approach focuses on teaching students how to debug rather than simply helping them fix bugs. Specifically, I envision an agent that observes students' debugging behaviors, encourages them to formulate and test their own hypotheses, and provides adaptive guidance based on their demonstrated understanding. Rather than immediately revealing solutions, the agent scaffolds students' debugging activities and gradually reduces its assistance as they develop more systematic debugging strategies. This approach would be particularly beneficial for novice programmers, such as students enrolled in Introduction to Computer Science. By emphasizing the development of transferable debugging mental models, my goal is to equip students with skills they can apply independently in advanced programming courses and beyond.

Integrating Agentic Debugging into CS Curricula

My study on debuggers shows that a lot of students do not use debuggers, as they have never learned how to use them in a formal classroom setting [6]. To address this gap, I plan to develop pedagogical resources that help instructors integrate agentic debugging tools into CS curricula. These resources will include instructional materials, classroom activities, and assignments that encourage students to actively engage in the debugging process. I will conduct controlled classroom studies, where the control group uses the existing pedagogical framework, and the variation group uses resources that I designed, to understand how students from the variation group navigate the debugging process differently from the control group. I plan to make the pedagogical resources open-source to facilitate their adoption by CS educators at other institutions.

References

[1] **Minhyuk Ko**, Omer Ahmed, Yoseph Berhanu Alebachew, and Chris Brown. Autoprint: Judging the effectiveness of an automatic print statement debugging tool. In *2025 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 379–384. IEEE, 2025.

[2] **Minhyuk Ko**, Shawal Khalid, and Chris Brown. Finding rses: Challenges in recruiting research software engineers for research. In *Proceedings of the 2026 1st International Workshop on Software Engineering and Research Software*, pages 42–44, 2026.

[3] **Minhyuk Ko** and Chris Brown. Designing tools to enhance best practices in research software engineering. In *Proceedings of the 19th International Conference on Cooperative and Human Aspects of Software Engineering*, pages 267–271, 2026.

[4] Jenny T Liang, Maryam Arab, **Minhyuk Ko**, Amy J Ko, and Thomas D LaToza. A qualitative study on the implementation design decisions of developers. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 435–447. IEEE, 2023. **Distinguished Paper Award**.

[5] Simon Arvidsson and Johan Axell. Prompt engineering guidelines for LLMs in Requirements Engineering. August 2023. Accepted: 2023-08-07T08:59:27Z.

[6] **Minhyuk Ko**, Dibyendu Brinto Bose, Hemayet Ahmed Chowdhury, Mohammed Seyam, and Chris Brown. Exploring the barriers and factors that influence debugger usage for students. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 168–172. IEEE, 2023. **Best Short Paper Honorable Mention**.