A Study on Improving Static Analysis Tools:
Why are we not using them?

Brittany Johnson
Department of Computer Science
North Carolina State University, Raleigh, USA
Email: bijohnso@ncsu.edu

Abstract—Using static analysis tools for automating code inspections can be extremely beneficial for software engineers. Despite the benefits of using static analysis tools, research suggests that these tools are underused. In this research, we propose to investigate why developers are not widely using static analysis tools and how current tools could potentially be improved to increase usage.

Keywords—static analysis; tool development; tool evaluation

I. Problem and Motivation

Software is present in many different aspects of our lives, making many new and innovative things possible. With such a high reliance on software, software quality is important. Many factors that can play a role in poor software quality, coding defects being one. Coding defects, or bugs, can cost companies significant amounts of money, especially if they are not found early in the development process, and potentially lead to software failures [1] [2]. One method for finding bugs in software is by using static analysis tools. Static analysis tools automate the process of inspecting code using well-defined programming rules. They also make it possible to find bugs early in the development process, when they are cheap to fix [3]. For example, there are static analysis tools that can alert developers of synchronization issues which can lead to unsafe thread interactions. Developers have been able to eliminate many bugs that were not found during testing at major companies such as Google [4] using the warnings produced by static analysis tools.

Despite the potential benefits of using static analysis tools, actual usage is not as high as expected [5] [6]. Our research aims to find out who does and does not use static analysis tools and why. We hope to be able to find ways to improve existing tools in order to increase usage. This way, developers will have access to a tool that is cheap and provides quick, valuable feedback while software users are able to enjoy their bug-free products.

II. Related Work

There have been many studies on static analysis tools, many of which focus on their correctness and functionality [5] [6] [7] [8]. Unlike existing work, our work focuses more on how developers feel about using static analysis tools, including interacting with the interface of the tool, and why.

A study done on the FindBugs static analysis tool is closely related to our work [3] [5]. In a study conducted by Ayewah and Pugh, they claimed that static analysis tools should help engineers find bugs as early as possible in the development cycle, when they are cheap to fix [3]. They surveyed and interviewed hundreds of FindBugs users to see how they use FindBugs and handle defects that are labeled “not a bug”. In our research, we hope to find out if and why static analysis tools are not widely used and ways to help increase their usage. Our work builds on this work, because we will be interviewing various tool users in an interactive, participatory manner while they mainly focused on FindBugs users in a controlled study. By conducting interactive and participatory interviews, we hope to build on their work by obtaining more detailed results about what developers are expecting from these tools and what they would like to see change about them.

Vorobyov and Krishnan conducted a case study on error detection approaches, specifically static analysis and model checking [8]. In their study, they compared the two methods of detecting errors by conducting an empirical evaluation on Parfait, a static analysis tool for C and C++, and CBMC, a model checker for ANSI C programs. They evaluated validity of their output among other factors. The results shows that although static analysis was faster, model checking was more accurate. Our work is related in that we are interested in whether developers are using other error detecting methods besides static analysis tools and why. Our work builds on this work because we are focusing on static analysis tools and their usage, while their study focused on when and when not to use static analysis.

The work done by Bessey et. al. can also be viewed in relation to our work [6]. The authors of this work reflect on their experiences in taking a research project on static analysis and commercializing it. They also make an interesting points concerning the kinds of static analysis that are appropriate for commercial use and the problems that a lack of programming language standardization causes. Here again, static analysis tools and their usage are being evaluated. This work is similarly related as the previous work to our work; it provides useful insights to consider in conducting our study.
III. Approach and Evaluation

For our approach, we plan to conduct semi-structured interviews with professional developers who have experience with static analysis tools in a professional environment. Conducting semi-structured interviews allows us flexibility in the script that will be used for each interview so we can follow the flow of the interviewee and get detailed answers instead of following the structure of an interview script [9]. We will use an electronic recruitment flyer for recruiting participants and give small gifts for compensation. Before the interviews, a short questionnaire will be sent to the participants in order to collect demographic information relevant to our study.

The interviews will focus on developers’ previous and recent experiences with static analysis tools and if and how they currently use them. Observing how developers use static analysis tools and learning developers’ relevant experiences may shed light on why these tools are not widely used. We also believe that the best way to develop a tool that developers want to use may be to observe how they use their tools and find out how they want the tool to behave. In order to conduct this study, we need a structure for our interviews that will get detailed feedback needed to evaluate static analysis tools, how developers use these tools and design and implement a new one that developers will want to use. We plan to organize the interviews into three main parts: Question and Short Response, Interactive Interview, and Participatory Design.

The purpose of the Question and Short Response portion is to get some background information on the interviewees’ general usage, understanding, and opinion of the static analysis tools that they have had experience with. During the Interactive Interview, we will ask the interviewee to run a static analysis tool of their choice on some code while we observe. The last portion of the interview is a form of participatory design where we ask the interviewee to tell or show us what they want in a static analysis tool [10]. Using participatory design is a way of giving the developer the opportunity to let us know exactly what they want in a static analysis tool and why. With this approach, we hope to learn developer work processes and use that information to develop a tool that will work with developer workflows.

We define a workflow as the steps a developer takes when writing, inspecting and modifying their code. To evaluate our interview approach, we have done practice interviews with students at our university. The students were not required to have industry experience with static analysis tools, but all of our participants were aware of what a static analysis tool is and had at least one experience with one. We conducted four preliminary interviews; two with the Interactive Interview portion, two with just the Question and Short Response and Participatory Design sections. In the interviews without the Interactive Interview, the Question and Short Response section was extended to include the questions that would have been asked during the Interactive Interview. The interviewees that participated in the interactive interviews were able to give more details about their workflows when using static analysis tools. In general, the students indicated that although they do feel using static analysis tools can be beneficial, they usually have to click something to invoke the tool. If the program is too large, running in the background seems to slow down the IDE, while if it is small, they do not want to be bothered with constant notifications and would rather run it when they see fit. This suggests that it may be beneficial for static analysis tools to be able to be more aware of the codebase on which it is being used in order to adapt to developers needs. In further interviews with software developers, we will determine whether this is important. Also, by having them use a static analysis tool prior to doing the Participatory Design portion, developers were also able to be more detailed about what they wanted in a static analysis tool.

An important part of this research is finding how we can help developers find bugs as early as possible, or provide “fast feedback”. Part of this involves finding out what developers define as “early”, or when they would like to be notified of bugs. For most of our interviewees, three out of four, being notified of bugs as you are typing is the best way of getting early reports of bugs. One participant, however, did not like the idea of being notified of bugs while typing his code. He would prefer to be notified of bugs when he compiles his code. There are a few compilers for C and C++, such as Clang, that have been extended with static analyzers. However in other popular languages, like Java, there are few if any tools that are or can be integrated with the compiler. Although there are compilers that can be extended to build static analysis tools, such as JastAdd, this takes time that developers may or may not be willing or able to take [11]. It may be worth investigating whether developers would like a static analysis tool that is, or can be, integrated with their compilers or IDEs.

In the near future, we plan to conduct interviews with industry developers, implement a static analysis tool prototype based off our results and conduct another study evaluating whether our tool meets developer needs and could actually increase the use of static analysis tools.

IV. Result and Contribution

The result of our work will be a way for developers to easily and quickly discover significant bugs early when they are cheap to fix. We expect that this will lead to better software quality, lower development costs, and easier testing.

Acknowledgment

I would like to thank Yoonki Song Emerson Murphy-Hill, and Robert Bowdidge for assistance with this research and Google for the funding.
REFERENCES


