Enhancing Tools' Intelligence for Improved Program Analysis Tool Usability

Brittany Johnson
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695-8206
Email: bijohnso@ncsu.edu

***Abstract*—Program analysis tools can help developers produce high quality code by automating time–consuming tasks such as error-finding. Research has shown, however, that these tools are often not used by developers. Results from studies I have conducted provide insights into the difficulties programmers may encounter when using program analysis tools, leading to lower productivity and desire to use them. Based on these findings, my dissertation will research improving program analysis tools by enhancing tool intelligence with a programmer model that adapts notifications to programmers based on their experience with the concepts relevant to the notification.**

I. Motivation & Background

Software is ubiquitous, which makes software quality an important consideration for software developers. Research has found that when programmers manually analyze or modify their code for quality it can take more time and even introduce errors into the code [1], [2]. Program analysis tools, such as FindBugs[1] and Eclipse built–in refactoring tools, are designed to automate these tasks for developers, potentially reducing the time, effort, and errors involved in developing software. However, research suggests programmers often do not use these tools and that this phenomena can be attributed to various factors, many of which pertain to tool or notification usability [3], [2], [4]. My dissertation research seeks to improve program analysis tools' ability to support programmers, by considering factors that affect programmers' desire to use, and productivity while using, these tools.

My research thus far is composed of three empirical studies: one interactive interview study, one classroom study, and one think–aloud study. For the interactive interview study, I asked general tool usage questions to 20 industry developers to learn about their tool experiences, observed their usage of FindBugs, and asked for tool design improvements [3]. For the classroom study, I gathered data from students' usage of various program analysis tools. For the think–aloud study, I observed 20 programmers with various levels of programming experience as they used three different program analysis tools. I discovered various difficulties, which I used to determine improvements program analysis tools can make to give programmers the support they need. I have used and will continue to use data from all three studies to develop my dissertation research. Data I have gathered suggests one reason developers may have for not regularly using program analysis tools is difficulty assessing, or sorting through, understanding, and addressing, tool notifications.

[1] findbugs.sourceforge.net

Much of recent research, stemming from similar findings, has focused on improving one part of the notification assessment process. For example, FaultBench attempts to ease the process of sorting static analysis notifications, while tools like HelpMeOut are meant to help programmers address or resolve compiler notifications [5], [6]. There even exist guidelines for improving the ability to understand compiler error notifications [7]. However, to my knowledge, no research has attempted to develop an approach that can be applied to any one or set of program analysis tools to better support programmers with the notification assessment process. My research suggests that one way to increase usability is to enhance tool intelligence, or afford notification presentation adaptations based on programmer knowledge, experience, and notification resolution patterns such that the problem is more clear to the individual programmer.

Based on the findings of my studies, my dissertation plan consists of 1) developing a model that creates approximations of programmer knowledge to understand the mappings needed for programmers to more quickly assess notifications, 2) developing notifications better suited to individual programmers' needs during the assessment process based on the model, and 3) implementation of a tool intelligence enhancement that refines and presents tool notifications to programmers.

II. Plan of Work

Programmer Model. The first step to enhancing tool intelligence is adding the ability to model a programmer's knowledge and experience. Intelligent tutoring systems (ITS) perform a process called "student modelling" to track the student's psychological state as they work and learn; this includes their skills, subject matter knowledge, and other identifying factors [8]. Each student's model is used to customize the activities, feedback, and hints they are given. Research has shown that programmers use experience and knowledge they have obtained when writing, reading or modifying source code [9], [10]. Based on these findings, I believe the notion of an effective notification depends on the knowledge and experience of the programmer; for example, one programmer may need more information regarding how a feature works while others just need to know why their usage of that feature is ineffective.

For my approach, I would perform "programmer modelling" where I would infer a programmer's knowledge and experience and use that information to adapt and augment exisiting tool notifications. I plan to collect information such as

978-1-4799-4035-6/14/$31.00 ©2014 IEEE

language features used and notifications successfully resolved or ignored. The model can be used to determine in advance if a user might struggle with the default notifications used by the tool and adapt them to increase the likelihood that the programmer can effectively assess each. To evaluate my model–building process, I plan to recruit 10–20 programmers, manually create a model for each, and have them perform model–assisted activities while comparing their performance with a control group.

Effective Notification Adaptations. Previous studies, including my own, have found that tool notifications often do not present information in a way that helps programmers quickly and effectively understand and resolve the notifications [3], [7], [11]. From the data gathered from my think–aloud study, I found that programmers sometimes percieve notifications as "incomplete," omitting details that particular programmer needed to understand it. If tools leveraged the programmer model to adapt notifications to each individual programmer, programmers may be able to more quickly assess each notification.

For example, take the text from a FindBugs notification:

> (a) This method synchronizes on an object referenced from a mutable field. (b) This is unlikely to have useful semantics, since different threads may be synchronizing on different objects. (1)

A more effective notification might augment (1) with explanations that better suit the programmer's knowledge concerning synchronization. For example, (1)(b) might be augmented with the following text:

> Multiple threads could access this method simultaneously, leading to the creation and locking of more than one instance of `object` – this is an ineffective way of using a synchronization block. (2)

(2) has been adapted for a programmer that has limited experience with synchronization and multi-threaded environments. Simply adding this additional text attempts to provide a description of the problem in a way that someone who has limited experience using synchronization might understand. For different programmers, notification (1) would include different explanations, or omit them all together, depending on the programmer model. To validate that this improves the assessment process, I plan to modify a set of notifications and conduct experiments where I present them to programmers and test their understanding. If this proves to be true, the next step is to automate this process and possibly include other additions such as code examples or visualizations to depict supplementary information such as relevant control flow or parts of the code.

Tool Intelligence Enchancement and Evaluation. The final piece to my dissertation will be automating the creation of programmer models and the presentation of effective notifications based on these models. I imagine a tool that (a) automatically examines programmers' code and calculates knowledge values for various features and concepts, (b) determines relevant features or concepts for a given notification, and (c) adapts tool notifications based on the programmer model. I plan to evaluate my tool enhancement holistically by having half of the programmers from the programmer model evaluation use the enhancement in a real–world setting for an extended period of time, while others do not. I will observe differences in productivity and evaluate the tool's usability with a post-study questionnaire.

Difficulties mentioned in my previous study also included sorting through tool notifications [3]. Future work might include augmenting programmer models with information regarding *when* the programmer cares about particular notifications or concepts. For example, a programmer's model may indicate that a programmer only assesses an unused variable notification before they leave that part of their code to work on another, whereas if they introduce an off–by–one error, they assess that notification immediately. Along the same lines, the model could be used to determine the best notifications to present when a programmer returns to their code after an extended break.

ACKNOWLEDGMENTS

Thanks to my advisor Dr. Emerson Murphy-Hill and co–advisor Dr. Sarah Heckman for their support. This material is based upon work supported by the National Science Foundation under Grant No. 1217700, a Google Faculty Award, and a National Science Foundation Graduate Research Fellowship under Grant No. DGE–0946818.

REFERENCES

[1] N. Ayewah, "Static analysis in practice," University of Maryland, Tech. Rep., 2010.

[2] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, 2012.

[3] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of International Conference on Software Engineering*, 2013, pp. 672–681.

[4] J. Pane and M. B., "Usability issues in the design of novice programming systems," Carnegie Mellon University, Tech. Rep., 1996.

[5] S. Heckman and L. Williams, "On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 41–50.

[6] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do: suggesting solutions to error messages," in *Proc. SIGCHI Conference on Human Factors in Computing*, 2010, pp. 1019–1028.

[7] V. J. Traver, "On Compiler Error Messages: What They Say and What They Mean," *Advances in Human-Computer Interaction*, 2010.

[8] A. C. Graesser, M. W. Conley, and A. Olney, "Intelligent tutoring systems," *APA handbook of educational psychology. Washington, DC: American Psychological Association*, 2012.

[9] S. Wiedenbeck, V. Fix, and J. Scholtz, "Characteristics of the mental representations of novice and expert programmers: an empirical study," *International Journal of Man-Machine Studies*, vol. 39, no. 5, pp. 793–812, 1993.

[10] J. J. Cañas, M. T. Bajo, and P. Gonzalvo, "Mental models and computer programming," *International Journal of Human-Computer Studies*, vol. 40, no. 5, pp. 795–811, 1994.

[11] M. Nienaltowski, M. Pedroni, and B. Meyer, "Compiler Error Messages: What Can Help Novices?" in *Proceedings of SIGCSE Technical Symposium on Computer Science Education*, 2008, pp. 168–172.