Adapting Program Analysis Tool Notifications to the Individual Developer

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

***Abstract*—There are a variety of tools available for developers to use in their IDEs. Research suggests, however, developers may not use these tools due to difficulty interpreting the output. My research explores the possibility of creating frameworks that enable more individualized program analysis tool notifications for increased usability. I propose creating models that represent what developers know about programming concepts using exisitng developer data. These models could then be used to inform tools on notifications the developer may or may not understand.**

I. Introduction

A variety of program analysis tools are made available to developers to automatically develop and analyze code. Automating the software development process is less tedious and helps increase code quality [2], [9]. FindBugs, for example, is a static analysis tool designed to be used by any Java developer to find potential defects in their source code. However, research suggests that developers are not always able to parse and understand the output provided by tools like FindBugs [8].

Many tools, like FindBugs, attempt to provide detailed information, in the form of *notifications*, to help developers when diagnosing potential defects in their code. However, there is typically no consistency as to what notifications provide more or less detail. For example, for most program analysis tool notifications, level of detail, availability of resources, and usage of examples varies by notification. Rather than variation based on information needed by the developer, it is inconsistent with no obvious pattern. If a notification does not provide enough information, the developer may have to interrupt what she is doing to find outside resources to determine the problem [1]. If the notification provides too much detail, she may find it too time consuming to find the important information, thereby ignoring the entire notification or discontinuing use of the tool.

Some tools, like CheckStyle[1] and Firefox Developer Tools, [2] allow developers to manually modify or add information to notifications, which could help deal with notification inconsistencies. However, this can be a manual, time consuming task and many developers may either not be aware of the option or do not want to take the time to create the custom messages [6]. It would benefit developers if tools could automatically determine how much they know about a notification and if tools used that information to adapt their notifications. One way to adapt a notification is to add or remove information, visual or textual, pertaining to the problem.

I propose research that explores the feasibility of tools that automatically tailor themselves to the developer using them. In this paper, I discuss my proposed approach to designing tools to adapt notifications, the novelty of this idea, and the work I have done and plan to do for this research.

II. A New Way to Approach Tool Design

Existing program analysis tools provide notifications to developers without considering what information each individual may need. I propose that tools could be more effective, and therefore more widely used, if they tailored their notifications to the developer using them. This can be done by creating models that represent an individual developer's understanding of programming concepts, such as multi-threading and null object dereferencing. Fritz and colleagues created similar models to determine how familiar a developer is with a codebase [5]. Other works have also created and used similar models, such as JADEITE, a tool for recommending API usage examples [10]. Rather than modeling knowledge of code, however, I would model knowledge of programming concepts.

Existing approaches to designing adaptive tools, such as work done by Zou and colleagues towards adaptive menus in Eclipse, build their models using either real-time data, such as question responses, or data from others, such as code other developers, typically experts, have written [11]. I propose the use of models built based on existing individual developer data and refined based on existing and real-time data collected in a non-intrusive manner. This data could come from a variety of sources. The primary source of existing data would be the code that the developer has written. Presumably, the more code she has written relevant to a particular concept, the more likely it is she understands that concept. Other sources of exisiting data would include bug tracker activity and online community participation.

Using models built based on the individual developer, tools can determine how much that developer may know about the concepts relevant to a given notification. For example, a developer that has written multiple code snippets using

[1]http://checkstyle.sourceforge.net/config.html
[2]https://developer.mozilla.org

978-1-4673-7457-6/15/$31.00 ©2015 IEEE

synchronized blocks may be able to understand a notification about synchronization without any additional information. However, a developer who has never written any code relevant to synchronization may struggle and require additional information or a different description all together. In the following sections, I discuss the work I have done so far towards notification adaptations.

III. Predicting Developer Conceptual Knowledge

I have begun to explore the potential for adapting tool notifications to the developer by collecting developer data to answer the initial question "Can we predict conceptual knowledge?" [7]. I chose to begin with the concept of `null` object dereferencing as this is a fundamental programming concept that many program analysis tools include in their analyses and notifications.

So far, I have collected data from students at NC State University and developers on GitHub, with a total of 17 participants. [3] I asked all participants who gave consent to fill out a concept inventory on `null` and analyzed their code on GitHub. Using the data collected, I created and evaluated models based on code written relevant to `null` object dereferencing; this includes the addition and removal of `null` checks. As a baseline, I created a model based on lines of code written in general; research suggests the code a developer has written is often an indicator of that developer's experience [5], [3].

Our findings suggest that relevant code written may be a better indicator of conceptual knowledge than all code written by a developer. Both models I built currently predict developer knowledge within 1 point of concept inventory score (out of 9) 47% of the time. However, the lines of code added model has a negative correlation with the score and null checks contributed has a positive correlation. Details on these results can be found in a previous paper [7].

IV. Working Towards Personalized Notifications

Despite this glimpse into the possibility of modeling developer conceptual knowledge, there exists challenges that need to be overcome and that shed a light on the need for frameworks to support the developement and use of such models. One challenge is gathering all the developer data needed to build the models, especially being some of the data may not be able to be collected in real-time. Analyzing developer source code provides insights into the developer's experience with the concepts, however, may not be entirely representative of all the developer knows. This could lead to intially inaccurate models. Other challenges include helping developers deal with transitions between notification presentations and scaling the idea up to the large number of programming concepts that can be modeled. As I continue this work, I will investigate how I can also collect data from the developer, such as self-reported experience, to increase the accuracy and usage of the models.

Overcoming the challenges to creating models for notification adaptation is a long term process that will continue to evolve. To further investigate the feasibility of this research, I plan to continue to extend and validate the models I have been building with more developers and build others for other concepts, such as string manipulation and multi-threading. To validate these models, I plan to recruit more developers and predict their knowledge values based on an analysis of their code. I will then ask them to take the concept inventory to determine the actual score and compare the two. Once my predicted scores are more often accurate than inaccurate, perhaps within 1 point of the actual score, I can move on to creating a conceptual knowledge scale. I plan to create a scale for mapping different adaptations to developer knowledge scores, similar to the scale developed by Egelman and Peer for predicting security behavior [4]. I also plan to explore what notification adaptations will look like so tools can more easily determine when to make an adaptation and what kind of changes to make to notification presentation to better support the developer.

[3] http://github.com

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant numbers 1217700 and DGE–0946818.

References

[1] M. B. Edwards and S. D. Gronlund. Task interruption and its effects on memory. *Memory*, 6(6):665–687, 1998.

[2] L. C. Briand, W. M. Thomas, and C. J. Hetmanski. Modeling and managing risk early in software development. In *Proceedings of the International Conference on Software Engineering*, pages 55–65, 1993.

[3] J. J. Cañas, M. T. Bajo, and P. Gonzalvo. Mental models and computer programming. *International Journal of Human-Computer Studies*, 40(5):795–811, 1994.

[4] S. Egelman and E. Peer. Scaling the security wall: Developing a security behavior intentions scale (sebis). In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2873–2882. ACM, 2015.

[5] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 385–394. ACM, 2010.

[6] T. Grossman, G. Fitzmaurice, and R. Attar. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 649–658. ACM, 2009.

[7] B. Johnson, R. Pandita, E. Murphy-Hill, and S. Heckman. Bespoke tools: Adapted to the concepts developers know. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2015.

[8] 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*, pages 672–681, 2013.

[9] N. Nagappan, L. Williams, J. Hudepohl, W. Snipes, and M. Vouk. Preliminary results on using static analysis tools for software inspection. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 429–439. IEEE, 2004.

[10] J. Stylos, A. Faulring, Z. Yang, and B. A. Myers. Improving api documentation using api usage information. In *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*, pages 119–126. IEEE, 2009.

[11] Y. Zou, M. Lerner, A. Leung, S. Morisson, and M. Wringe. Adapting the user interface of integrated development environments (ides) for novice users. *Journal of Object Technology*, 7(7):55–74, 2008.