Get started imminently: Using tutorials to accelerate learning in automated static analysis

Jan-Peter Ostberg and Stefan Wagner

University of Stuttgart, Institute of Software Engineering
Universitätstr. 38, 70569 Stuttgart, Germany
{jan-peter.ostberg},{stefan.wagner}@informatik.uni-stuttgart.de

Abstract

Static analysis can be a valuable quality assurance technique as it can find problems by analysing the source code of a system without executing it. Getting used to a static analysis tool, however, can easily take several hours or even days. In particular, understanding the warnings issued by the tool and rooting out the false positives is time consuming. This lowers the benefits of static analysis and demotivates developers in using it.

Games solve this problem by offering a tutorial. Those tutorials are integrated in the setting of the game and teach the basic mechanics of the game. Often it is possible to repeat or pick topics of interest. We transfer this pattern to static analysis lowering the initial barrier of using it as well as getting an understanding of software quality spread out to more people.

In this paper we propose a research strategy starting with a piloting period in which we will gather information about the questions static analysis users have as well as hone our answers to these questions. These results will be integrated into the prototype. We will evaluate our work then by comparing the fix times of user using the original tool versus our tool.

1998 ACM Subject Classification D.2.5 Testing and Debugging, D.2.9 Management, H.5.2 User Interfaces

Keywords and phrases static analysis, motivation, usability, empirical research, gamification

Digital Object Identifier 10.4230/OASIcs.ICCSW.2012.109

1 Introduction

No one wants to read an encyclopaedia to play a game. –Nolan Bushnell

Many companies are aware of the potential benefits of static analysis, such as increased maintainability of source code[1, 8], but they are afraid to invest the time which a developer needs to get used to a static analysis tool. To a certain degree this fear is justified, because it could take hours to days to understand the basics and many years to fully grasp the warnings and mechanics of the static analysis as well as the implications to software quality. Additionally, it is hard to justify these spendings, if the results are not delivered in a short time and if there is little to no knowledge about software quality and especially maintainability.

Gamers today are in some way similar to these companies. They also do not want to invest much time and effort before they can start experiencing their virtual worlds and have fun. Knowing this, the game developers often have a tutorial stage included into their games, which walks the players through the basic concepts of the game. This tutorial is linked to the game's story and thereby also functions as an introduction to the context, the so called *setting*. Also, to not annoy the experienced or returning gamer, these tutorials are not mandatory to play, but can be re-entered any time later, if the player feels the need to.

So why not bring this idea to static analysis? Besides the reduction of the time to get used to a static analysis tool, the introduction to the context of software quality could lead

2012 Imperial College Computing Student Workshop (ICCSW'12).
Editor: Andrew V. Jones; pp. 109–115

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

more people to a deeper understanding of what software quality is and how it is influenced. This also may enable the users of the tutorial to argument more efficiently with their newly acquired knowledge in favour of the benefits of static analysis. In addition, expert knowledge can be used to customise the tools, which leads to a significantly reduced rate of false-positives as shown by Wagner et al. [13]. In the following, we will lay out a research strategy to cover the current state of the problem, create a suitable prototype and evaluate the benefit of the idea.

2 Related Work

The work by Zheng et al. [14] describes the areas which can be improved by static analysis. In conclusion, the authors state that the use of static analysis could eliminate the less sophisticated faults, freeing up time and capacities to work on the faults which need more thorough analysis by human beings. We can conclude that there is a significant benefit in knowing how to use static analysis and an in-depth knowledge can reduce the amount of time for the analysis, freeing up even more time.

Ayewah and Pugh [2] look into how the static analysis tool FindBugs is used in companies. In detail, they are interested in how they conquer the initial hump due to the high number of warnings at a first time use, as well as the processes used to keep warnings from reappearing. The results show that developers are interested in almost every warning. Also the authors are able to identify some of the processes used by companies to make their work with static analysis more effective.

A view from the creator's side is shown by Bessey et al. [3]. They describe what problems they faced, when they introduced their scientific tool to the "real world". This is a nice example of what is possible today and what it needs to get a tool into everyday practice, for example a well designed installation process or less sophisticated, but understandable analysis.

Pagulayan et al. [10] talk about stumble stones in game development. They point out that if a system is complex, it will profit from a tutorial. They also point out, however, that a tutorial has to follow certain rules, like finding the right pace or not bore the player with tedious instructions.

James Paul Gee [4] has detailed comments about what a good tutorial should provide. He shows this on examples of prominent games. He also points out the possible negative results a bad designed tutorial can have. Both of these sources show that a good tutorial will help get the player more deeply involved, but we have to value some rules, when we create our tutorial, to not generate a negative effect with it. These rules are, for example, a well set pace of difficulty increase as well as not to force the user to strictly go through the whole tutorial.

Randel et al. [11] conducted a literature study on the question of the effectiveness of games for educational purposes. The authors concluded, that depending on a number of variables, e.g. cognitive learning style, games can help learning, because they demand interaction of the player. This interaction increases the chances of the material to be integrated into the cognitive memory and so be remembered more easily. This implicates for our idea that with the tutorial idea, which is close to games, we might have a higher chance of having the users remembering the tool usage.

3 Research Goal

Static analysis tools should be used more commonly in software development, because this would increase the overall quality of the software created. With our research, we aim to find the reasons why they are not widely used and develop strategies to address these problems. In this paper, we focus on a way to shorten the time needed for understanding how to use automatic static analysis and the time needed to understand its analysis results. We think understanding static analysis is a problem, because we observed that in many companies, which use static analysis tools, there is only one person, who takes care of the tool, making the configurations and maintaining the process. Most of the other employees have little to no knowledge of the tool besides starting it. This shows that companies do not want each of their employees to invest the same amount of time into getting familiar with the tool, because they are aware that it is time intensive.

To address this problem, we build on ideas from games which is also known as gamification ([12], [6]). Built-in tutorials, which explain the basic mechanics step by step, seem to us as promising. In the following we well lay down in detail a research strategy to research this hypothesis.

4 Example Static Analysis Tool: FindBugs

There is a huge amount of tools for automatic static analysis available. The abilities of these tools vary from simple style checking to highly sophisticated analyses. Also the form of licensing ranges from free open source to very expensive pay-per-use models. From this motley crew of tools we decided to take FindBugs[1]. FindBugs is a open source tool, distributed under the terms of the Lesser GNU Public License and was developed at the university of Maryland. It uses rather simple rules[2] to find problematic part in Java byte code [1]. This and the fact that it is free of charge makes it one of the most popular analysis tools. By deciding to modify this tool we will have no licensing problems and can benefit from a large community, which we later can offer our modification and so hopefully get feedback for further improvement.

5 Research Strategy

In the following we take a look at the steps we aim to take for our research, which will be described in more detail in the following subsections. The first step will be piloting our idea to gain a feeling for the problems of the users. Step number two will be the creation of a prototype utilizing the information of the pilot. With the prototype done, we can start, as step three, to evaluate the impact of our ideas. To reduce the amount of data to a manageable size, we will first focus on the static analysis tool FindBugs. We decided on this tool, because due to its open source nature it is easily accessible and extendible. We will also provide a code base for the experiments on which the analyses are conducted on.

5.1 Step 1: Piloting the Idea with a Tutor

Before we create a first prototype of the built-in tutorial, we will assess the possible benefit of the tutorial with a tutor in person. The "Wizard of Oz" [5, 7] research method is the

[1] http://findbugs.sourceforge.net/
[2] http://findbugs.sourceforge.net/bugDescriptions.html

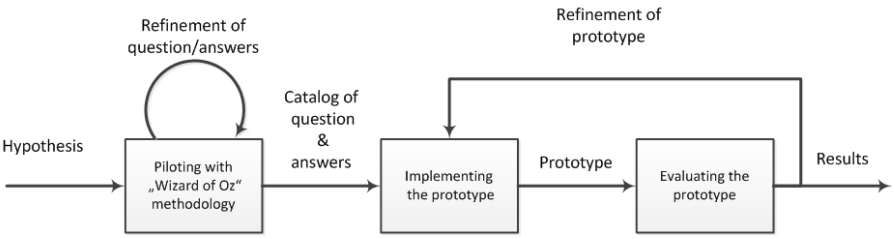

Figure 1 Schematic of the research strategy.

method of choice here. The "Wizard of Oz" is a research method where the participant of the experiment is not aware that he or she is interacting with a human being which is simulating the intelligent tool. We will be able to easily adapt to questions and problems the user has and so have a close feedback loop to refine our question/answer catalogue. We can achieve this in our experiment by using a screen sharing tool like Skype or Teamviewer and a chat. The screen sharing will be hidden to the participant and the chats optic will be modified so that the participants cannot recognise it as such. By conducting this pilot, we will be able to gather information on what the users are interested in to learn from a tutorial. We will be able to adapt our ideas in the piloting phase until they fit the users demand more closely and so deliver a more satisfying experience. The participants of the pilot will be recruited from the students of our university with various study courses to represent the various stages of IT knowledge in the real world. We plan to perform the experiment only with three to 10 students because this setting is time-intensive and, as the sole purpose of the pilot is to assess the feasibility and acceptance of the idea, we expect to gather enough information for the next step with these numbers, as Nielsen et al. [9] shows that 5 testers will find about 85% of the problems. Additionally, we have access to information gathered by a recent study conducted by us, were we observed first time users of FindBugs with an eye tracker and think aloud. These results will also have an influence to our prototype.

5.2 Step 2: The Prototype

With the gathered data from the pilot phase, we will be able to create a first built-in tutorial prototype. For the reward system, we would like to provide, we take some inspiration from ribbon hero[3] created by the Microsoft Office labs. The first challenge to overcome here is finding a story to tell which is related to the topic, not too plain and not too complicated or weird. The setting of a detective story seems to be a nice fit, as we can handle categories of findings as ”cases”. The second challenge will be the design of an engaging experience points system which is fair and comprehensible to the users. The built-in tutorial will offer tasks which wili correspond to the fix of warnings issued by the analysis tool. The tasks issued by the tutorial will increase in difficulty but should never ask too much of the user. Also the user should be able to skip most of the tutorial but has to demonstrate his or her understanding of the task by completing it.

The task should be taken from the source code he or she wants to analyse, but we will have ”back-up” code for the tasks that are not contained in the provided code but still are necessary to be learned for the optimal usage of the tool. This ”back-up” code could also be used as an additional example for tasks that are hard to understand with the code provided

[3] `http://www.ribbonhero.com/`

or if the user requests another example. To get the user more engaged, we will reward the solution of tasks with some kind of point system to make the increase in knowledge visible for the user.

To make the benefit of the "back-up" code more clear, let us consider the following example. We have analysed the source code of JabRef[4], an open source reference manager. One warning issued by Findbugs is:
"... /SampleCode (JabRef)/... /jabref/imports/PdfXmpImporter.java:48
VERY confusing to have methods net.sf.jabref.imports.PdfXmpImporter.getCLIid() and net.sf.jabref.imports.ImportFormat.getCLIId()".
This warning is rather cryptic and hard to understand. Even if you take a look at the code, it might take a long time before you realise what is the problem addressed here. The user might want to have a more easy example of source code which would provoke the same type of warning. An early mock-up of the tutorial information is presented in figure 2. With this

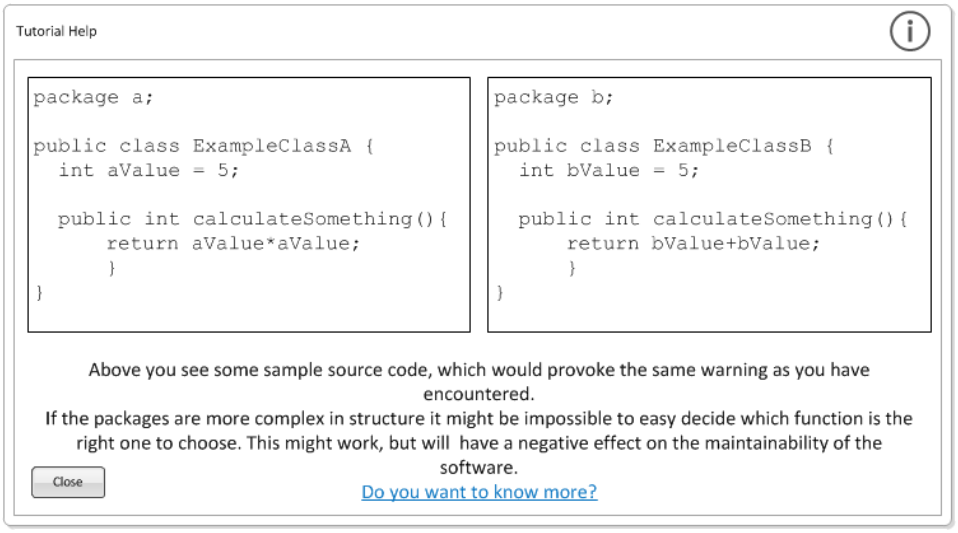

Figure 2 Mockup of prototype tutorial window.

example it is clearer that the warning issued should remind the creator of the code that it is not a good idea to have two methods with the same name in the same project doing different things. As we still follow the tutorial idea, we would also offer some information text, which will provide a link to even more detailed information comparable to an encyclopaedia, for those user, who really want to master the tool and want to dive deep into the ideas of software quality. This should make the proposed changes of the tool and the tutorial more convincing to the users as well as help them to understand how to create software of higher quality and avoid future mistakes.

5.3 Step 3: Evaluation of the Prototype

After the prototype is finished, we are planning an evaluation phase. To evaluate the benefit of our approach we will compare the time needed to fix a given set of warnings in a code base of participants using our enhanced tool versus participants using the original tool. We will

[4] `http://jabref.sourceforge.net/`

take the time from start to finish of the whole operation as well as of the fix time only. To do that, we will measure how long the participants work with the tutorial and how much time the other participants invest into getting to know the tool. We expect that especially the more complex problems will be solved faster by user with our tutorial approach. Additionally we ask the participants afterwards to use the tool, they did not use yet. Here the time is, of course, not measured, because the participants already learned from the other tool. We will issue questionnaires then to cover the subjective helpfulness of the tutorial for getting started with FindBugs and static analysis, as well as the level of engagement created through the gamification of the tool versus the unmodified tool. For example, planed question are:

- Do you think the enhanced tool is faster understandable then the original one?
- Do you feel more motivated to fix issues by the game mechanics?
- Have you used the possibility to gain more information considering software quality?
- Did the tool raise your interest in software quality?
- Would you prefer to use the enhanced or the original tool?
- ...

To make the answers more comparable, we will offer four possible answers for the question that do not need a more complex answer.

- "Yes, I fully agree."
- "Yes, I agree mostly."
- "No, I do not agree to that."
- "No, I completely disagree with that."

Moreover, we will have a section where the participant can give free feedback on the prototype. We will carefully keep track of this feedback and use it from time to time to make useful improvements to the software. For a long term evaluation, we consider to include the prototype as a lecture accompanying instrument in teaching. Here we plan to use our tool for a whole semester and ask the students at the end of the semester to state their experience with the tool. The details of this are subjects of future work.

6 Summary and Future Work

We presented our overall research goal, which is making automatic static analysis a more common tool in software development. In this paper we propose a tutorial attempt to shorten the time needed to get started with static analysis. As a side effect the proposed idea will teach the willing user to learn the ideas behind the issued warnings. These ideas reach into software quality and software engineering topics. We laid out a research strategy to create a problem oriented catalogue of questions and answers for our tutorial by a pilot study and to evaluate the benefits of our approach.

There are other aspects of the automated static analysis, that might make it unattractive and is not covered here. For example the problems could originate from a poor operability. We are planning to examine this aspect with an eye tracking study which we will conduct shortly. Finally, there is still the problem with the false positives. Future work will also aim to find techniques to reduce or make them easier to spot and track.

References

1 N. Ayewah, D. Hovemeyer, J.D. Morgenthaler, J. Penix, and W. Pugh. Using static analysis to find bugs. *IEEE Software*, 25(5):22–29, 2008.

2 Nathaniel Ayewah and William Pugh. A report on a survey and study of static analysis users. In *Proceedings of the 2008 workshop on Defects in large software systems*, DEFECTS '08, pages 1–5. ACM, 2008.

3 Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM*, 53:66–75, 2010.

4 James Paul Gee. What video games have to teach us about learning and literacy. *Comput. Entertain.*, 1(1):20–20, October 2003.

5 J. F. Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst.*, 2(1):26–41, January 1984.

6 Jane McGonigal. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. The Penguin Group, 2011.

7 Lennart Molin. Wizard-of-oz prototyping for co-operative interaction design of graphical user interfaces. In *Proceedings of the third Nordic conference on Human-computer interaction*, NordiCHI '04, pages 425–428, New York, NY, USA, 2004. ACM.

8 Nachiappan Nagappan and Thomas Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 580–586. ACM, 2005.

9 Jakob Nielsen and Thomas K. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, CHI '93, pages 206–213, New York, NY, USA, 1993. ACM.

10 Randy J. Pagulayan, Kevin Keeker, Dennis Wixon, Ramon L. Romero, and Thomas Fuller. The human-computer interaction handbook. chapter User-centered design in games, pages 883–906. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003.

11 Randel. The Effectiveness of Games for Educational Purposes: A Review of Recent Research. *Simulation Gaming*, 23:261–276, 1992.

12 Byron Reeves and J Leighton Read. *Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete*. Harvard Business School Press, 2009.

13 S. Wagner, F. Deissenboeck, M. Aichner, J. Wimmer, and M. Schwalb. An evaluation of two bug pattern tools for Java. In *Proc. 1st International Conference on Software Testing, Verification, and Validation (ICST'08)*, pages 248–257. IEEE Computer Society, 2008.

14 J. Zheng, L. Williams, N. Nagappan, W. Snipes, J.P. Hudepohl, and M.A. Vouk. On the value of static analysis for fault detection in software. *IEEE Transactions on Software Engineering*, 32(4):240–253, 2006.