Journal of Information Technology Review

DLINE JOURNALS

Print ISSN: 0976-3511
Online ISSN: 0976-2922
JITR 2025: 16 (3)
https://doi.org/10.6025/jitr/2025/16/3/85-93

Gamified Tutorials for Accelerating Adoption of Static Analysis Tools in Software Development

Jan-Peter Ostberg and Stefan Wagner
University of Stuttgart, Institute of Software Engineering
Universitätstr. 38, 70569 Stuttgart, Germany

ABSTRACT

We investigate how tutorials inspired by video game design can improve the adoption and effective use of static analysis tools in software development. Static analysis offers significant benefits for code quality and maintainability; however, its adoption is hindered by steep learning curves and time-intensive on boarding processes. Drawing parallels to gamers who prefer quick and engaging tutorials, the authors propose embedding interactive tutorials in static analysis tools to lower entry barriers.

Their research strategy includes three phases: (1) a pilot study using the Wizard-of-Oz method to simulate a tutorial via human guidance and identify user needs, (2) the development of a prototype tutorial within the open-source tool FindBugs using a gamified approach (e.g., storytelling and rewards), and (3) an evaluation comparing user performance and engagement with and without the enhanced tool. The goal is to reduce training time, improve comprehension of warnings, and raise awareness of software quality.

The study anticipates that gamified tutorials can not only accelerate tool learning but also educate users on deeper software engineering principles. Future work includes long-term classroom testing and addressing usability issues such as false positives and poor interface design.

Keywords: Static Analysis, Game Design, Software Quality, Embedding Interactive Tutorials

Received: 18 November 2024, Revised 29 January 2025, Accepted 23 February 2025

1. Introduction

Numerous organizations recognize the potential advantages of static analysis, such as enhanced maintainability of source code [1, 8], yet they hesitate to allocate the necessary time for developers to familiarize themselves with a static analysis tool. To some extent, this hesitation is warranted, as it can take anywhere from hours to days to grasp the fundamentals, and many years to fully understand the warnings and intricacies of static analysis, along with its impact on software quality. Furthermore, justifying these expenditures is chal lenging if results are not produced promptly, and there is a limited understanding of software quality, particularly in terms of maintainability.

Today's gamers share similarities with these companies in some ways. They are also reluctant to invest significant time and effort before they can immerse themselves in their virtual experiences and enjoy themselves. Aware of this, game developers often incorporate a tutorial phase into their games, guiding players through the fundamental principles of the game. This tutorial is tied to the game's narrative, thus serving as an introduction to the context, or the so-called setting. Moreover, to avoid frustrating seasoned or returning players, these tutorials are optional; they can be revisited at any later time should the player wish to do so. So why not apply this concept to static analysis?

Beyond minimizing the time required to adapt to a static analysis tool, introducing the context of software quality could foster a more profound understanding among more individuals regarding what software quality is and how it is affected. This might also empower tutorial users to more effectively advocate for the advantages of static analysis with their newfound knowledge. Additionally, expert insights can be utilized to tailor the tools, resulting in a significantly lower rate of false positives, as demonstrated by Wagner et al. [13].

In the following sections, we will outline a research plan to assess the current state of the issue, develop an appropriate prototype, and evaluate the advantages of this concept.

2. Related Work

The research conducted by Zheng et al. [14] highlights the aspects that can be enhanced through static analysis. Ultimately, the authors conclude that employing static analysis could eliminate less complex errors, allowing more time and resources to be devoted to addressing faults that require thorough human examination. We can deduce that possessing knowledge of how to utilize static analysis yields significant benefits, and a deeper understanding can further reduce analysis time, thus freeing up even more time.

Ayewah and Pugh [2] examine the utilization of the static analysis tool FindBugs in various organizations. Specifically, they focus on how companies overcome the initial challenges posed by the numerous warnings encountered during first-time usage, as well as the strategies they employ to prevent warnings from recurring. Their findings indicate that developers show interest in nearly every warning. Moreover, the authors identify several practices that companies use to enhance their effectiveness when working with static analysis. From the creators' perspective, Bessey et al. [3] discuss the challenges they encountered when introducing their scientific tool into the "real world." This serves as a compelling illustration of what is achievable today, along with the requirements for integrating a tool into daily operations, such as a well-structured installation process or more straightforward yet comprehensible analyses.

Pagulayan et al. [10] address obstacles in game development, emphasizing that a complex system benefits from having a tutorial. They also note that a tutorial must adhere to specific guidelines, such as maintaining an appropriate pace and avoiding tedious instructions that could bore the player.

James Paul Gee [4] provides in-depth insights regarding the essential components of a practical tutorial. He illustrates his points using examples from well-known games and highlights the potential adverse effects that a poorly designed tutorial can cause. Both sources suggest that a well-crafted tutorial can enhance player engagement, but it is crucial to adhere to certain principles to avoid adverse outcomes. These principles include establishing a suitable progression of difficulty and allowing users the flexibility to navigate the tutorial without strict constraints.

Randel et al. [11] performed a literature review to investigate the effectiveness of games for educational purposes. The authors concluded that, depending on various factors, such as cognitive learning styles, games can facilitate learning by requiring player interaction. This interaction increases the likelihood that the material will be assimilated into cognitive memory, making it easier to recall. This suggests that our tutorial concept, which closely resembles games, might enhance users' retention of tool usage.

3. Research Purpose

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 Find Bugs[1]. Find Bugs 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

[1] http://findbugs.sourceforge.net/

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

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

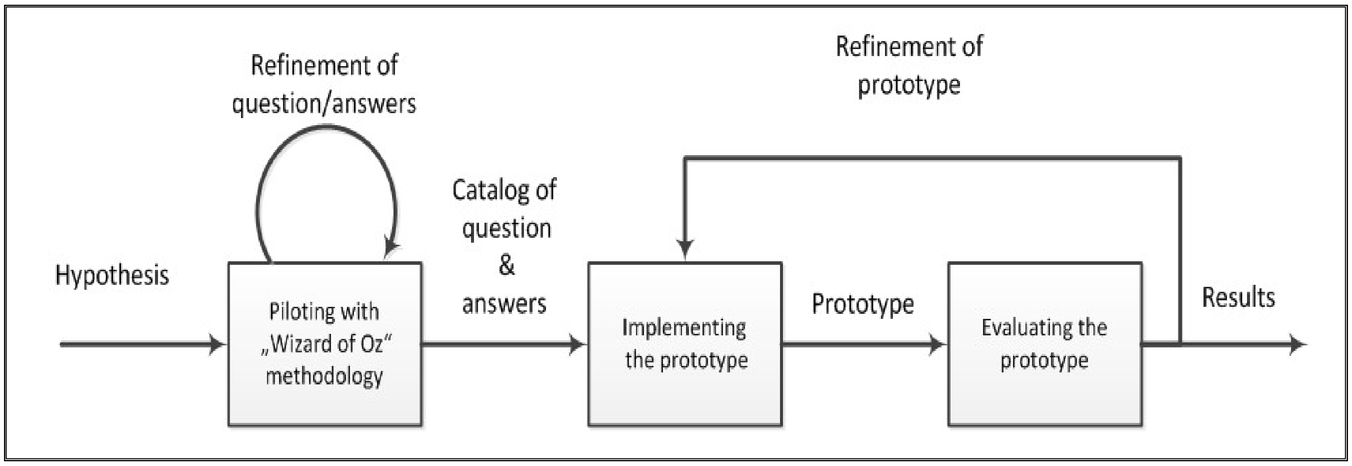

Refinement of question/answers
Refinement of prototype
Catalog of question & answers
Hypothesis
Piloting with „Wizard of Oz“ methodology
Implementing the prototype
Prototype
Evaluating the prototype
Results

Figure 1. Schematic of the research strategy

method of choice here. The “Wizard of O_z” 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.

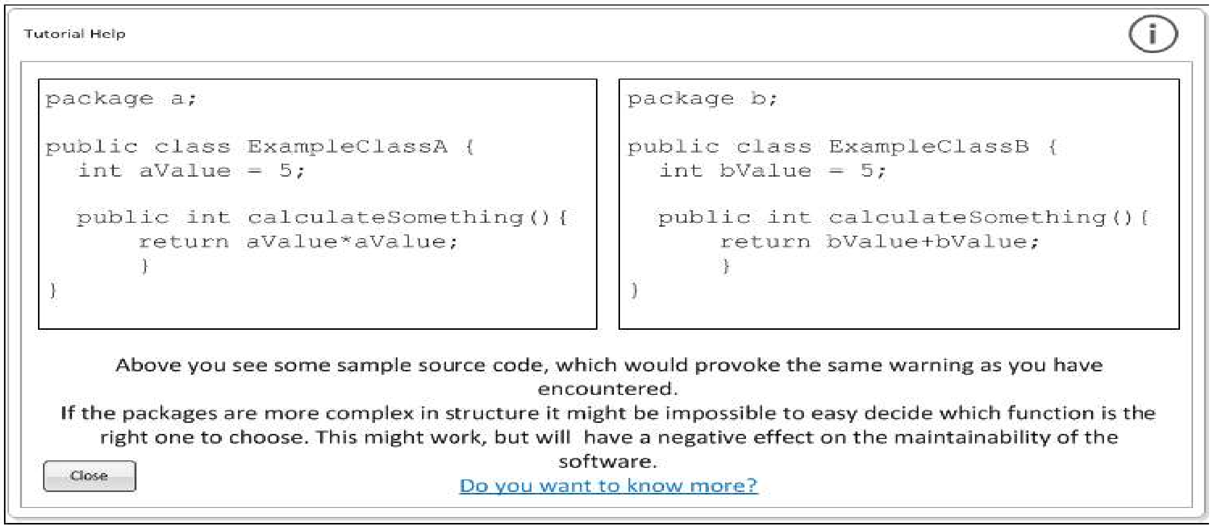
For the reward system, we would like to provide, we take some inspiration from ribbon hero[3] created by the Microsoft Office labs. The first challenge to overcome here is finding a story to tell which is related to the topic, not too plain and not too complicated or weird. The setting of a detective 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 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 JabRef4, 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



```
package a;

public class ExampleClassA {
  int aValue = 5;

  public int calculateSomething(){
      return aValue*aValue;
      }
}
```

```
package b;

public class ExampleClassB {
  int bValue = 5;

  public int calculateSomething(){
      return bValue+bValue;
      }
}
```

Above you see some sample source code, which would provoke the same warning as you have encountered.
If the packages are more complex in structure it might be impossible to easy decide which function is the right one to choose. This might work, but will have a negative effect on the maintainability of the software.
Do you want to know more?

Figure 2. Mockup of prototype tutorial window

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

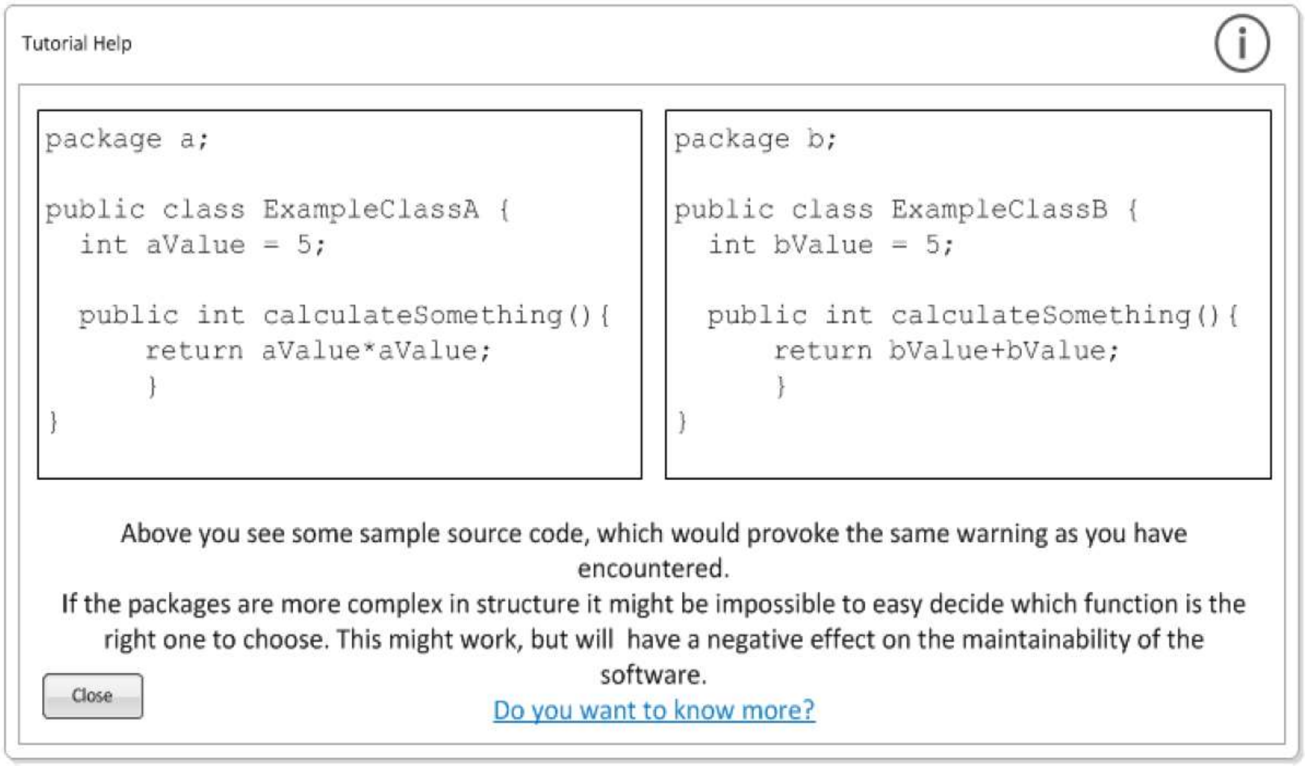
or if the user requests another example. To get the user more engaged, we will reward the solution of tasks with some kind of point system to make the increase in knowledge visible for the user.

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



```
package a;

public class ExampleClassA {
  int aValue = 5;

  public int calculateSomething(){
      return aValue*aValue;
      }
}
```

```
package b;

public class ExampleClassB {
  int bValue = 5;

  public int calculateSomething(){
      return bValue+bValue;
      }
}
```

Above you see some sample source code, which would provoke the same warning as you have encountered.
If the packages are more complex in structure it might be impossible to easy decide which function is the right one to choose. This might work, but will have a negative effect on the maintainability of the software.
Do you want to know more?

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

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

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

[2] Ayewah, Nathaniel., Pugh, William. (2008). 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* (pp. 1–5). ACM.

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

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

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

[6] McGonigal, Jane. (2011). *Reality is broken: Why games make us better and how they can change the world*. The Penguin Group.

[7] Molin, Lennart. (2004). 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* (pp. 425–428). ACM.

[8] Nagappan, Nachiappan., Ball, Thomas. (2005). Static analysis tools as early indicators of prerelease defect density. In *Proceedings of the 27th international conference on Software engineering, ICSE '05* (pp. 580–586). ACM.

[9] Nielsen, Jakob., Landauer, Thomas K. (1993). 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* (pp. 206–213). ACM.

[10] Pagulayan, Randy J., Keeker, Kevin, Wixon, Dennis, Romero, Ramon L., Fuller, Thomas. (2003). User-centered design in games. In *The human-computer interaction handbook* (pp. 883–906). L. Erlbaum Associates Inc.

[11] Randel. (1992). The effectiveness of games for educational purposes: A review of recent research. *Simulation Gaming, 23*, 261–276.

[12] Reeves, Byron, & Read, J. Leighton. (2009). *Total engagement: Using games and virtual worlds to change the way people work and businesses compete*. Harvard Business School Press.

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

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