



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Visualizing the UI impact of Methods for Program Comprehension

Master's thesis in Software Engineering Programme

ANTON GREGORY

MASTER'S THESIS 2018

Visualizing the UI impact of Methods for Program Comprehension

ANTON GREGORY



Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Visualizing the UI impact of Methods for Program Comprehension
ANTON GREGORY

© ANTON GREGORY, 2018.

Supervisor: Regina Hebig, Software Engineering
Examiner: Jan Philippe Stegnoefer , Software Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2018

Visualizing the UI impact of Methods
ANTON GREGORY
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Software developers spend 58% of their time comprehending source code. Most methods to support this task are limited to providing abstract views on a system's building blocks and connections. However, this perspective is fundamentally different to the user view on a running software system. Indeed, it was found in a study that developers working with UI systems use the UI as a starting point to comprehend the software. However, there is a lack of techniques that help developers to map user view to the source code. We developed a first simple prototype to automatically generate documentation that includes links to the user view and performed a qualitative user study with 24 participants. The results indicate the feasibility of the development of such prototype and provide insights into perceived advantages and limitations. This indicate that this research direction has the potential to change the way we understand software in future.

Keywords: software, engineering, computer, science, thesis, comprehension, UI, visualization, documentation, maintenance, reverse engineering, user interface, document generation.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor Regina Hebig for guiding me and supporting me through out the research. Your constant feedback during my writing and continuous assistance helped me a lot.

I would also like extend my gratitude to my examiner Jan-Phillipp, all the anonymous people who responded my questionnaire and my opponent Logesh for being patient with me to complete my thesis.

Anton Gregory, Gothenburg,Sweden

Contents

List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Problem Description	1
1.2 Purpose of Study	2
1.3 Disposition	3
2 Background	4
2.1 Related Work	4
2.1.1 Documentation	5
2.1.2 Architecture	5
2.1.3 Code	5
2.2 GUI-based Software Testing	6
2.2.1 1 st generation: Coordinate-based	6
2.2.2 2 nd generation: Component/Widget-based	6
2.2.3 3 rd generation: Visual GUI Testing	6
3 Methods	8
3.1 Research Methodology	8
3.1.1 Problem identification and motivation	8
3.1.2 Define the objectives for a solution	10
3.1.3 Design and development:	11
3.1.4 Demonstration	12
3.1.5 Evaluation	12
4 Implementation	20
4.1 Prototype Development	20
4.2 Configuration	25
4.3 Process of Tracing a Method	28
5 Results	31
5.1 UI of Arts Of Illusion	31
5.2 Overview of Results	33
5.2.1 Build failure	34
5.2.2 Successful build and failed to open jar	34

5.2.3	No difference found	34
5.2.4	Differences found	35
5.3	Quality of images recorded	36
5.4	Mapping of Recordings	38
5.5	Impact Observed on UI Component	40
6	Evaluation	47
6.1	Analysis of Data	47
6.1.1	Familiarizing the data	47
6.1.2	Generating initial codes	48
6.1.3	Searching for themes	49
6.1.4	Reviewing themes	49
6.1.5	Defining and naming themes	51
6.1.5.1	Advantages	51
6.1.5.2	Limitations	53
6.1.5.3	Usages	54
6.2	Findings from Analysis	56
7	Discussion	61
7.1	Threats to validity	61
7.1.1	Internal Validity	61
7.1.2	External Validity	61
7.2	Limitations	62
7.3	Discussion	63
8	Conclusion	68
	Bibliography	70

List of Figures

3.1	Design science research activities [36]	9
4.1	Architecture diagram of prototype	20
4.2	Chart to illustrate the results obtained from the tool	24
4.3	Project setup GUI of UI-Tracer tool	25
4.4	Project setup GUI-2 of UI-Tracer tool	26
4.5	Screenshot of the ‘Arts of illusion’ application	27
4.6	Image to illustrate how the screenshot in image 4.5 is divided	27
4.7	First quarter of the image from ‘Arts of illusion’ application with un-manipulated code	28
4.8	Screenshot of the ‘Arts of illusion’ application with manipulated code	29
4.9	Image saved as result by UI-Tracer	29
4.10	Sequence diagram of UI Tracer tool	30
5.1	Screenshot of ‘Arts of Illusion’ application	31
5.2	Menu Bar of ‘Arts of Illusion’	32
5.3	Tool Bar of ‘Arts of Illusion’	32
5.4	View window of ‘Arts of Illusion’	32
5.5	Object panel & Properties panel of ‘Arts of Illusion’	33
5.6	Score panel of ‘Arts of Illusion’	33
5.7	Chart to illustrate the results obtained from the tool	35
5.8	Pie chart of images recorded as changes	36
5.9	Example of an invalid recording that has the screenshot of background application	37
5.10	Example of image recorded as difference with blank UI	37
5.11	Difference in component length <i>Front</i>	38
5.12	Camera object transformed	41
5.13	Camera object missing from view window	42
5.14	Result of ‘File’ menu button missing in Menu bar	42
5.15	Comparison between original Tool Bar panel and result from tool	43
5.16	Comparison between original Object panel and result from tool	43
5.17	Comparison between original Properties panel and result from tool	44
5.18	Difference in component View window <i>Front</i>	44
5.19	Comparison between original Control elements from view control and result from tool	45
5.20	Comparison between original score component and result from tool	45
5.21	Difference in component View window <i>Front</i>	46

6.1	Mind map of the central theme ‘Advantages’	50
6.2	Mind map of the central theme ‘Limitations’	51
6.3	Mind map of the central theme ‘Usages’	51
6.4	Chart of Themes obtained for question 1 and their count	57
6.5	Chart of Themes obtained for question 2 and their count	58
6.6	Chart of Themes obtained for question 3 and their count	59
6.7	HeatMap of answers obtained for question 4	60

List of Tables

3.1	Question given to participants	17
4.1	Values returned based on method return type.	21
5.1	Types of results produced based on methods	35
5.2	Number of images grouped by their category	38
5.3	Mapping of Methods and number of valid recordings	40
5.4	Mapping of UI components and number of changes	41
6.1	Sample answer obtained for question 4 from questionnaire	48
6.2	Advantages of UI Tracer and the number of participants who mentioned it	56
6.3	Limitations of UI Tracer	57
6.4	Usage of UI Tracer	58

1

Introduction

Maintenance of software is as important as its development because 65 % to 75% of the overall effort in software lifecycle goes to maintenance [1]. The code undergoes changes during maintenance of the software, e.g. for fixing bugs, to cope up with hardware upgrades, or to incorporate changing requirements [4]. Due to these reasons, the maintenance phase of the software lifecycle is often regarded as one of the most important and expensive phases above all when compared to other phases of the software development lifecycle [2]. The size of the code and complexity of the code increases as the source code evolves. This has a significant impact on the understandability of the software [25].

Software developers and maintainers need to read and understand the source code and other documentation at their work. Programmers use software comprehension techniques to understand code and it aids them more during the maintenance phase of the software life cycle. These techniques help the programmer to understand the system components, architecture, source code etc. For example, there are software such as Graphviz that help the user to visualize the software components and their dependencies. Müller [18] describes software comprehension as, “a process where a software practitioner understands a software artifact using both knowledge of the domain and/or semantic and syntactic knowledge, to build a mental model of its relation to the situation”. Program comprehension also acts as a core factor in analysis, refactoring and reengineering of the program [19]. Due to the significance of program comprehension on software maintenance, there is always a need to investigate and understand the issues involved in software comprehension. This necessity is the reason for this research and contribution to program comprehension.

1.1 Problem Description

The developers who developed the software might not be available when software enters the maintenance phase. Now the new developers who maintain the software struggle to understand the code. They have to understand the source code by themselves or use documentation. Necessary documentation might not exist and if it does exist it might be outdated. Research shows that developers rely more on source code when compared to documentation [5]. In some cases, there will be too much of information provided in the documentation whereas in other cases too little information [5]. These cases stall the software maintainers to get the informa-

tion they require. Easy and quick understanding of the source code could have an enormous potential to save the time required for comprehension during bug fixing and maintenance tasks. It has been found that 58% of developers time is spent on comprehending the software system under maintenance [55].

In fast-paced working environments finishing work on time is vital. During maintenance, the software developers spend more time trying to understand the software. This delays the maintenance process even further. High pressure on the maintainers might make them work without having adequate knowledge of the software system. This is an immense problem for organizations because the chances of new bugs to get introduced are high [9]. Moreover, one change might cause a ripple effect in the source code, leading to the introduction of additional bugs. The results of a study concerning code inspection showed that 60% of the issues reported by professional reviewers were maintenance issues related to software understandability [4]. These above mentioned factors and results indicate the significance of understandability in software.

Boehm [28] observed that modification of software is usually performed in three steps: understanding of the existing software, modifying the existing software and re validating the modified software. A study [23] conducted on program comprehension techniques found that maintainers interact with the UI to comprehend the software. According to this study 17 out of 21 participants who work with GUI applications used UI to comprehend the software. Certain participants in the interview mentioned that they used UI to inspect code. For example, to know the part of the code that was executed by a button click. This information is then used as a starting point for the exploration of code. The information obtained will help in the process of comprehending the software for developers who maintain systems with UI, and thus this proves the significance of UI on comprehension for developers working on systems with UI. Their study concludes that maintainers test by interacting with the user interface to know the starting point of the application for further inspection. From the same study it is known that there are no techniques available to help the user with the mapping of code to UI. In addition to the above finding, it has been found in the study [23] that developers follow a “problem-solution-test” pattern when they work. This pattern implies that developers begin by identifying the location of a problem, implement the solution to the problem and test the solution. This indicates that knowing the location of logic in the system could help developers in comprehending software systems.

1.2 Purpose of Study

In this research, a new comprehension technique will be proposed which would help the user understand the impact a method has on the UI. This technique will be different from the conventional text-based and model-based comprehension techniques. The aim of using this UI-based technique is to provide the user with an easy visual description of the impact that the source code has on the system, and thus making it easier for user to understand the location of methods that are responsible for UI

rendering. This will in turn create a mapping for users to know the significance of the methods on the user interface. We believe that this will help the user understand the impact of a piece of code in the software and thus helping the user to identify the mappings of the code to functionality.

Through this thesis, we wanted to understand the feasibility of creating a tool that could generate a mapping of UI components to its source code. We also wanted to understand the results of the tool and investigate if the developers could use them. We believe this technique could be used in conjunction with other standard comprehension techniques that already exist right now. This way we would like to create a comprehension technique and contribute to the research in the software comprehension domain.

1.3 Disposition

This document provides the reader with a comprehensive description of the thesis research. In the remainder of this document, we will present the background theory related to the subject of the thesis in chapter 2. Following that in chapter 3 we have presented the methodology we used to conduct the research. Next, we have presented the implementation of the prototype developed as a part of the thesis in chapter 4. The results we obtained from the prototype are discussed and analyzed in chapter 5. The findings from the evaluation of the results are presented in chapter 6. Finally, we discussed our contribution in chapter 7 and concluded with some ideas for future research in chapter 8.

2

Background

In the following sections, the background of this research is discussed in detail. This chapter will illustrate the already existing comprehension techniques providing a clear outline of this research.

2.1 Related Work

Understandability of the software code is heavily affected by multiple factors followed during software development. These include violating the best practices while documenting the code, introduction of complex logic in the code, bad readability etc [5]. Software maintainers work with unfamiliar code that may not be written by them [3]. When the maintainers understand the code faster it becomes easy for them to start working on it.

When the developers understand the code, they can identify the potentially reusable components which might be of use to them during maintenance. In the U.S. department of defence alone \$300 million could be saved by increasing the re-usability of the code just by 1% [6]. Although reuse saves cost and time, the difficulty in understanding the source code will stall the developers to re-use the components created by other developers. One study report states that 40% to 60% of the maintenance activity is invested in understanding how the software and how to implement the planned changes in the maintenance [4]. This indicates how important it is for organizations to focus on the creation and updation of software documentation .

Considering the understandability issues maintainers face, program comprehension has always been studied intensively [19]. Many techniques are followed to generate documentation of software. For example in java, annotation based documentation is supported. Annotations provide information about the program but it is not a part of the program itself [13]. These annotations are extracted using JavaDocs[14] to provide the documentation. Tools such as Doxygen [15]/Graphviz [16] provide visual representation of the structure of the code which could be of use to developers to understand the system visually. Software visualization acts as a documentation helping the user with a visual representation of the complex views of the system [17]. There are various techniques followed widely in program comprehension on different levels of the software. Each of them are discussed below.

2.1.1 Documentation

Documentation is a well known program comprehension technique. Documentation is a textual description that is intended to communicate the information about the software system. As this acts as a communication model it is rather important that a good and up to date documentation is available to the maintainers of the system. Documentation might contain requirements, design architectures or code explanations etc. There are currently various tools available that help in generating document from the source code. These tools extract the information from the source code and convert the source code into documentation. Doxygen [15], Sphinx [39] etc are some of the document-generating software currently available. Even though software documentation contains information about the software system, it is important to note whether the information in the documentation is up to date or not. As the software system evolves the documentation might get outdated. Rise of agile practices in organizations also puts the engineers in a situation where they have less documentation [20]. These factors affect the documentation to a large extent.

2.1.2 Architecture

Software visualization is a technique which aids in the understanding of the software by presenting the components of the software as visual images[21]. This technique gives physical shapes to shapeless software code and presents them to the user. The user could gain an insight about the structure of the software code or comprehend any complex component etc. Jarchitect, SourceTrail, JIVE [40, 41, 42] are some of the popular known tools that offer software visualization. Reverse engineering in software is a research area which aids in program comprehension. This technique emphasizes the understanding of the system, its components and their inter relationship from the end product. With reverse engineering the structure of the components and their dependencies could be understood, and thus aiding in program comprehension. Reverse engineering is, however, a tedious task for large amount of data [38].

2.1.3 Code

On code level there are techniques that support program comprehension. Following proper coding standards throughout the entire code base, addition of comments at required places and good design etc impact the readability of the code. Program slicing is another technique which simplifies programs by abstracting the intended method from a complex program. This is done by removing the parts of the program which have no effect on the method of interest. This helps in understanding a specific part of the code by eliminating complexity in the method [22]. Research [23] shows that most developers try to comprehend the software directly by debugging it. It should be noted that this approach is time consuming when dealing with systems with a huge code base.

2.2 GUI-based Software Testing

GUI testing of software is a way to ensure the functionality of the user interface meet its specifications. There are various tools developed as an application of GUI testing. These tools help the user to write scripts that automate and perform testing. These tools could be made to perform actions on GUI and emulate user actions as well. But these tools though sharing the same purpose they have differences in the way they interact with the GUI. According to these differences they are classified in to the following generations [26] .

- 1st generation: Coordinate-based.
- 2nd generation: Component/Widget-based.
- 3rd generation: Visual GUI Testing.

2.2.1 1st generation: Coordinate-based

The first generation automation is based on getting the coordinates from the screen and then using it to interact with the GUI. To get the coordinates the user actions is recorded which is then used to generate scripts that could be played as an automation process. The risk involved in this approach is that any minor change in UI component will flunk the automation because the action is triggered based on coordinates. Examples of this generation include JUnit [37], Sikuli [12] etc

2.2.2 2nd generation: Component/Widget-based

In this generation of automation, the automation is simulated by direct access to GUI components. This technique is also called as Widget or Tag based GUI testing. This technique is followed widely in industries due to its robustness. In this approach the tags in GUI is used for automation. For example there is a button with label ‘OK’ then the tool will scan to locate the tag ‘OK’ and then simulates user action. Examples of this generation tools are Selenium [45], QTP [45] etc. The main limitation of tools belonging to this generation is that testing over certain GUI applications is not possible as the tags are inaccessible. The other limitation is that there is no mean to verify the correctness from a GUI point of view, both in terms of appearance or behavior [43]

2.2.3 3rd generation: Visual GUI Testing

The 3rd generation GUI-based testing is also referred to as Visual GUI Testing. This generation tools use image recognition capabilities to interact and assert the UI components in GUI application. Examples of this generation includes, Sikuli [12] an open source application, JAutomate etc. Each form of GUI testing in this generation has different advantages and disadvantages due to their individual features. E.g. Sikuli is free as it is an open-source while JAutomate is not [44].

Image recognition on these tools is usually done in two steps. First the current state of GUI is captured as a screenshot. Then the image recognition algorithm scans for the image(eg : an image of a button) in the screenshot . If it recognizes the image then the coordinates of the matched object is returned. These coordinates will be then used to perform actions on them. Different tools uses different algorithms but most algorithms rely on similarity based matching. This means that there is a percentile up to which a match will be considered acceptable. For example, if the acceptability reaches the 70th percentile it is considered a match. This degree could, however, be adjusted in most of the tools that fall under this generation category [43].

3

Methods

In this chapter, the methodology of this research is explained. Here along with the methodology we also discuss how we derived at research questions and how this research is carried out to address them.

3.1 Research Methodology

This research is carried out with a design science research methodology [36]. This methodology helps in addressing unsolved and important problems in new and innovative ways. Therefore, we chose design science research methodology for this thesis research which enables us to develop and study new approaches that addresses the problems in program comprehension. This design science research method consists of the following activities: identifying the problem, defining the objectives for a solution, design and development, demonstration and evaluation. The six activities that constitute a design science in research is shown in the figure 3.1.

3.1.1 Problem identification and motivation

This activity is performed to identify the problem area and to derive at a solution that is concrete. This phase therefore determines the value of the solution itself, and thus contributes to the development of the artifact. Moreover, this phase also provided a motivation for carrying out the research and to know whether the results obtained are valid to be considered as a solution.

The paper “How Do Professional Developers Comprehend Software?” [23] is a study conducted to explore how program comprehension tools are used in practice and comparing the techniques effectively with one another. As the base of our thesis research is about program comprehension, the paper [23] answered questions about the anomalies we had in the beginning. In this paper [23] it has been found that developers or maintainers who work with GUI applications comprehend by identifying the methods that are executed as a consequence of button click or other use actions. In addition to this the paper also proves that users interact with the UI to understand the system. The paper “Which documentation for software maintenance?” [5] is a study conducted to know how much documentation is of help to the maintainers and that too what artifacts are considered important to them.

When the research was started, we were unsure how developers comprehend a GUI

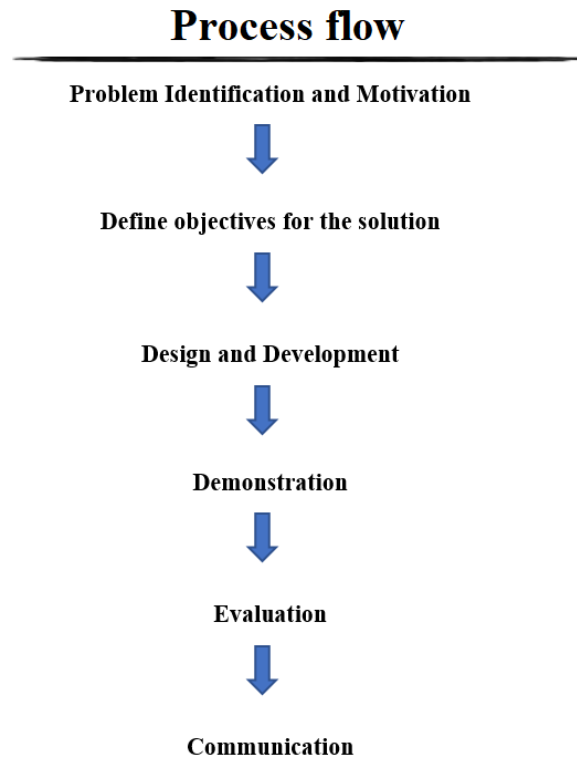


Figure 3.1: Design science research activities [36]

systems. Moreover, we were unsure whether the information about the mapping of code and GUI elements could be of any use to them. On reading papers about program comprehension techniques, the paper [23] helped us in understanding how professional developers comprehend software in the industry. The study shows the approach developers follow when they comprehend. They found that 18 participants out of 28 followed a “problem-solution-test” [23] work pattern, which is in conjunction with what Boehm [28] observed during the modification of software. This is a three-step process: Identifying the problem, applying the solution and testing the applied solution. In case of implementation of a new feature, the first step of identifying the problem is replaced with the identification of code locations.

Moreover they also found that, in systems that involve UI, the developers tend to examine the UI and try to understand the code from there. In order to locate the part of the code that impacts the UI, the developer must comprehend the code using conventional methods. These methods can include text-based documentation or model-based techniques. There are, however, no tools that provide the necessary documentation for this purpose. This means that the developer has no tools to support them in mapping the UI to the code. They debug to trace information about how UI and code are related. Based on these findings we concluded that automation support deriving this tracing could assist developers.

Research questions

The research questions were formulated such that they can address the aim and intended contribution of this project. Considering the problems analyzed and discussed during the previous activity we derived at the following research questions.

RQ1. How can we design and implement the technique to visually observe the impact a method has on the UI?

RQ2. Can we map the UI components of the view that appears after the application is loaded (the “starting screen”) to its source code through this approach?

RQ3. How do developers or maintainers perceive the documentation created using the technique under study?

3.1.2 Define the objectives for a solution

After the problem has been identified the next step was to frame the objectives of the solution according to the chosen methodology. At first, we wanted to comment each method, record the differences in the UI and then produce the result as documentation. However, to comment each method one by one in source code is a tedious task. We decided to approach the problem by developing a tool that could automate how developers debug the system with UI to locate parts of the code that impacts UI. We believed that such tool if developed could help the developers by providing them with useful information such as the parts of the code impacting the UI and thus enabling them to comprehend the software faster.

The intention of developing the proposed tool is to automate the way a developer would debug the application to locate the parts of the code. For example, if a developer wants to change the layout of a button, he or she needs to locate the code linked to that button. The developer therefore debugs the system manually in order to locate the code. There are no technologies that could automate this debug process. However, we planned to develop a tool that could comment the body of the method. For the commenting out process we decided to have it on a method level. This is because commenting out the whole class could lead to more errors during compilation whereas commenting out line by line is also not practical as it would consume much time. Thus, we believe that studying the impact on a method level is more meaningful and reached a conclusion that development of such tool could be studied during the research thereby answering RQ1. This also allows us to explore the challenges involved technically.

As the next step, we wanted to examine the results of the tool after its development. As the tool intended to be developed for the research is in its prototype version, we decided to study the possibility of generating documentation for UI components in one screen (starting screen). We wanted to know if there would be UI components that get affected by any of the code manipulations or whether it will be of results that are less significant. We wanted to study the quality of the results and the

type of methods that created them. So, we decided to perform an analysis of the results about to be obtained from the study. This practical aspect of the analysis would relate to RQ2. We wanted to know if the results obtained have the potential to be considered as a comprehension technique that could benefit the maintainers or software developers. In order to analyze the perception of developers on this technique, we planned to set up a qualitative study of the results. This way we wanted to find the answer for RQ3 which could be used as feedback for further researches.

3.1.3 Design and development:

To answer RQ1 we had to analyze how this new comprehension technique could be implemented. In order to do this, we decided to check for changes exist in UI if a method in the project is commented out. There are visual GUI testing tools which provide the comparison of images and simulate user actions. Upon studying these tools, we found that 3rd generation category of GUI testing tools offers the opportunity to interact with UI components based on image recognition [11]. Sikuli[12], provides the options to check for a pattern on a screen and to compare images. The very same GUI testing tool also provides the ability to read or write a file which could be used to comment a method in the code. In Sikuli IDE scripts are written in 'python' [29] language. While learning about Sikuli we discovered that the API provided by said GUI testing tool offers all the features provided in Sikuli scripting via Java. We decided to choose Sikuli API in Java to develop the tool because reading or writing a file is more straightforward using Java in comparison to writing python scripts to achieve the same for this research. Moreover, there is no performance difference as Sikuli IDE uses Jython [71]. Java also offers more support in the form of external libraries to help in automating the tasks needed to be performed and also easy to perform debugging during the development process. For example, after commenting on a method, the system should be compiled and started. In order to do so, the whole project should be built (i.e. using build tools such as Ant, Maven [31, 32]). It is possible to programmatically run build files (i.e. execute build files) through Java whereas Sikuli IDE required creating complex scripts. The other advantage is the debugging support offered in standard Java IDEs such as Eclipse and IntelliJ. Sikuli API in Java is, therefore, more appropriate for this research study in comparison to scripting in Sikuli IDE.

During the design, we noted that it is necessary to observe the changes and record these changes as files. It is, however, complicated to trace these changes back to the method that caused the change. So, we needed a mechanism that is robust, and thus we decided to use the database to save the mapping of the changes in the UI and the method that is responsible for its creation. When the tool is being used the observed difference will be saved as a file, and its location will be stored in the database. For every method, if there are any changes observed the details corresponding to the change would be stored. The database chosen is SQLite [33] as it offers the basic database operations with the support to be embedded in the tool itself. As the tool being developed helps in tracing the source code to the UI components, it will

be named 'UI - Tracer'. Detailed description of the implementation is provided in chapter 4.

3.1.4 Demonstration

This activity is performed as a formal evaluation of the solution. This section of the thesis, therefore, relates to RQ2 as it deals with the tool's ability to identify important methods for the starting screen's UI components. In this phase, the tool developed has to be tested on a project. As this research is based on studying the impact a method has on UI; we decided to choose a project which has a user interface. The project to be analyzed must contain visual components, i.e. certain UI that can be tested. Therefore, we selected a project called 'Arts of illusion' which is a free and open source project which provides 3D modelling and rendering of images/objects. The user interface has components which could be used by the user to create 3D images or objects based on the needs of the user. For this reason, we decided to choose this UI system for the study. This project has around 398 classes in it.

The tool developed is decided to be tested on this 'Arts Of Illusion' application's source code and results from this tool will be evaluated manually in relation to the following questions and purposes:

- Evaluate the quality of differences in images obtained as a result of the tool.
- Identify if all the UI components present in the starting screen disappear change or go missing when the tool processes them.
- Identify whether multiple methods are associated with the impact on a single UI component.

3.1.5 Evaluation

The purpose of this activity in this thesis is to answer research question RQ3 and thereby finding out how software engineers or developers perceive the new comprehension technique proposed. In order to obtain their perception, we decided to perform a usability evaluation of the results from the tool and study them qualitatively. By evaluating the results, we wanted to study and understand the areas of improvement which could be useful for future research works.

There are different options to choose when framing an evaluation study. The research setting to use in the study is one among them. The research setting could be controlled or natural depending on various aspects of the study. Here we discuss the choice of using natural settings in this thesis. In controlled experimental investigation the variables of interest ("independent variables" and their measurable outcomes the "dependent variables") have to be studied in controlled settings [68]. In case of our research, measuring the impact this technique has on developers in a controlled setting requires a heavy setup. For example, measuring the difference in time taken by participants when they use the new comprehension technique and

the traditional comprehension techniques. So the study performed as a part of this evaluation phase is done in natural settings [69] and not performed as a controlled experiment.

The research question RQ3 is connected to the perception of the developers, making it qualitative would help us in exploring people's views on their understanding and experiences [47]. These factors led us to choose formative evaluation approach for performing the evaluation as formative evaluation relies on qualitative methods [67]. Formative evaluation helps in enhancing a program, policy group or product [70]. According to Stetler, "formative evaluation is herein defined as a rigorous assessment process designed to identify potential and actual influences on the progress and effectiveness of implementation efforts" [64]. Results from formative evaluations typically include opinions and suggestions. Identifying these outcomes are crucial as they will help in meeting the goals of the evaluation. Formative evaluation is also called as exploratory evaluation [62]. Thus using this approach qualitatively made it possible for us to elicit "What", "Why" and "How" questions related to the comprehension technique under study [47].

Motivation: Any evaluation needs to have realistic goals and so prior to choosing the type of evaluation we wanted to decide on the goals and the outcomes expected of the evaluation. The proposed comprehension technique in this research is new and we wanted to know if such a comprehension technique through a visual documentation (generated by the tool) could be used by developers. Also, as mentioned in RQ3, we wanted to know how developers or maintainers perceive the documentation created using the technique under study. Thus, as we are concerned about people's view on the results, we wanted to know primarily the following and have set them as evaluation goals.

- Evaluate whether participants could understand and use this technique.
- Identify whether the participants could find scope for usage of this technique among other comprehension techniques.
- What are the advantages, uses and weaknesses of the new comprehension technique over other standard comprehension techniques?

Since the participants have no prior experience with this approach it is tough to understand their perception of the new comprehension technique. This created a need for the participants to understand how this new technique works leading us to perform usability testing [62] prior to the identification of other outcomes such as advantages, uses and limitations of the technique. Usability testing has become a standard and an integral part in formative evaluation [66]. This involves tasks or scenarios that the user has to perform during the study. In other words, tasks are activities the participants (i.e experimental units) of a study should perform as a part of an evaluation. Using the tasks in usability testing we wanted to evaluate whether participants could understand and use this technique. Moreover, through the usability tasks, we wanted to understand if participants could find usage areas to use this technique among the existing standard techniques.

After knowing whether the participants could understand the technique we wanted to identify other outcomes such as advantages, limitations and uses of the new comprehension technique. By identifying the advantages of the results it would help us to emphasize the importance and contribution of the newly proposed documentation. Knowing the limitations would help in improving the documentation technique and also by finding the usage scope of comprehension technique will help in knowing the usage scope of this technique among other standard comprehension techniques. The above-mentioned outcomes of the evaluation advantages, uses and weakness of the of the subject are studied qualitatively after the usability testing.

Procedure: Usually usability testing is done to evaluate usability of software systems and more commonly on systems with user interface. In our case as the subject under study are results that has no UI we adapted the usability testing and performed it on the results we obtained from the tool. From [62] and [63] the following activities are performed in this research to perform the evaluation of results obtained from the tool.

- Choosing experimental units
- Evaluation tasks
- Data collection
- Analysis of data

Choosing experimental units

In this section, we will discuss how the participants are chosen for the study and about the sampling technique we followed for choosing participants. All the participants who took part in the study are Master's students of software engineering programme of the Chalmers University of technology. The participants are students of a course "Software evolution project", a 15 credits course that primarily revolve around software quality and software maintenance. The course contains lectures and lab projects based on Software Quality and Software maintenance. Also, it is a pre-requisite for the course that the students should know Object Oriented programming and Software quality. It was eight weeks already into the course by the time the study was done and it is an added assurance that the participants knew software quality and maintenance from the lectures and lab projects of the course. This gave us the confidence about their knowledge on software development, maintenance and common comprehension techniques.

The students who participated in the study also have good English knowledge, and this made us carry out the process in the English language. They are also new to 'Arts of illusion' application thus giving us room for knowing how they approach or use the results from the tool. As a part of the course, the students were divided into groups of 5-7 people per group for the lab project. The lab was scheduled for 8:00 -12:00 hrs and when we performed the study, participants were chosen using Convenience Sampling technique [46]. This was done as we did not have time to approach each group of students and perform the study in the available time. So,

we approached the students of the course group by group seeking whether they are interested in participating in our study which is related to program comprehension. We presented them about our research and also about the evaluation study that we have planned to perform. The participants were informed by us that approximately 30 minutes would be taken for performing the evaluation study. The participants had the freedom to decline their participation in their study. Few students responded that they did not wish to take part in our study as they had to work on their lab assignments. In these cases, no participants were compelled to participate in our study. We ensured that verbal consent is obtained from the participants before they are chosen for the study. The process of obtaining consent from the participants was performed with the teacher of the course (Software evolution project) present. The course supervisor assured the students that their grades are not impacted anyway despite their participation or non-participation in the study. In total 24 participants took part in the study out of which two of them were teaching assistants from the course and rest of them being students of the course.

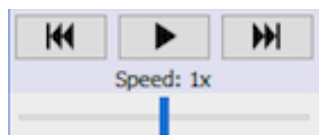
Evaluation Tasks

To perform usability testing, tasks have to be performed by the participants. The main intention of this usability testing is to understand if the participants could use the results for comprehension purposes and elicit their opinions. Tasks are categorized into three categories [65] and they are prescribed tasks, participant defined, and open ended tasks. In prescribed tasks, the researcher frames the tasks that have to be performed by the participants. In contrast to prescribed tasks, the participants are made to frame the tasks in participant defined category. Whereas in open ended tasks the participants are allowed to use the subject under study without any specific demands. In this study, we made use of prescribed tasks as the technique is rather new to the participants. The participants therefore were given two tasks and they are as follows.

- *Task 1* Hide the camera object (shown in figure below) from the display pane only during start-up without affecting other functionalities of the ‘Arts Of Illusion’.



- *Task 2* Change the default playback speed in the playback slider.



These tasks are prior tested with a person who is familiar with Software Development, to ensure that these tasks are solvable in the limited time we had for the study. Both the tasks could be solved in multiple ways. There is no strict solution

we expected the participants to perform as the idea here is to meet the evaluation goal of understanding whether the participants could comprehend using this technique.

The first task was framed in a way such that the solution to it should affect the UI alone instead of the functionality of the ‘Arts Of Illusion’ application. This way we wanted to see if the user can comprehend using the new comprehension technique to work on UI related changes or issues. There is a method named ‘visibility’ in the Camera.java class which is connected to the task 1. This method is also in the results recorded by the tool as the camera object disappeared when the tool commented the body of the method ‘visibility’. As both, the name of the object(Camera) and the class(Camera.java) are the same we felt that this task is more straightforward for solving. Moreover, we observed that the task could be solved using the same technique that we used to produce the result i.e. commenting the body of the method. To avoid this risk, we decided to add a small constraint to the first task. So, the results from the tool will just lead to the code location and the participants should understand the code logic to perform the solution. So instead of just having the task as ‘hide the camera object’ we added a small constraint to the task by avoiding the participant to comment the whole method.

The second task affects both the UI and also the functionality of the ‘Arts Of Illusion’ application. The default playback speed is set to 1 in the application logic of ‘Arts Of Illusion’. Unlike the task 1, the task 2 cannot be solved directly by commenting the method associated to the result in the documentation. Using the results, the participants could identify the starting point of comprehension but the solution could not be achieved by commenting the method completely. Commenting the method here will remove the ‘playback speed’ object completely from the UI which is not the expected solution for this task. Thus, we did not add any additional constraint to task 2 unlike task 1.

In addition to the results obtained from the tool, we also provided standard documentation from the official ‘Arts Of Illusion’ tool, SonarQube generated documentation of ‘Arts Of Illusion’ and also the documentation generated from Doxygen. We provided this extra documentation as it would help the participants to use them if they get stuck with not knowing how to proceed in solving tasks using the results from the tool. Moreover, as one of the objectives of this usability testing is to identify whether they could find usage scope for this technique among other standard techniques, providing them with other resources helped us in achieving this objective. The participants had the freedom to choose any documentation they wanted to solve the tasks.

While performing usability testing using tasks, the participants were divided into groups of 2 to 3 persons, and they were allowed to work on the tasks as a group. As discussed for solving the tasks we also provided them with results generated from the tool that is created as a part of the study. As the next step, a small brief about ‘Arts Of Illusion’ tool is given to the participants. We showed the participants the actual

tool ‘Arts Of Illusion’ in a machine (laptop) and demonstrated what it does. After this step, we introduced the participants about our research, i.e. about what we are trying to achieve by developing a tool “UI Tracer”, the results we obtained and how we plan to use them as a documentation technique for UI developers. We had the results i.e. the images we obtained from running the tool on ‘Arts Of Illusion’ to them printed on a paper. The participants were provided with two machines, one running IntelliJ [50] and one running eclipse [49], as an aid for the task. The purpose of providing these machines was to give the participants the possibility to choose an IDE to work with. We also explained them about the questionnaire section that follows after this step.

Data collection

For data collection, we used questionnaires and decided to proceed against interviews to ensure that participants feel free to give honest and critical answers. The questionnaires were filled anonymously by the participants so there is no option to trace them back. To elicit the opinions and feelings about using this approach, we made use of standardized, structured, open-ended questions for the questionnaire method [48]. Having the questions open-ended gave us the advantage of participants not biasing their responses. Once the participants are done with solving the evaluation tasks, they were provided with a sheet containing the questions that follows:

1. Do you think UI-Tracer provides an advantage?
 - Yes •No •No opinion
 If yes what is the advantage?
2. Do you think UI-Tracer approach has limitations?
 - Yes •No •No opinion
 If yes what are the limitations?
3. When would you use UI Tracer & how would you use the information?
4. Think of different situations you might be in a system that is new to you, e.g. need to fix a bug m change a behaviour , evolve the system, refactor the system, new requirements are coming in ...
 What do you have to understand about the system, and when would you rather use which of the following tools?

Rather SonarQube (Visualization & Analysis)	Rather UI Tracer
Rather Documentation	Rather DoxyGen (Documentation & Call graphs)

Table 3.1: Question given to participants

The participants are given around 10 minutes of time to answer the questions in table 3.1. However, it was not a mandate that they need to complete it in the provided time.

The questions in 3.1 are chosen exclusively for deducting answers to the research question RQ3. Every question mentioned above are planned in a way that it could elicit the answer for RQ3 and aimed at meeting the goals of the evaluation. One of the goals of the evaluation is to elicit the advantages, weaknesses of this new comprehension technique over other comprehension technique. Since this comprehension technique is quite new we wanted to elicit the positives/negatives of this approach through these questions. For this reason, we started framing the questions in the questionnaire that revolves around the advantage of this comprehension technique has over other traditional comprehension techniques. This way we would know how the users would get benefited if they are going to use the approach. The next question is about the limitations this technique has over other traditional comprehension techniques. As it is a new technique, we wanted to know the areas of improvement and thus helping the next iterations of this research to know the key areas. This is also framed as a direct question as the first one. Following the elicitation of advantages and limitations of the technique we wanted to know how and when the participants would use this technique under study. So the question 3 in questionnaire 3.1 was framed concerning the usage situations of this technique.

Finally, we wanted to know the usage scenarios of this new technique among other comprehension techniques. This is because we already provided the participants with the tasks to work on using the documentation from official Arts of illusion, generated documentation from SonarQube and Doxygen along with the newly generated documentation from the UI Tracer. We wanted to elicit if the participants could think of situations where they would use these techniques specifically. The main idea here is to get as much of information possible about the technique under study thus helping us deducting the outcomes of our evaluation. This reason paved the way for choosing questions that are open-ended allowing the participants to answer what they feel and think about the technique.

Analysis

After getting the data from the questionnaire section it is necessary to consolidate and analyze it. So, we used thematic analysis as the method to analyze the data we had. According to Boyatzis [51], thematic analysis is a qualitative method that helps in identifying, analyzing and creating patterns within the qualitative data. Thematic analysis is a widely used method for the analysis of qualitative data [51]. After performing the data collection through questionnaires, we started analyzing the data using thematic analysis. This thematic analysis is performed using the six steps as mentioned in the research paper [52]. The following steps are carried out on the data collected and these are further discussed in detail in chapter 6.

- *Familiarizing the data:* Repeated reading is performed as the first step of the thematic analysis. This way it was possible for us to understand the depth

and the breadth of the content. Notes were taken to understand the overall data.

- *Generating initial codes:* Coding is performed after familiarizing the data. In this research the coding process is carried out manually instead of using software programmes to generate codes.
- *Searching for themes:* Themes are generated as the next step after coding. The codes obtained are grouped under broader themes and these themes are generated in a way that they represent the data collected.
- *Reviewing themes:* Refinement of the themes generated from the previous step is done here. All the themes generated are reviewed to ensure that the data and the themes cohere meaningfully.
- *Defining and naming themes:* In this step, all the themes generated are defined and explained. This way we establish the connection between the themes and the data.
- *Producing the report:* As the last step, the summary of the analysis is produced. Here we explain the findings in relation to RQ3.

4

Implementation

The prototype developed as a part of the research methodology followed is explained in this chapter.

4.1 Prototype Development

In this chapter, the implementation of the tool developed for answering the RQ1 will be covered. While developing the tool we planned in a way that could help the user by automating most of the manual tasks that is required to generate the documentation. The main tasks needed by the user to perform the approach we had is by manipulating the body of the method, build the program run the program and check for any differences in UI. We decided to automate each of these tasks in the tool we built. So, if we have to do the same task manually, then the following steps have to be performed.

- Comment the body of the method.
- Build the application.
- Compare for difference on the UI of the application and record them if different.

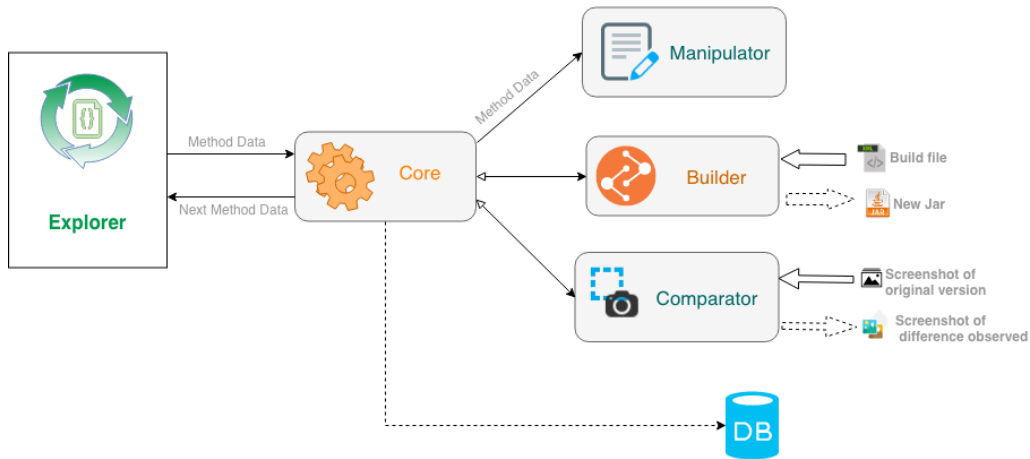


Figure 4.1: Architecture diagram of prototype

Based on these tasks we created the following components that form the basis of the prototype. The above-mentioned tasks have to be performed in order to generate differences. So, in the design phase, we decided to automate each of these tasks to help the user with the generation of the documentation.

Explorer

As the first step, we developed the component that could loop through the source code and get the body of the method. As the source code here is ‘Java’, we decided to parse the contents of a Java program. This step if performed manually would take a lot of time as the code base might contain a large number of methods. To avoid this, we made use of the library ‘Java Parser’ to parse the contents of a Java file. This API contains methods that could iterate through each class files and provide details about each method one after another. The library had functionality which was extended in the tool to meet the requirement of iterating through each java file one by one. The method `visit()` from the class `VoidVisitorAdapter` is extended to provide information about the method. The method can provide information such as the starting line of the method, the return type, the ending line etc. This information is used by other components for further processing. But it has to be noted that the location of the source code should be provided to this component. This component excludes the interfaces and abstract classes. This is done because the interfaces and abstract classes in Java are allowed to contain just method declarations. So, there is no method body in them, and so we discard these methods.

Manipulator

The next step in the process is to make use of the information provided by the previous component and manipulate the code. Here simple file processing is done using Java to read the code and manipulate it accordingly. We named this component ‘Manipulator’ as this part contains all the logic to comment the body of the method. The main task of this component is to modify the code and also to revert it. This is because once the code is modified, and after the necessary comparisons are done, it is important that we revert the changes we made. This way we ensure that all the comparisons recorded are based on the impact it has on one method.

<i>DataType</i>	<i>Modified_Value</i>
Onject/ Complex data type	null
int, double, float	0
String	null
boolean	false

Table 4.1: Values returned based on method return type.

In Java commenting the body of the method that returns a value will result in compilation errors. For this reason when the method information is obtained we check whether the method has any return value or not. For example, if there is a method that returns void then the whole body of the method could be commented. But if there is a return type then based on the data type that is being returned, the body of the method will be commented, and a string will be inserted at the end of the method. For instance, in case there is a method that returns a value of Object data type then the return value would be null. On the other hand, for a primitive data

4. Implementation

type, the value will be returned 0 or false according to the data type. The table 4.1 above shows the comparison of the return type that is manipulated. For complex data types, a null value is returned.

```
public void setGrid(double spacing)
{
    gridSpacing = spacing;
}
```

The above code snippet will be transformed into the following code where the whole body of the code will be commented out.

```
public void setGrid(double spacing)
{
    /*
        gridSpacing = spacing;
    */
}
```

For example , the methods that return a value will be transformed with a slight variation as discussed.

```
public double getDistToScreen()
{
    return distToScreen;
}
```

The above-mentioned method contains a primitive data type as the return type, and so the return type of the same will be modified based on the values from 4.1

```
public double getDistToScreen()
{
    /*
        return distToScreen;
    */return 0;}
}
```

Similarly, for the methods that return complex data type in the method will return a null value after commenting the body of the method.

```
public final Mat4 getViewToScreen()
{
    /*
        return viewToScreen;
    */return null;}
}
```

All the changes made have to be reverted before notifying the explorer for next method. In case of exceptions when making manipulation of the file, it is ensured that the method which was manipulated is reverted back to its original state. This way before moving to the next method we ensure that all the changes done previously are reverted.

Builder

The next task that has to be performed after manipulation of the file is to create a build of the system under consideration. For ‘Arts of illusion’ the build file is an Ant file. Ant API from Apache is used as the library to create the component

that executes the default target of the build file. This component gets the build file as input and executes the ant target. So, each time a method is manipulated by the manipulator component, we use this builder component to execute the build. The results of the build might vary depending on the method which is getting manipulated. For instance, commenting out few methods might cause build failures and other might result in successful builds etc. For each method, the build status of the application is saved in the database. For example, if there are any run-time exceptions due to the manipulation of the method, the build might fail to lead the creation of corrupt jar. Whereas in some cases the build might be unsuccessful and these data are saved in the database.

Comparator

The last task forms the base of the tool. If the Builder component generates a successful build, then the task is to run the newly built jar file and check for any differences that exist in the UI of the application. The outcome is completely dependent on the method that is commented. We used Sikuli API in Java to make use of its image processing abilities. It has powerful image comparing or recognition libraries that helped us in achieving what is required for the study. Once the Builder component notifies that the jar is built, the comparator component opens the jar and begins looking for any differences that exist between the actual version of the application and the newly built version. This component makes use of the actual version of the application to compare for differences. For doing so, the screenshot of actual version is passed as reference. So every time after the jar generated by the builder is opened the comparator uses the reference image for comparison.

This component uses the exists() method from the Screen class for finding any differences. If there are any differences, then the screenshot of the difference is captured and saved in the local file system following which the details of the image are saved in the database. If there are no differences, then no screenshot is captured. This component compares what is being displayed on the screen. It captures, records and compares based on what is being displayed on the screen. So irrespective of the language a project is built with this component could be extended or adapted to be made use.

Core

This component coordinates all the actions performed by the above-mentioned components and ensures that they are in sequence. In addition to this, this component also performs some checks before requesting other components. For example, if the build fails in the middle of the process, then it ensures that the manipulator reverts the changes to its original version and requests for next method to the 'explorer' component. The 'core' component is also responsible for updating the database with results according to the action performed in the other components. The status and the results of the other components will be noted and updated in the database. The core component after the completion of the operation performed by the builder component opens the jar for the next component 'comparator'. Only if the jar could

be opened by the core component, it requests the ‘comparator’ component to look for differences.

Database

All the images saved as a result along with the method responsible for creating them are saved in the database. The database in this prototype has three tables, and they are project table, method table and image table. The project table contains all the information about the respective project. In this case, this table will contain information about ‘Arts of illusion’ as that project is under study. The next table is Method table which contains information about methods of the corresponding project. The third table Image contains information about images that are recorded as differences by the ‘comparator’ component.

The ER diagram 4.2 shows the relationship between the tables and how they exist in the database.

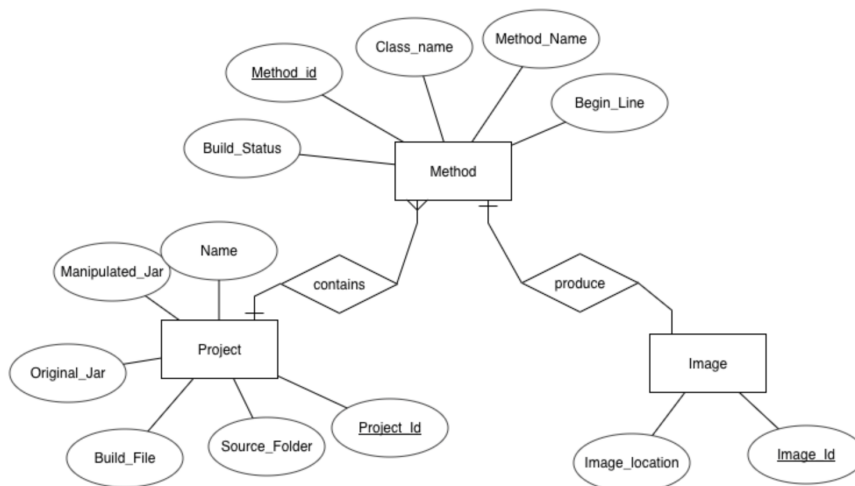


Figure 4.2: Chart to illustrate the results obtained from the tool

As shown in figure 4.2 the relationship between tables in the database is chosen with few considerations. It was designed in a way to add more projects if needed. Each project will contain many methods, so the relationship between a project and a method is one to many. Whereas one method might produce zero to multiple images, and that is the reason the table has one to unspecified relation.

The database is setup using SQLite as this tool can then make use of this local database in its machine. The attributes in the table are corresponding to the requirements of the project. The values from these tables can then be used to produce the mapping as intended using simple query.

4.2 Configuration

The tool developed could be configured to support other java based UI projects that have Ant build system. Some information is required from the user to begin its processing of the source code. In this section, we will outline the dependency each component has to perform its respective operation, and how these dependencies are obtained from the user.

- The *Explorer* component needs to know the location of the source code of the application that is under study. In this case, as it is the source code of ‘Arts of illusion’ application, the location of this particular code is a dependency of this component.
- The location of build file is needed by the *Builder* component. This build file will be used by the Builder component to run the default task associated with the project.
- The *Comparator* component needed to know the location of the screenshot of the UI of the ‘Arts of illusion’ application. These set of images will be used by UI-tracer tool to make the comparison.

For convenience, we added some basic features to the UI-Tracer tool to get all the information that is needed by these components as inputs. This information is stored in the database and used for a later purpose. Every information once stored in the database can be reused again for later purposes. For example, if the ‘Arts of illusion’ project is set up in the UI- Tracer tool, then the project setup information will be stored in the database and can be re-used again.

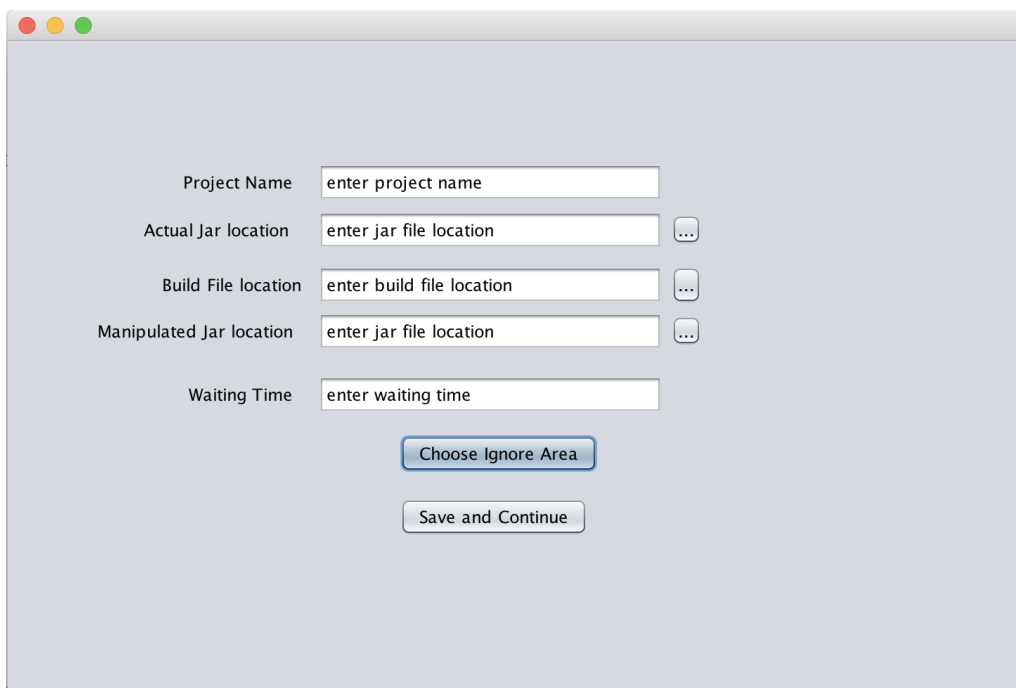


Figure 4.3: Project setup GUI of UI-Tracer tool

The figure 4.3 shows the image of the UI screen from the UI-Tracer tool that gets the input from the user related the project that he/she needs to trace. To provide the location of source code as input, we have a field that could be used to get the value from the user. The ‘Comparator’ component needs the screen-shot of the starting screen of the application that is under processing. We added a feature to the tool that could be used to take a screenshot of the application. To do so, we need to know the location of the jar file which is unmodified. The screenshot of the starting screen of this jar will be used by the tool as a reference when it looks for differences. The user has to ensure that the correct version of the application is used for this purpose.

The ‘Manipulated jar file’ field is to get the location of the jar file which will be generated after building the application. The ‘Actual jar’ field is to get the location of the jar which is not manipulated. We added two separate fields if the user has to pass different locations e.g. if the location of the build file is different from the actual version. Once these fields are provided, the user can choose the area that needs to be ignored. This setup takes the area that has to be ignored while taking the screenshots of the actual jar. This is done as there are areas of the screen that needs to be ignored like ‘Task Bar’ in windows, dock in Mac Os. This way we eliminate the comparator component comparing unwanted portions of the screen when it looks for a difference. The value in the ‘Waiting time’ field will be used every time as a static value for opening the jar by the core component in the later stage of the process. The ‘core’ component will wait only the specified time and if the jar is not opened it will continue with the processing of rest of the code. On clicking the ‘Save and continue’ button, the user will be prompted with a screen as shown in image 4.4. The ‘OpenJar’ button will start the jar from the location specified by the user in the GUI 4.3.

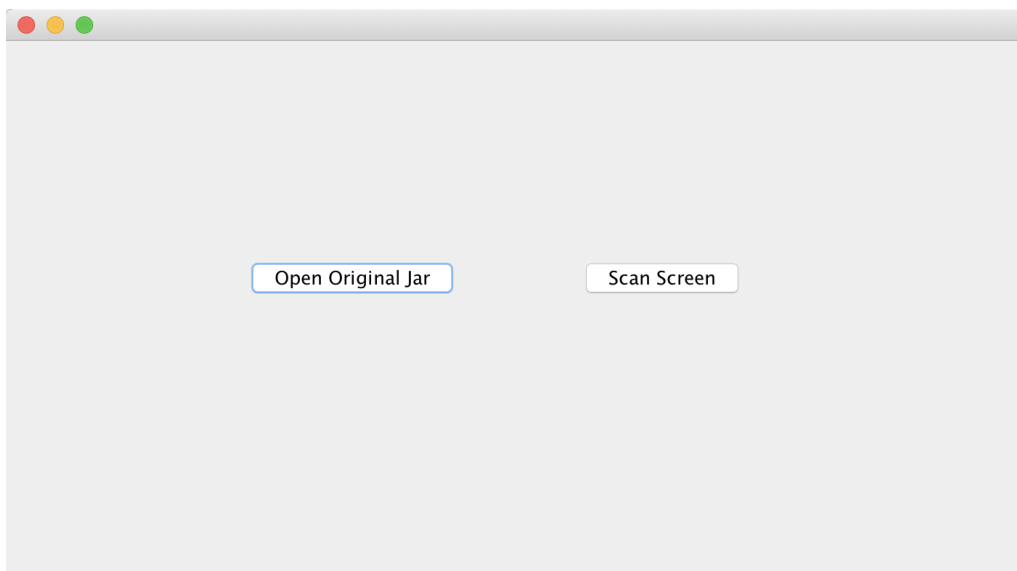


Figure 4.4: Project setup GUI-2 of UI-Tracer tool

On clicking ‘Scan screen’ will take a screenshot of the screen that appears on the screen. The below image 4.5 shows the screenshot of ‘Arts of illusion’ application

with the un-manipulated code. This screenshot is divided into four portions during the initial setup of the tool. These divided images will be used as patterns to check whether the manipulated version run by the tool contains them.

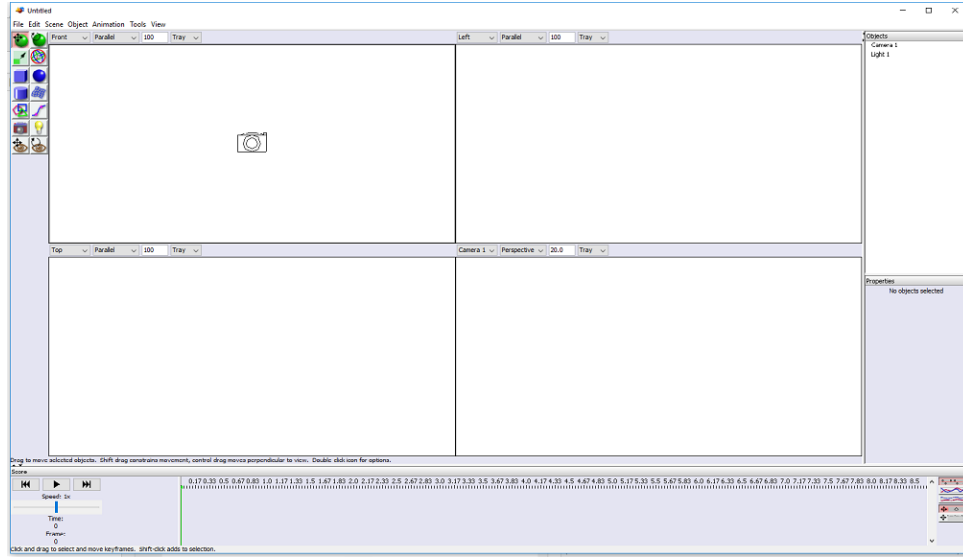


Figure 4.5: Screenshot of the ‘Arts of illusion’ application

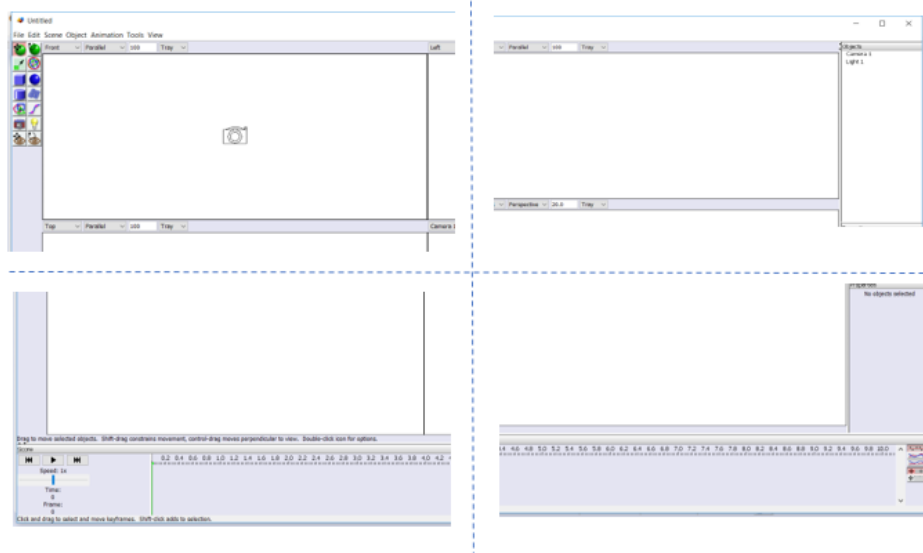


Figure 4.6: Image to illustrate how the screenshot in image 4.5 is divided

The screenshot taken is split into four parts as shown in figure 4.6. This helps in easily finding differences i.e. if there is any difference in the first quarter the *comparator* component will record only the first quarter. Now as the area to check for is low, this adds convenience in the end after the results are obtained. All this information can be set using this GUI as one-time setup and can be reused for later purpose. However, this GUI and functionality are designed for the convenience of the

researcher to perform the study, and the whole setup is in its basic version.

The image 4.6 are the split images of the screenshot of the ‘Arts of illusion’ application. Each image will be compared one by one against the application using Sikuli. The first quarter will be checked for its existence against the application running on the manipulated code. If any changes exist, then there is an impact in this portion of the image. Similarly, the rest of the images are compared against the application running on the manipulated code. When there is a change that portion will be saved as an image and stored in a file. The location of that file will be stored in the database for future reference.

4.3 Process of Tracing a Method

In this section, the steps involved in creating a result will be explained. Consider that the tool is about to analyze the method *visibility(BoundingBox bb)* from Camera.java class of ‘Arts of illusion’ application. As the first step, the method details are fetched by the explorer component and it is passed to the component ‘Core’. The runCore() method of the ‘Core’ component is executed which sequentially handles and coordinates the activities. To begin with, the core component asks the ‘Manipulator’ component to add a comment to the method *visibility(BoundingBox bb)*. This is done through ‘addComment()’ method of the ‘manipulator’. Once the method is manipulated the ‘executeAntTask()’ of the ‘Builder’ component is called and it executes the build file associated with the project. If the build fails at this stage then the result is stored as ‘Build failure’ in the database by the ‘core’. On the other hand, if the build is passed, the jar file created is opened. The starting screen of this version is compared for differences by the ‘comparator’ component. After recording the differences, the ‘core’ component uses the ‘manipulator’ to revert the comments and performs the same set of operations for the next method.

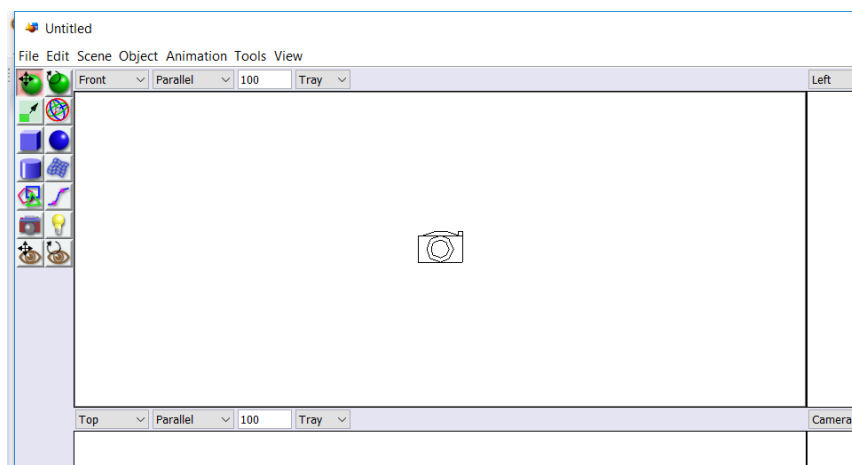


Figure 4.7: First quarter of the image from ‘Arts of illusion’ application with un-manipulated code

For example, consider the code has created an impact as follows in the manipulated version of the application as shown in figure 4.8. Notice the camera icon in the first quarter of the screen is missing. This difference will be monitored during the check performed by the ‘Comparator’ component of the tool.

The first quarter of the image from 4.6 which is shown in 4.7 and this will be checked for its existence in the image 4.8. Now the ‘comparator’ component will notice the difference as the camera object is missing in 4.8. Thus the impact created is noticed and saved as an image as a file as shown in figure 4.9. As the difference is corresponding to the first quarter only that portion of the UI will be recorded as the difference. The corresponding details are also stored in the database. Now the subsequent quarters of the image from 4.6 will be used by the ‘comparator’ component to see if it exist in the UI of the application with manipulated code.

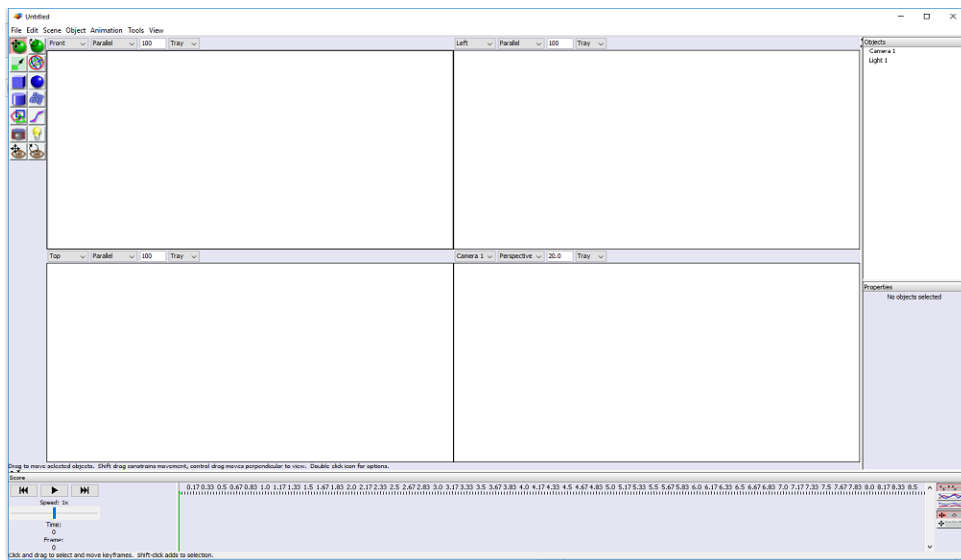


Figure 4.8: Screenshot of the ‘Arts of illusion’ application with manipulated code

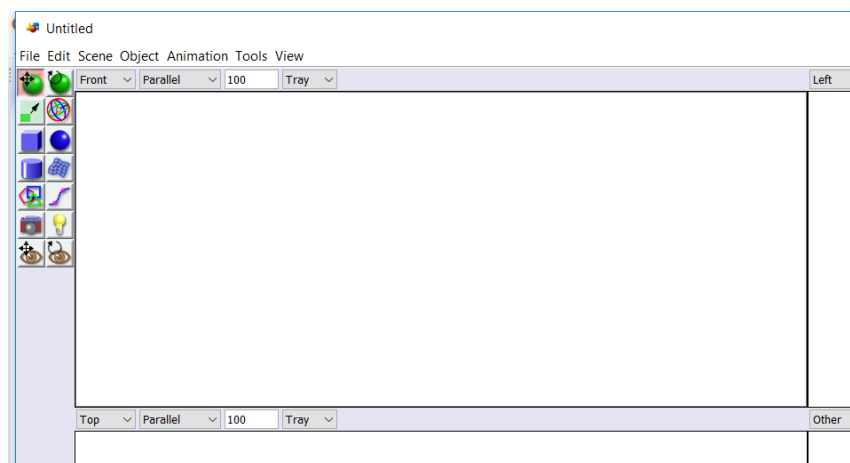


Figure 4.9: Image saved as result by UI-Tracer

After checking for the existence of original application's screenshots in the manipulated application, the core component uses the manipulator component to revert the changes done to the Camera.java file. Once this step is done the next method will be looked upon by the 'explorer' and the above operations are repeated until all the methods are processed by the tool.

The sequence of steps involved in the creation of a result is illustrated in the sequence diagram 4.10. Each component is represented by the rectangle boxes in 4.10. The operations are represented by the text above arrows pointing to the right-hand side. The arrows pointing to the left side return data from the methods.

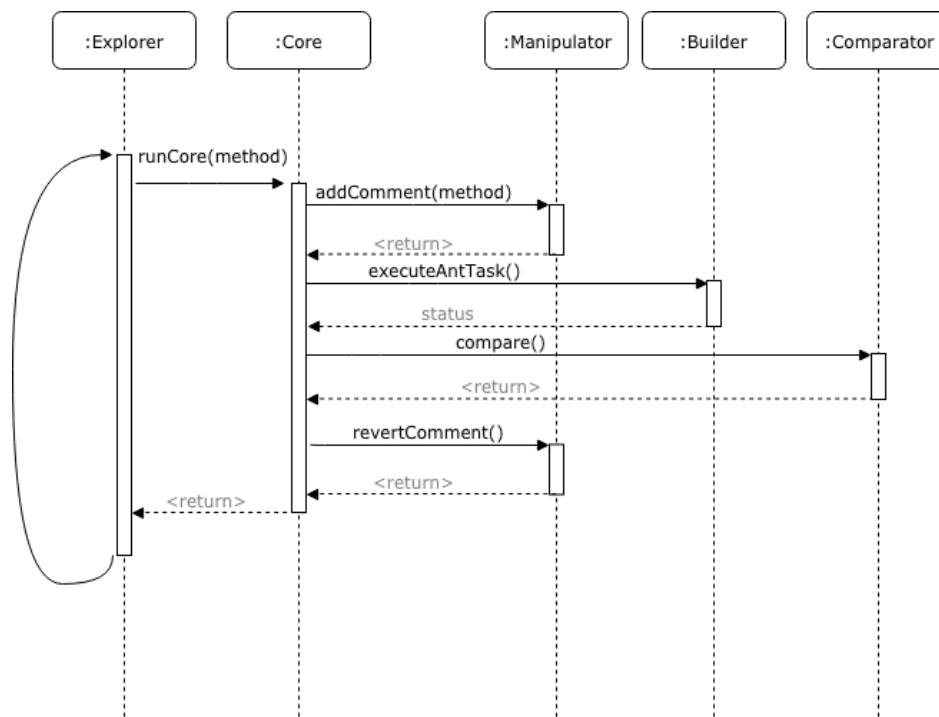


Figure 4.10: Sequence diagram of UI Tracer tool

5

Results

This chapter will cover the contents that answers the research question RQ2. At the beginning of this chapter, we discuss the significant components of the ‘Arts of Illusion’ UI followed by the analysis of the results obtained from this tool.

5.1 UI of Arts Of Illusion

The UI of ‘Arts of Illusion’ application contains many interactive UI components [53]. These include buttons, panels, drop down list etc. In this section, an overview of the UI of ‘Arts Of illusions’ will be given. The figure 5.1, is the screenshot of ‘Arts of Illusion’ application.



Figure 5.1: Screenshot of ‘Arts of Illusion’ application

The main components of ‘Arts of Illusion’ will be discussed below. These components are obtained from the official documentation[54] of ‘Arts of Illusion’ application.

The *Menu bar* panel contains clickable buttons. They appear at the top left of the UI and clicking them will drop menu items as a list that has buttons encapsulating functions respective to each menu item. The image 5.2 represent the Menu Bar of the ‘Arts of Illusion’ application.

File Edit Scene Object Animation Tools View

Figure 5.2: Menu Bar of ‘Arts of Illusion’

Tool Bar contains the commonly used tools that allows the user to create objects and work on them [54]. The figure 5.3 shows the representation of Tool Bar in the application.



Figure 5.3: Tool Bar of ‘Arts of Illusion’

The main window of the ‘Arts of Illusion’ application is divided into four main components. These components are called as the view windows. These view windows represents different views in four areas. The figure 5.4 represents the view window of ‘Arts of Illusion’ application. When the application starts the default setup is that one view window contains the ‘Camera Object’ as shown in the centre of the 5.4 . On top of each view window there are controls to modify the behaviour of the view window. These are known as ‘*View Controls*’

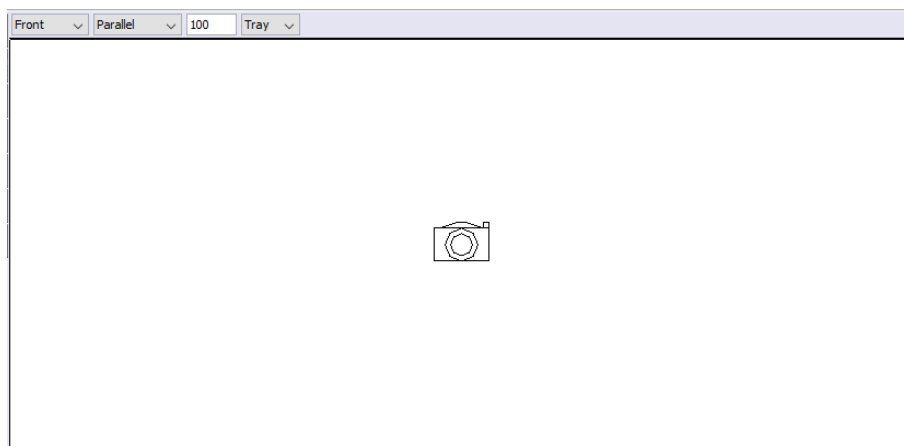


Figure 5.4: View window of ‘Arts of Illusion’

To the right of the main screen of the ‘Arts of Illusion’ application there are two panels. One is the Object panel and the other is the Properties panel. The object

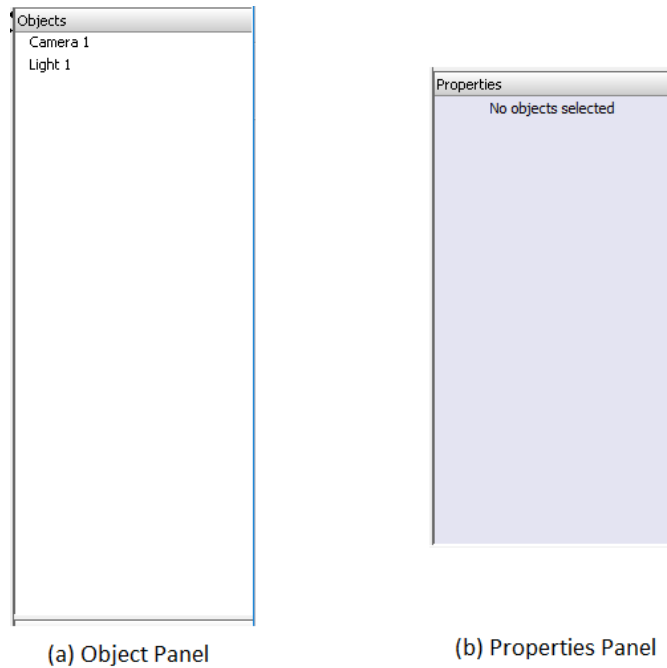


Figure 5.5: Object panel & Properties panel of ‘Arts of Illusion’

panel represents the objects that are in the view whereas the properties panel is about the properties of the objects that are in the view. By default when the ‘Arts of Illusion’ application is started, the Camera and Light object is there in the objects panel at the start of the ‘Arts of Illusion’ application. The figure 5.5 shows the image of the Object panel (Fig a) and Properties panel (Fig b).

The ‘Art of Illusion’ controls animations through a score or timeline on which various actions can be mapped. For controlling this the application has a Score panel on which it has a Scale and controls associated with it. The figure 5.6 shows the image of ‘Score panel’ from the ‘Arts of Illusion’ application.

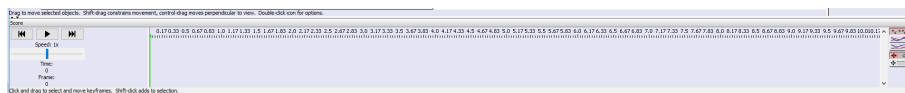


Figure 5.6: Score panel of ‘Arts of Illusion’

5.2 Overview of Results

The UI Tracer tool during its run manipulated the Arts of illusion code and generated results. The tool was run on 5348 number of methods of Arts of illusion application. So overall the core component had to loop through 5348 times the code commenting each method with an intention of finding differences in the UI. The tool took approximately 28 hours to produce the results and complete the processing on

‘Arts of Illusion’. Based on the results obtained from the tool we classified them into the following categories:

- Build failure.
- Successful build and failed to open jar.
- No differences found.
- Differences found recorded as images.

The classifications will be discussed one by one:

5.2.1 Build failure

Build has failed for 182 times during the run of this tool on ‘Arts of illusion’ source code. We performed analysis on the methods that produced this set of results. Many methods caused this scenario because of the removal of the body of the constructors. These are the constructors of classes that extends a parent class with no default constructor. Here in the source code of ‘Arts of Illusion’ many constructors have calls to the super constructors. When these methods are commented they produce compile-time errors and thus the build failed as result. Maybe a workaround could be considered for the future work to avoid this issue. These set of methods are also considered with no significance as it is not adding any value to the approach proposed.

5.2.2 Successful build and failed to open jar

The indicator of a successful build is that there are no compilation errors when building the ‘Arts of Illusion’ system. However, there are cases when we get a successful build but the jar produced could not be opened due to some run time exceptions that have occurred due to the removal of the body of the method. There are 60 results produced by the tool that falls under this category. It has to be noted that for methods that return a complex object, we commented out the body of the method and added null as the return value. These null objects are unexpected by the caller and has caused run time exceptions and thus jar fails to load. Thus, these set of methods produced results that cannot be considered significant for this research.

5.2.3 No difference found

Of the total 5348 methods processed by the tool, 4954 methods did not produce any differences in the UI of the starting screen of the application. As the focus is to find visible differences only on the starting screen of the application, we obtained this high number of 4954 with no variations. It has to be inferred from this result that the build is successful and the jar could be opened for 4954 times and no differences were recorded by the comparator component. We did not investigate the behaviour of these methods as it is a time-consuming task and it yields little benefit to the focus of this study.

5.2.4 Differences found

We found during our analysis that there are 217 methods that produced visual differences on the UI of the application. The comparator component has picked these differences and recorded them as images. The output of the comparator component recorded as visual differences is 503 images. As mentioned before while taking the screenshot and recording them, the image is split into four quarters for easy traceability in the later stage of mapping. For example, if there are any differences in the upper left side of the UI of the application then that part alone will be saved as an image. This has led to the creation of more than one image for a method if there are multiple differences located in more than one part of the UI.

There are methods/constructors that have caused more than one image as a result and that is why the number of methods that produced the results is not corresponding to the number of images obtained as output.

Summary

The tool produced 503 images as differences. The tool produced images into parts for easy navigation. The figure represents the number of sets obtained from the set of images categorization of images obtained as result.

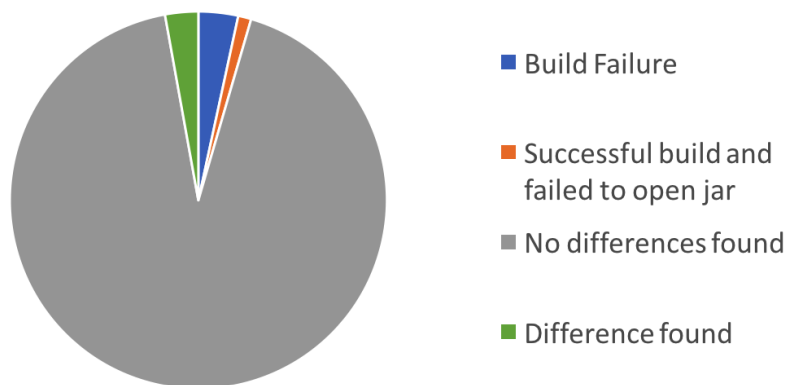


Figure 5.7: Chart to illustrate the results obtained from the tool

Type of result based on methods	in Numbers
Build failure	182
Successful build and failed to open jar	60
No difference found	4954
Difference found	152

Table 5.1: Types of results produced based on methods

5.3 Quality of images recorded

As discussed, on running the tool UI tracer on ‘Arts of Illusion’ source code, we have obtained **503** images as differences. These images represent some significant changes but it is important to understand few factors such as the quality of the images obtained, the UI component that was changed and the source code responsible for these changes. Out of the 503 images obtained an extended analysis is done to see if all these images could be considered as valid. The following are some of the classifications that we did on these images and this classification is discussed below.

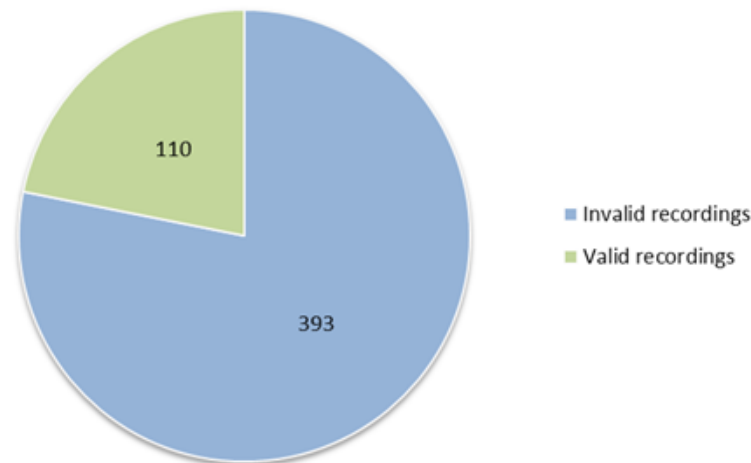


Figure 5.8: Pie chart of images recorded as changes

Invalid Recordings

On analysis of the images obtained as results, we observed that there are images that do not provide any inferences i.e we could not observe any component of the ‘Arts of Illusion’ UI in these set of images. We categorized these images that have no UI components under invalid recordings category. For example, in some recordings, we observed that the images are completely blank and in few others, the recording has been performed on the desktop background or any other application that was running behind ‘Arts of illusion’. These invalid recordings are caused due to failure in the loading of UI when the ‘Arts of Illusion’ application is started with the manipulated code. Run time exceptions that occurred when the application was started has created the failure of loading the UI. The comparator component has noticed that the ‘Arts of Illusion’ process is running and started its comparison believing that the UI is loaded successfully.

As already mentioned the UI-Tracer has three main tasks manipulation of code, compile and run the application and look for differences. After we run the application with the manipulated code the UI-Tracer tool enters a waiting phase. The user could configure this waiting phase and the purpose of this waiting phase is to wait for application with the manipulated code to load completely. Once the waiting phase is over we check if the already started ‘Arts of illusion’ application is still

running or whether it crashed due to some exceptions. If it is still running then we start the comparison process looking for differences. However, in some cases, the manipulation of code will result the application to fail the process of loading its UI. Unfortunately, in these cases, the application process is still alive but the starting screen is left unloaded. After the UI-Tracer tool steps out of its waiting phase it starts recording as the process is still alive but as there is no UI loaded it records the background application running with ‘Arts of illusion’ process as shown in figure 5.9.

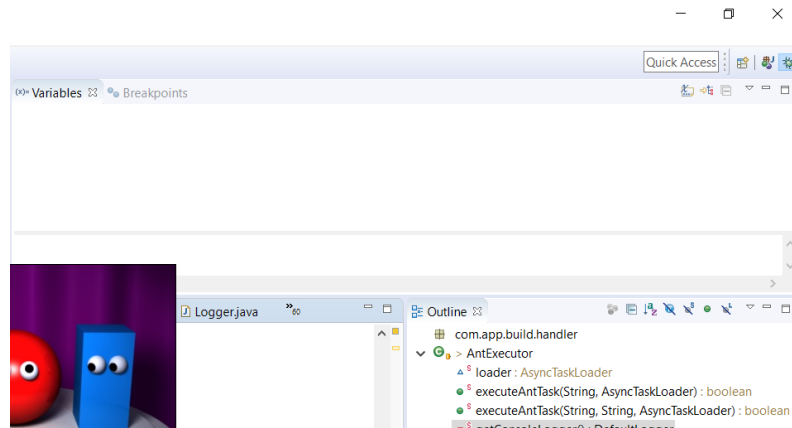


Figure 5.9: Example of an invalid recording that has the screenshot of background application

Another invalid recording is shown in the figure 5.10. Here due to run time exceptions such as ‘Null pointer exceptions’ the ‘Arts of Illusion’ application has failed to load the main window causing the UI Tracer to record it as a difference as well.

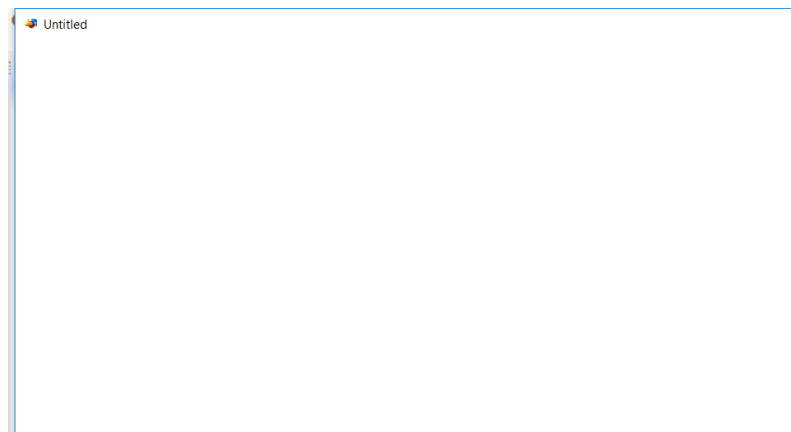


Figure 5.10: Example of image recorded as difference with blank UI

As the above-mentioned figures 5.9 & 5.10 are not changes in the UI that could be regarded as valid and so they belong to the category of invalid recordings. These recordings provide no significant information and thus could be discarded.

Valid Recordings

Images that have resulted in significant differences compared to the ones mentioned in invalid recordings are grouped under valid recordings category. These images have the potential to be used to identify the impact a method has on the UI. This is because the generated images recorded as differences are rather significant. Overall *110 images* are considered to be valid recordings as the changes in the UI components are significant. There are an interesting set of images that have resulted in minute differences between the actual version and the modified version. For example consider the example given in the figure 5.11: fig (a) represents the image of View Control component in the ‘Arts of Illusion’ application with unmodified source code. The second figure (b) in Figure 5.11 represents the image of the View Control component of the ‘Arts of Illusion’ application with modified source code. The modification done here is the commenting out of *rebuildCameraList()* method from the *ViewerOrientationControl.java* class by the UI Tracer tool. As shown in the figure the difference here in both the images is that the length of the component *Front*. This is recorded as a difference and as a result, this difference is recorded.

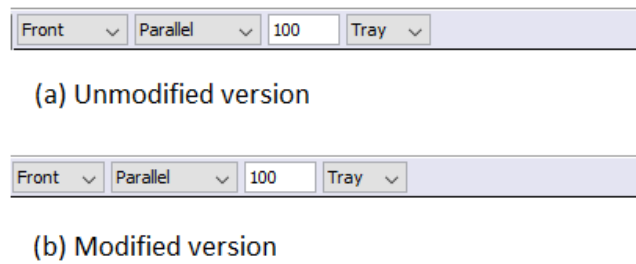


Figure 5.11: Difference in component length *Front*

The table 5.2 shows the overall summary of the changes recorded

Category of images	Number of images
Invalid recordings	393
Valid recordings	110

Table 5.2: Number of images grouped by their category

5.4 Mapping of Recordings

Analysis was done to find the type of methods responsible to create the recordings that are considered as valid. On examining the images that are grouped under valid recordings we found that there are few constructors that are responsible for resulting in these recordings. The other recordings observed were caused as a result of the manipulation of regular methods. The following table 5.3 shows the summary of the

methods and constructors that created the recordings as images.

It also has to be noted that some methods produced multiple images as changes. As discussed this is due to dividing the screen-shot into four parts for easily tracking the changes.

Class name	Method name	Number of Valid Recordings
ApplicationPreferences.java	getAnimationFrameRate	2
Camera.java	setScreenParamsParallel	1
Camera.java	setObjectTransform	2
Camera.java	visibility	1
LayoutWindow.java	LayoutWindow	1
LayoutWindow.java	createFileMenu	1
LayoutWindow.java	createEditMenu	1
LayoutWindow.java	createObjectMenu	1
LayoutWindow.java	createAnimationMenu	1
LayoutWindow.java	createSceneMenu	1
LayoutWindow.java	setHelpText	1
LayoutWindow.java	setTime	1
ObjectPropertiesPanel.java	ObjectPropertiesPanel	1
ObjectPropertiesPanel.java	rebuildContents	1
PluginRegistry.java	getResource	4
Scene.java	addObject	4
SceneViewer.java	rebuildCameraList	3
SceneViewer.java	setOrientation	3
SceneViewer.java	finishAnimation	2
SceneViewer.java	viewChanged	2
SceneViewer.java	updateImage	2
PositionTrack.java	isNullTrack	2
Score.java	setTime	2
Score.java	setPlaybackSeed	2
Score.java	layoutChildren	2
TimeAxis.java	paint	2
DefaultDockableWidget.java	DefaultDockableWidget	4
DefaultDockableWidget.java	paintBorder	3
DefaultToolButton.java	paint	2
GenericTool.java	activate	1
ThemeManager.java	getIconURL	4
ThemeManager.java	getIcon	2
ThemeManager.java	getAppBackgroundColor	1
ThemeManager.java	getPaletteBackgroundColor	1

ThemeManager.java	getDockableBarColor1	4
ThemeManager.java	getDockableBarColor2	4
ThemeManager.java	getNodeFromNodeList	4
ThemeManager.java	getAttribute	4
ToolPalette.java	paint	2
Translate.java	text	2
TreeList.java	addElement	1
TreeList.java	buildState	1
TreeList.java	paint	1
UIUtilities.java	applyDefaultBackground	1
UIUtilities.java	applyBackground	2
ValueField.java	ValueField	1
SoftwareCanvasDrawer.java	SoftwareCanvasDrawer	2
SoftwareCanvasDrawer.java	paint	2
SoftwareCanvasDrawer.java	drawBorder	1
SoftwareCanvasDrawer.java	renderLine	2
SoftwareCanvasDrawer.java	renderWireframe	1
ViewerOrientationControl.java	createWidget	2
ViewerPerspectiveControl.java	createWidget	1
ViewerScaleControl.java	createWidget	2
Vec2.java	Vec2	3
Vec3.java	length	1
ObjectInfo.java	isVisible	1
ObjectInfo.java	setVisible	1

Table 5.3: Mapping of Methods and number of valid recordings

5.5 Impact Observed on UI Component

The images considered to be valid are examined to identify the component that was affected along with the type of change in UI component. Few components have resulted in many numbers of differences whereas few components have resulted in few number of differences. The following table shows the summary of the number of changes observed under each component of the UI.

UI Component	No of changes
Camera Transformed or missing	25
Menu Bar	6
Tool Bar	2
Object Panel	7
Properties Panel	5
View window	18
View Control	14

Score	22
Miscellaneous	23

Table 5.4: Mapping of UI components and number of changes

This section will cover the changes observed per component and describe what the changes along to justify the answer for RQ2.

Camera Transformed or missing

The table 5.4 shows the number of times the components of the Arts of illusion UI has changed or modified.

For example, when the constructor of the class *Vec3.java* is commented and compared by the tool UI-Tracer, the image 5.12 is obtained. As shown the camera object is completely transformed.

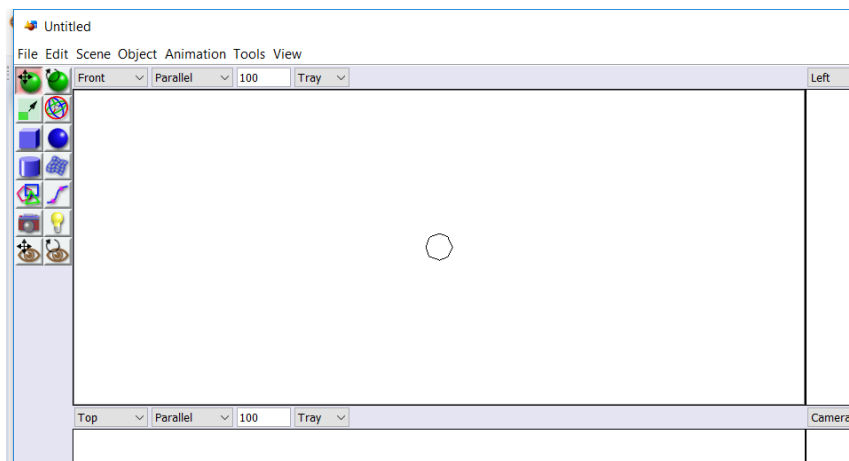


Figure 5.12: Camera object transformed

Similarly, the image 5.13 is obtained when UI tracer compared the build of Arts of illusion version after commenting out the method `renderline()` from `SoftwareCanvasDrawer.java` class of the source code. From the results of UI Tracer, it is evident that the camera object has been transformed multiple times. These include either the camera object missing from the view window, transformed into a different object or changed its position from its default position in the view window. All these changes are recorded as Valid changes by the tool.

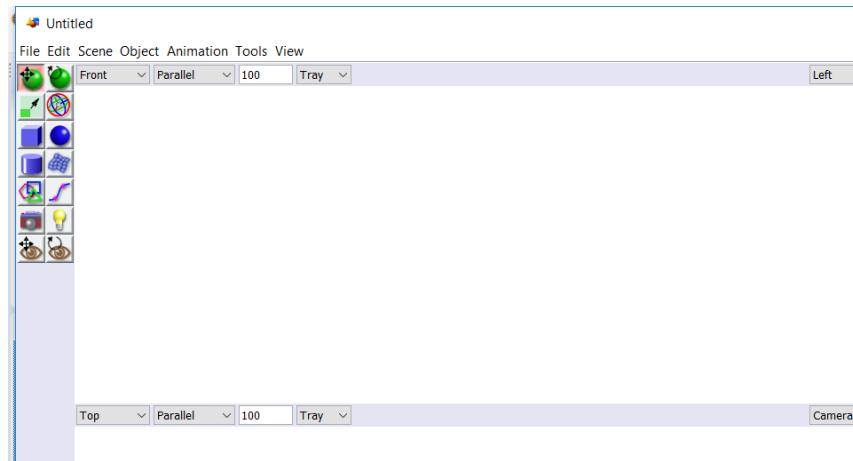


Figure 5.13: Camera object missing from view window

Changes in Menu Bar

Similarly, for the UI component Menu Bar from the Arts of illusion UI, we obtained 6 changes as valid ones. Shown in figure 5.14 is an image produced by UI Tracer tool when it processed the method *createFileMenu* from the *LayoutWindow.java* class. As shown in figure 5.14, the button 'File' from the menu bar is missing after the processing by the tool and this shows that this method is responsible for rendering the UI components associated with the button File in the 'Menu' bar. Similarly, we obtained changes for other buttons that are in the menu bar like 'Edit', 'Scene', 'Object' etc.

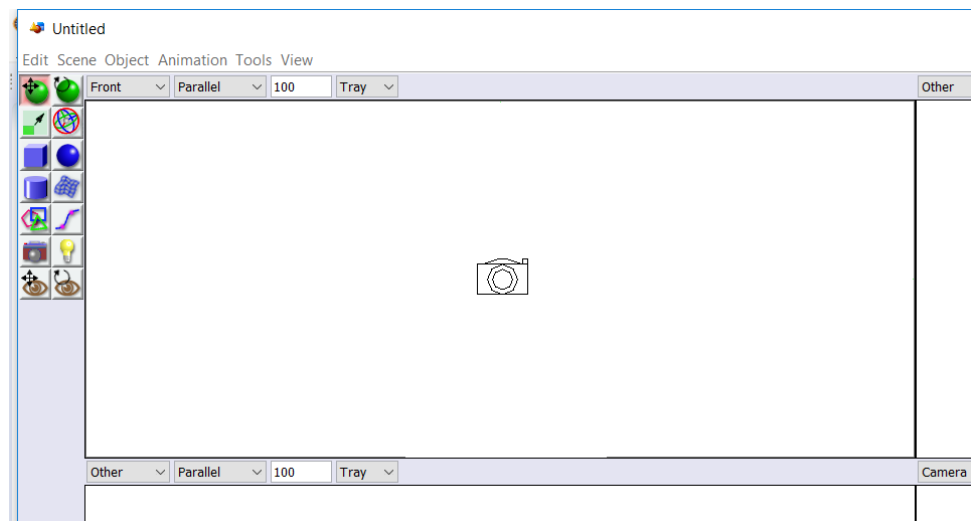


Figure 5.14: Result of 'File' menu button missing in Menu bar

Changes in Tool Bar

The UI component Tool Bar has produced least number of changes as result recorded by the tool. There were just two images produced as differences by the tool. The

figure 5.15 show the differences observed. The tool bar from the UI has disappeared completely in one of the image observed and in the other one the component is transformed into an error representation of the image as shown in figure 5.15. However, both the images could be considered as valid ones as they produce visibly clear differences as result.

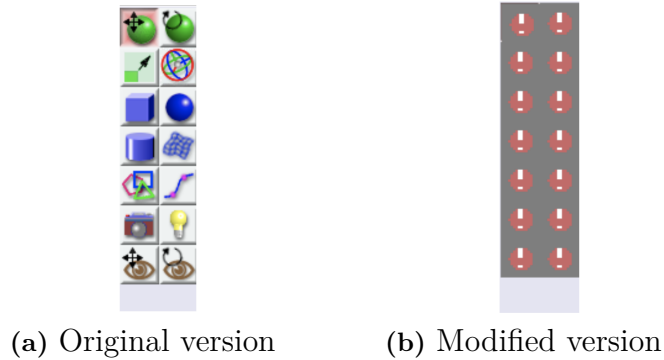


Figure 5.15: Comparison between original Tool Bar panel and result from tool

Changes in Object panel

The object panel component has undergone changes seven times during the processing done by the tool. Some changes happened in the title of the component ‘Object panel’. In one of the images obtained the title of the component has disappeared as shown in 5.16(c). Whereas in most of the other images obtained as differences, we observed that the elements that were under the component Object panel have disappeared as shown in figure 5.16(b).



Figure 5.16: Comparison between original Object panel and result from tool

Changes in Properties panel

Similar to the changes observed in object panel, the properties panel also had changes in the resulting images from the tool. It was observed that initially, the properties panel had a text ‘No objects selected’ inside the panel. In the image differences when we examined we found that one change is that the text underneath the Properties panel is missing as shown in figure 5.17.

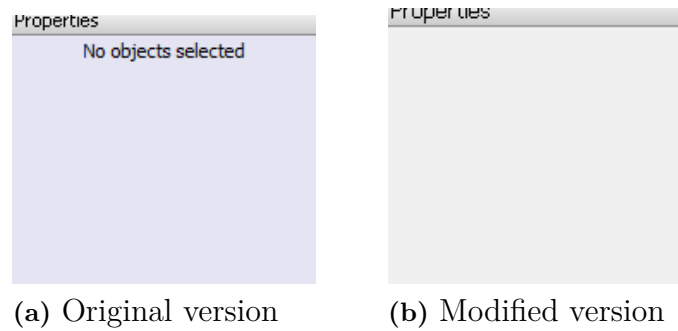


Figure 5.17: Comparison between original Properties panel and result from tool

Changes in View Window

View window covers most areas of the UI as shown in figure 5.18. During our analysis, we found that many images that contained changes in the component View window are due to the transformation of its size. These differences are recorded and produced as results by the tool. The other difference is when the camera object changes its position from its default position in the figure to some other view window. The figure represents this difference and this also serves as an example that one method might produce changes that affect more than one component in the UI. Here in this figure both the camera object and the view window is considered to have got changed when the tool did its processing. The camera object has moved from its default view window to another view window and thus making a difference in both the UI components as shown in fig 5.18.



Figure 5.18: Difference in component View window *Front*

Changes in View Control

View Control as discussed contains the UI elements for configuring the view window associated with it. This component has changed for 14 times during the processing by the tool. The figure 5.11 is one example where the view control has changed its length. The other changes include missing or transformation of values from the view control as shown in figure 5.19.



Figure 5.19: Comparison between original Control elements from view control and result from tool

Changes in Score component

Score component is one of the main components as it contains multiple interactive UI elements in it. The changes observed by the tool after it processed the code are mostly related to the missing or transformation of the UI elements in this component. For example, the figure 5.20 shows the UI elements to control the score component. The other image shows the difference in the set speed label i.e. the label 'Speed: 1x' is missing from the figure 5.20(b). Similarly, we identified 22 images containing differences in the Score component.

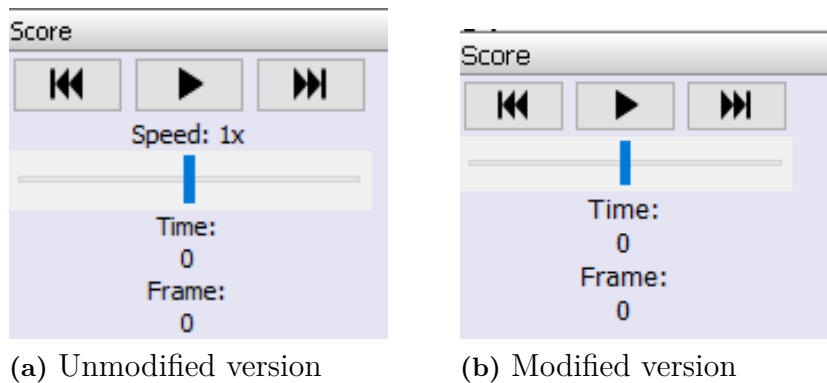


Figure 5.20: Comparison between original score component and result from tool

Miscellaneous Changes

There are 22 images that fall under the miscellaneous category. We have obtained images that have changed the color of the entire UI background. These images cannot be categorized as the results obtained because of the change in component or transformation of components of UI. For this reason, a set of images are grouped under category Miscellaneous. For example the following image 5.21 is a result of background color change in the application UI.

5. Results

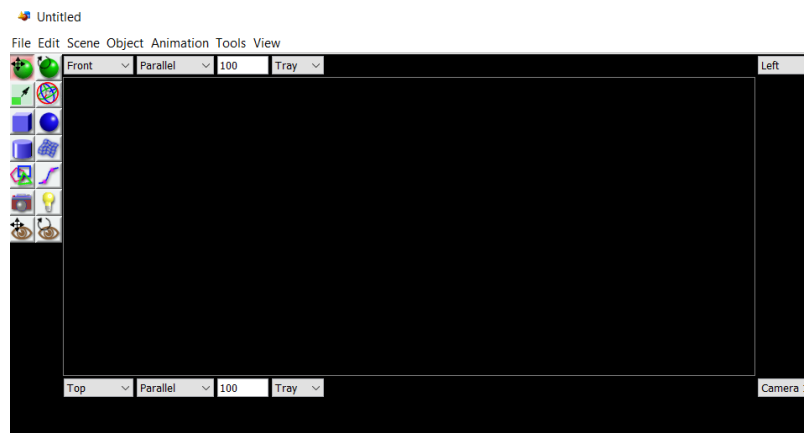


Figure 5.21: Difference in component View window *Front*

6

Evaluation

For answering the RQ3, we wanted to understand how software engineers perceive this technique and so we collected data from the participants through questionnaires. The data was collected qualitatively and this created a need for performing an analysis. We used thematic analysis to analyze the data collected and this chapter will cover the steps we performed to do the analysis.

6.1 Analysis of Data

As mentioned in the Methods chapter 3, tasks were given to the participants to see if they can understand and use the technique. This is done as a part of usability testing where the participants could experience the technique. All the participants who took part in the study managed to solve the tasks using the new comprehension technique provided to them. The participants successfully used the results provided to them as a new comprehension technique in spite of having the option to use other standard comprehension techniques. This proves that the participants understand and are able to use it for comprehension purposes which is one of the evaluation goals. After solving the tasks, the participants answered the questions presented to them through questionnaire section. There were few participants who answered all the questions and few others did not answer some of the questions. These data helped us in achieving the other evaluation goal of identifying the usage scenario of the new comprehension technique along with its advantages, disadvantages and uses. The data from each participant are analyzed using thematic analysis which will be discussed in the below sections.

6.1.1 Familiarizing the data

As the first phase of our analysis repeated reading of the collected data was done to get familiar with it so that we could find patterns with ease for later stages. Since the data we obtained is in handwritten form, we transcribed the data into an electronic document for convenience. This way it was easier to find the common patterns and to do further analysis on it. At this stage, we transformed the words or phrases that are incorrectly written in the handwritten answers from the questionnaire. At this stage, we also segregated the data question wise which helped us understand the data even deeper. After the conversion of data into a digital format we analyzed the data and noted some ideas about it.

6.1.2 Generating initial codes

Coding was carried out in this phase. Here the term coding implies the terms or phrases from the collected data that are potentially relevant to the research question are identified and noted. Codes are labels or categories generated from the data we obtained. In other words, coding is a method done in qualitative analysis to break the descriptive data into categories. It was easy for us at this stage to understand the data and do the coding because of the categorization of data question wise. For convenience in this research, coding is done using inductive analysis. Inductive analysis is a way of exploring the information by generating theory from the data available. This enabled us to compress the information into a summary and also helped us to create a link between the research question RQ3 and the data.

From the answers we obtained from the participants we noticed that in some cases there are just one line statements which gave us a single code. Whereas in some cases, one answer was broken into multiple codes. For the question 2, the following is one of the answers we obtained.

Question 2: *Do you think UI-Tracer has limitations?*

If yes: what are the limitations?

Answer: *“It is difficult to compare lot of images”*

For the above answer obtained, the code generated is “Lot of images”. As we are referring to limitations in the question, the phrase “Lot of images” is relevant here. For this reason, this code is chosen under the data collected for question 2. Similarly, coding is done for other three questions in the questionnaire.

In the question 4 of the questionnaire, we added all the resource provided to the participants as options during evaluation. We asked the participants to come up with scenarios when they would be using the technique over other. So, the answers were obtained in the small table as shown 6.1 and this led the participants to provide short answers. Thus, the data collected as answers for question 4 are of two or three words and this in turn made the coding process simpler. For example, consider the following example in 6.1:

Question 4: *Think of different situations you might be in a system that is new to you, e.g. need to fix a bug in change a behaviour, evolve the system, refactor the system, new requirements are coming in ...*

What do you have to understand about the system, and when would you rather use which of the following tools?

Rather SonarQube (Visualization & Analysis)	Rather UI-Tracer tool
<i>Answer: Refactoring</i>	<i>Answer: New to system</i>
Rather Documentation	Rather DoxyGen
<i>Answer: Always out of date</i>	<i>Answer: New to the system</i>

Table 6.1: Sample answer obtained for question 4 from questionnaire

As shown in table 6.1, a participant has mentioned that he/she would be using

SonarQube for refactoring, UI-Tracer and Doxygen when new to a system. The participant has also complained that the documentation technique is always out of date. Similarly, answers from other participants for question 4 are short which gave us the opportunity to proceed to the next step of analysis i.e. “Searching for themes”.

6.1.3 Searching for themes

Once we generated the codes from the data, the next step done was to create or find themes from the codes available. Themes are a subset of codes. From the list of available codes obtained from the previous step, themes are created. This is done by categorizing the codes from the above list under a broader level of themes. Grouping the codes generated from the previous step helped us to obtain condensed and broader view of the data under consideration.

Question 1: *Do you think UITracer provides an advantage?*

If yes what is the advantage

Answer: *“To locate UI elements functionality not sure of other tools that do that ”*
“Functionality with UI, Full documentation of how front-end and back-end are connected”

All data that are relevant or related to a theme is grouped. An answer collected through questionnaire might have one or more theme depending on the code generated. The above two answers are from the data that was collected for the question 1 in questionnaire phase. For the first answer the code “Locate UI elements” was created and for the second answer the codes “Functionality with UI” and “Connection of front-end and back-end” were created. These three codes on a broader view fall under one theme, i.e. the mapping of UI to its code/ functionality. Thus, the theme “UI-mapping” is created and in a similar way themes are generated for the rest of the data.

6.1.4 Reviewing themes

At this point, the generated themes from the previous step were reviewed to ensure the correctness of the analysis. Some themes did not have data to support them. This reviewing is done on two levels. One review is done on code level(which are below the themes). In this step, a review was done to check if the codes under the themes form a pattern with the themes. The other stage is done at the theme level. Here the themes are checked to ensure that they fit in the context under discussion and if not we discarded them or renamed to make the analysis consistent.

Mind maps are generated for easy visualization of the data that was being analyzed. A mind map is a visual representation of data that projects the theory of objects linked to and arranged around a central key word or idea. We used mind map in this step to help us with the review of the themes as it gives an overview of the data. Here the primary branches represent the themes we generated from the qualitative

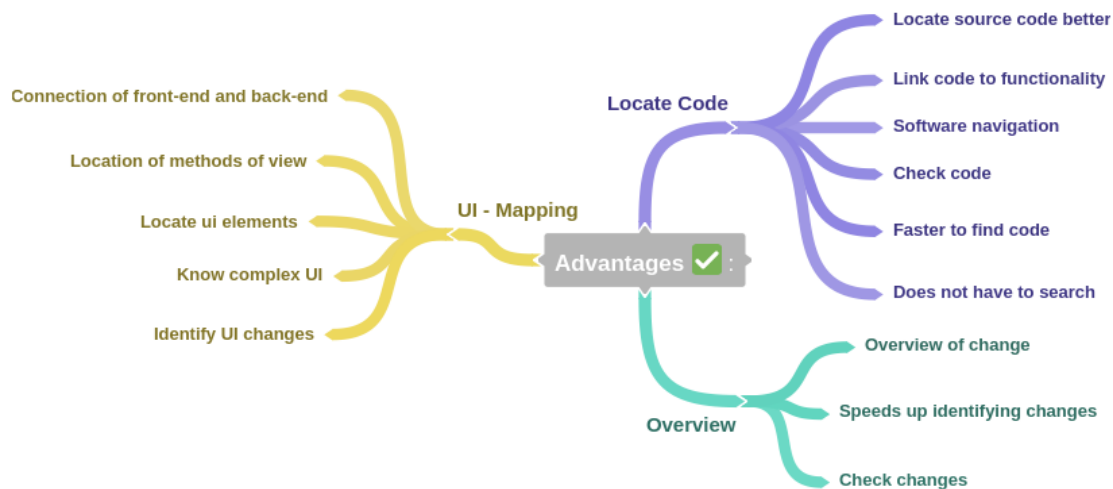


Figure 6.1: Mind map of the central theme ‘Advantages’

data whereas the secondary branches represent the code or label obtained from the data collected. So, for the mind map shown in the figure 6.1, the central idea is ‘Advantages’. This mind map is generated as a part of the analysis performed on the data collected for the question 1.

Question 1: *Do you think UI Tracer provides an advantages?
If yes: what is the advantage?*

The central topic advantages have the themes “UI-Mapping”, “Locate Code” and “Locate Changes”. These are the primary themes of the data collected, and the secondary branches represent the phrases or terms used to group the data during the coding process.

The figure 6.2 represents the mind map of the limitations of the approach. That is the reason the central theme/idea is “Limitations”, and the surrounding branches represent the themes created during this analysis. “Different Projects”, “Misleading”, “Larger Systems” and “Too much documentation” are the themes that represent the data collected for the question 2.

Question 2: *Do you think UI Tracer has limitations?
If yes: what are the limitations?*

The mind map in figure 6.3 represent the data collected for the question 3. The data represents the usages of the results from the approach. Thus “Usage” is the central idea here and the surrounding themes represent the overall data collected.

Question 3: *When would you use UI Tracer & how would you use the information?*

The data obtained from the question 4 contained answers that are mostly single line statements. So, coding is not performed on the data collected for question 4. The data obtained for this question will be discussed in the last section of this analysis. Few themes were removed during this review stage from mind map, and the figures with mind map represent the data after review. The next subsection will cover the description of these themes and what they infer about the data collected.

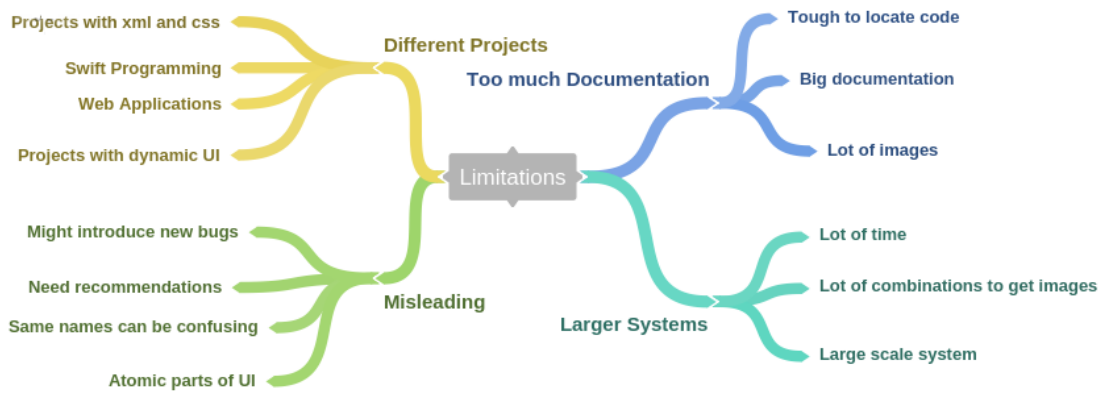


Figure 6.2: Mind map of the central theme ‘Limitations’

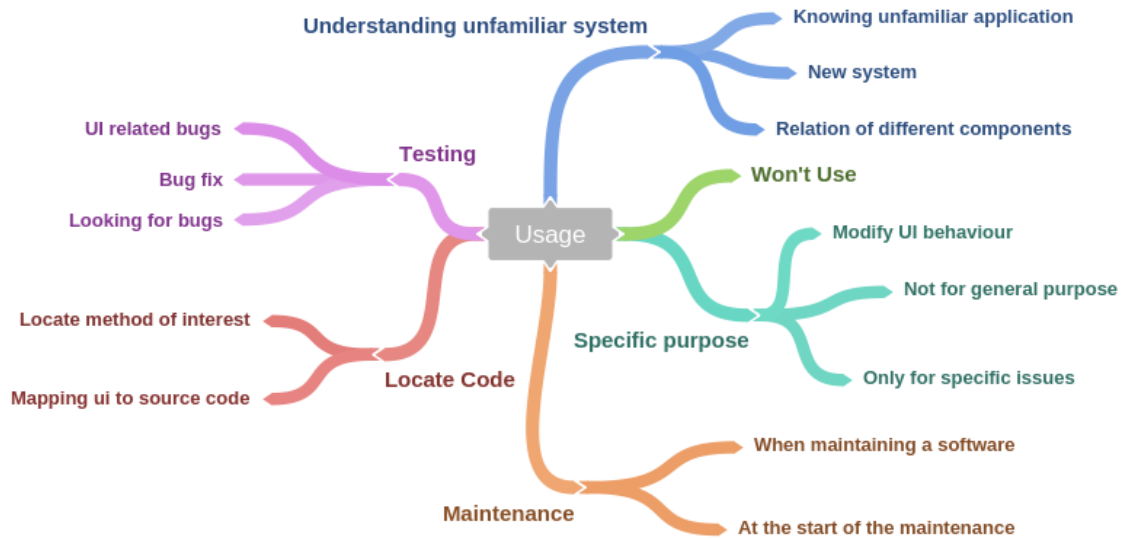


Figure 6.3: Mind map of the central theme ‘Usages’

6.1.5 Defining and naming themes

This step is to explain what the above-mentioned themes from mind-maps imply and how they represent the data collected. This step also describes how the mentioned themes fit in the context of the research relating to the findings that helped us in answering research question RQ3.

6.1.5.1 Advantages

The following are the themes obtained from the data collected for the question 1. Each theme will be discussed and described along with statements from the data.

Locate Code:

This theme represents the data that mentioned that the comprehension technique under study provides the advantage of locating the relevant code the participants are searching. From the codes generated during the initial phase of our analysis this broader theme was created. There were answers obtained from the participants that are linked to this central theme. Participants reported that using this tool, they could locate the code they are looking for easily without having to search for the code.

“To locate UI elements functionality not sure of other tools that do that”

Using this tool they have achieved faster search of the piece of code they are searching. For example, a participant stated the following as an advantage.

“Faster to find relevant code”

Overall this theme abstracts/ represents these statements. Some participants also reported similar data for the answers to question 3 i.e they implied that they would be using this technique for “Locating code”. This led us to use the same theme for the “Usages” as well and hence will not be discussed separately under ‘Usages’ section.

UI-Mapping

Many participants mentioned that the new comprehension technique helped them in comprehension by providing the mapping of the UI components and its source code.

This theme comprises the data which expresses more about the results generated by the tool i.e. the mapping of UI Component and its source code. On the other hand the theme ‘Locate code’ comprises the data which expressed the ease of locating code achieved via using the results from the tool. One participant reported that this technique has the advantage with the mapping of UI components to its source code of know complex UI.

“It allows visual elements on the screen to easily be connected to methods that create them. Reverse engineering to know complex UIs.”

A participant also said that this technique has the advantage of connecting the front-end and back-end. This statement is given below.

“Functionality with UI, Full documentation of how front-end and back-end are connected”

These similar answers led us to generate this central theme. Overall this theme represents the data from the participants that states the advantage of the technique as a mapping between the UI and the code.

Overview

Some participants found the technique to help them with knowing the overview of the system. As we gave the participants the results of the approach, they found them as a good overview to refer when needed. They mentioned them as the advantage

of this technique. One answer also suggested that this technique has the advantage of speeding up identifying changes.

“Quickly getting a overview of what methods interact with the UI”

As these data revolves around the central idea of helping them with finding the changes in the system, we generated this theme “Locate changes”.

6.1.5.2 Limitations

For the question 2 from the questionnaire, we obtained different themes that deals with the limitations in general. These themes will be covered here.

Different Projects

Participants questioned whether this technique would be suited for projects other than the one that was the subject of the study. For example, one participant doubted the support of this approach on swift programming and this statement is given below.

“Swift programming for macOS and iOS”

One of the participants has also mentioned in his/her answer that this approach would not work for web-based applications as quoted below.

“Does it work for web applications”

Another interesting answer suggested that this approach would fail for systems with no UI or systems with dynamic UI which is given below.

“Based on understanding it might be tough to find differences in dynamic behaviour of the UI without manual interaction from humans”

All these data represent the limitations this approach has for projects other than “Arts Of Illusion”. So, the theme “Different Projects” was created as a central theme to group these data under them.

Misleading

Many participants suggested that using this approach and making the changes would lead to the introduction of new bugs. There is also one answer that mentioned about the naming that could confuse. The following are some answers that are grouped under this theme.

“Functions and UIs have same names that cause confusing.”

“We did the task 1 in a dirty way by commenting the code thus it might introduce to new bugs”

In the results provided to the participants, some methods have same or similar names. This similarity in the names led to confusion for the participants to solve the tasks given to them. It is understandable why this comment was given. Another similar answer is for the comment that mentioned that the approach needs recommendations to the right method. What we inferred from this answer is that he/she expects the technique to be accurate to point to the right code or at least to provide some recommendations(hints).

Too much Documentation

From the data collected we observed that many participants have hinted out that too much information in the form of documentation would make the search slower. One of the answers obtained is given below.

“It is difficult to compare lot of images”

There is also an answer that mentions that referred that this approach in Larger systems would result in big documentation thereby making the approach slower. We find this answer interesting as the participants have found a limitation related to the generation of the documentation. This statement is stated below in quotes.

“If there are more methods then that would result in huge amount of images. This would not only be unhealthy but it would also prevent the localization as too much to search through.”

From these statements, we inferred that having too many images in the result would not help the user much.

Larger Systems

An answer obtained from the participant mentioned is that it would require a lot of time to process and generate documentation for systems is large in scale. The following is one of the related statements that is grouped under this theme.

“It doesn’t go through all the parts of the code important to find the most atomic parts of the UI, can be cumbersome on larger projects”

And there is also an answer that mentioned about the time required to generate documentation for systems with complex UI which also stated that testing all the combinations would consume a lot of time. This statement is quoted below.

“UI Traces has to use the whole user interface. UI Traces has to click button, test every possibilities and open every panel. There are too many combinations to test them all.”

We find these answers surprising as the participants considered even the generation of documentation into account when they were answering the questionnaire.

6.1.5.3 Usages

The data collected from question 3 talks about the situations where participants would use this technique. Here we have listed all the themes obtained for this question, and the description of the themes obtained for question 3 is given below.

Testing

Participants stated that they would use the approach when they have to do testing in UI or do bug fix in UI. The following is one of the statements that represent this theme.

“Testing my code , to locate instance of the UI element”

There are also participants who mentioned in their answers that they would use this approach for finding bugs in the UI. This statement is quoted in the below text.

“Finding UI related bugs and change in ui related behaviour”

Similar answers obtained from the data collected were grouped under this theme. We could interpret that participants find this tool to help them during testing systems with UI.

Maintenance

Participants stated that they would use this approach when they are maintaining a system. One participant also mentioned that they would use this approach when they have to use the software at the start of the maintenance as mentioned below.

“To map UI to source code, at the start of maintenance. As the results are sorted by area it is even more easier. ”

Another answer obtained from the data collected that were grouped under this theme is also mentioned below.

“I need to maintain a software, The info will be used to locate the method of interest but as I said previously locating the method reduces the effort when the method is not too big ”

We could observe from the results that certain participants could find using this tool during the maintenance phase of a software system.

Specific Issues

Few answers described that the participants would use the technique for specific issues. However, the context of specific issues is still unknown. It is not safe to assume the topic specific issues to something else. So, we did not transform theme to something else. Rather we kept it as it is and used it to imply the participants answers as, that this technique is not generic and not suited for all situations.

“For specific issues in an application that I’m not familiar with ”

The above-mentioned themes are one of the statements that are grouped under this theme. Another statement obtained is given below:

“Use it whenever I want to do specific stuff in the system or If I wanted to better understand how the different component work or are related to each other. ”

This statement again implies that that participant would use this technique to do something specific in the system. These answers that have a similar context in them are grouped under this theme.

Won’t Use

We obtained few answers that stated that the participants would not be using this technique. There are not many reasons provided here except a participant who stated that he/she rarely develops UI in java as stated below.

Personally don’t use as rarely code UI in java. Will contact previous designers or developers

But we find this theme as important as others and so we specify this theme separately.

Understanding unfamiliar system

Few participants said that they would use the approach to understand the system that they are not familiar. Few others mentioned that they would use this approach when they are new to a system. As all these answers are related to the core idea of understanding an unfamiliar system we generated this theme.

For example, one participant mentioned that:

“When working on code have never touched before”

Also, one participant mentioned that he/she would use this approach to understand an unfamiliar system. This statement is mentioned below in quotes:

To know UI changes in application that I’m not familiar with. Also for comprehending the dependencies of the UI

Related answers from the collected data are classified under this theme.

6.2 Findings from Analysis

The answers are so diverse, and this allowed us to explore more areas of the topic under research. However, to condense all the information collected, we have quantified the findings. This summarizes the data collected, and the themes analyzed to make it easier for interpretation. We have counted the themes mentioned by each participant and presented the information as a summary of analysis in charts.

Summary of Advantages

We have obtained three different themes for the data collected for question 1. Each adds a different value to the research. Here in table 6.2 we have presented the statistics of the number of participants who quoted each theme.

Themes	No of participants who mentioned this theme
Locate Code	12
UI- Mapping	5
Overview	4
No Opinion	2

Table 6.2: Advantages of UI Tracer and the number of participants who mentioned it

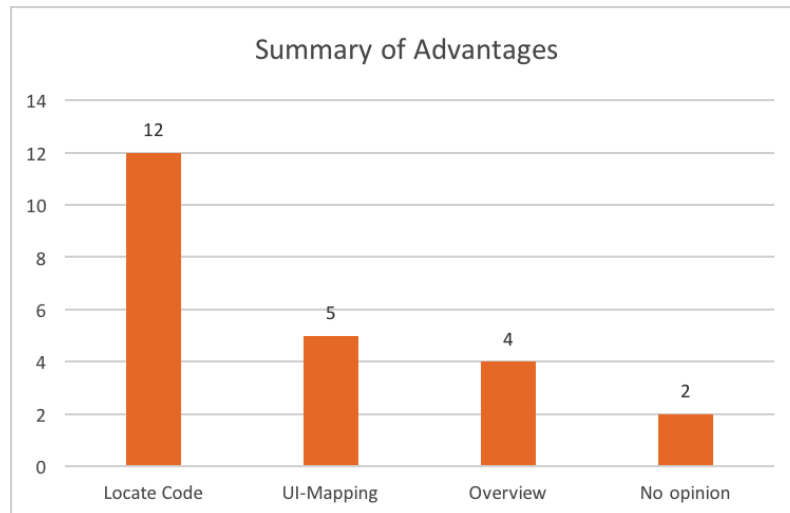


Figure 6.4: Chart of Themes obtained for question 1 and their count

The table 6.2 and graph 6.4 shows that the theme Locate code has been mentioned by 12 number of participants making it the highest among its group. Second to Locate Code comes the UI-Mapping mentioned by 5 participants. Next to this is the Overview theme quoted by 4 participants. It should be noted that out of the 24 participants who took part in the questionnaire section only 2 participants chose 'No opinion' and expressed nothing for Advantages part.

Summary of Limitations

Similarly, from the data obtained from the question 2 the table is obtained. We have obtained four themes under Limitations.

Themes	No of participants who mentioned this theme
Different projects	5
Larger Systems	7
Misleading	4
Too much documentation	3
No opinion	4

Table 6.3: Limitations of UI Tracer

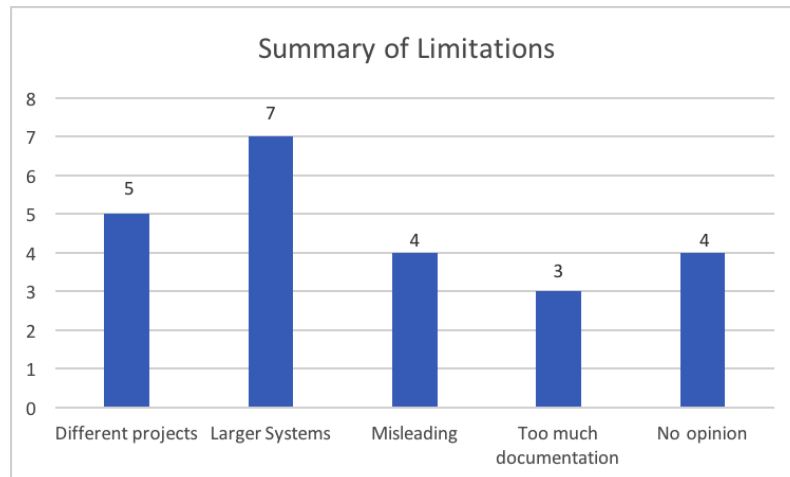


Figure 6.5: Chart of Themes obtained for question 2 and their count

The table 6.3 and chart 6.5 shows the number of participants who mentioned each theme under limitations. It could be observed that the seven participants have mentioned the theme ‘Larger systems’ in their answers. This theme as discussed questions the support of this tool on larger systems and stated that it would be time-consuming to generate documentation. Next to this five participants mentioned the theme ‘Different projects’. Four participants mentioned that it would be misleading to use this approach. Three participants have expressed the theme ‘Too much documentation’ from their answers. Four participants chose ‘No opinion’ as their option for answering question 2 in the questionnaire. Overall, we managed to obtain some interesting and different themes out of the data collected.

Summary of Usages

The table is created using the data from the themes obtained from the analysis of the data collected for question 3.

Themes	No of participants who mentioned this theme
Testing	4
Locate Code	8
Understanding unfamiliar system	11
Wont use	3
Specific purpose	8
Maintenance	3
No opinion	0

Table 6.4: Usage of UI Tracer

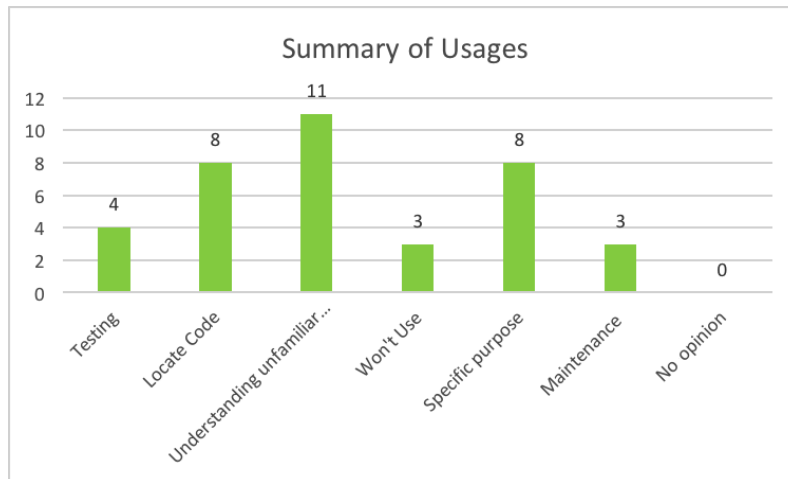


Figure 6.6: Chart of Themes obtained for question 3 and their count

The chart 6.6 provides a visual representation of data given in table 6.4. It could be observed that almost half of the participants have mentioned through their answers that they would use this approach to understand a system that they are unfamiliar with. This is significant information as eleven participants have mentioned this theme. Eight participants have mentioned that they would use this technique for locating the code that they are looking for. Another interesting theme mentioned by the participants is the theme ‘Specific purpose’. As discussed as the term ‘specific’ could not be assumed to be something that the participants did not specify. However, it could be interpreted that eight participants find themselves to use this technique for a specific purpose and not for generic purpose. Eight participants mentioned that they would be using this technique for locating code and four participants mentioned that they would use this while testing UI based systems. Just three participants mentioned via their answers that they would be using this technique during maintenance of the software. Every 24 participants answered the question 3, and none chose the option No opinion. It has to be noted that three participants mentioned that they won’t be using this technique.

Summary of Answers obtained for question 4

For answer 4, unlike all other questions, the participants answered general concepts in one or two words. This, in turn, led us not to follow thematic analysis completely as the data obtained already contained themes. In few cases few participants did not answer few sections. For example, they left blank for “Rather doxygen”.

So we gathered all the information and managed to condense all the data to form the table given below 6.7.

	Rather Sonarqube	Rather UI-Tracer	Rather Documentation	Rather Doxygen
Maintenance	4	3	2	3
Fixing bugs	2	6	4	3
Changing behaviour	0	5	12	4
Understanding an unfamiliar system	2	6	4	6
UI mapping	0	5	0	0
Refactoring	11	3	2	5
Never helpful	2	0	3	4

Figure 6.7: HeatMap of answers obtained for question 4

As shown in table 6.7, the participants have mentioned that they would use Sonarqube for refactoring code compared to other documentation techniques. Whereas 12 participants have mentioned that, they would use text-based documentation for changing the behaviour of the system. Six participants mentioned that they would be using UI-Tracer for fixing bugs in a system whereas four participants mentioned they would use documentation for the same purpose. Few participants stated that Doxygen, Sonarqube and documentation never help them. While the concept of documentation generated by the UI-Tracer tool is rather new to participants, no participant mentioned that the tool is never helpful. Moreover, six participants mentioned that they would use UI-Tracer and Doxygen to understand an unfamiliar system and four participants mentioned they would use documentation for the same purpose. Only two participants mentioned that they would use Sonarqube for this purpose. Five participants mentioned that they would use UI-Tracer to find UI mapping whereas no one mentioned that they would be using other techniques for the same purpose. Interestingly we found that six participants mentioned that they would be using UI-tracer for the understanding of an unfamiliar system which is one more than the number of participants who mentioned that they would be using this technique for locating the code. This finding is also similar to the data we collected were 11 participants reported they would use this technique for the understanding of an unfamiliar system while 8 participants mentioned that they would be using this technique for locating the code. It is clear that despite the standard techniques available for comprehension, this study shows that participants find key areas where they find scenarios to use UI-tracer.

7

Discussion

This chapter presents the threats that concern the validity of the research along with the validity of generalizing the results. This chapter ends with the discussion on answers to the research questions of this thesis.

7.1 Threats to validity

In this section, we discuss the potential threats to validity that affect the soundness of the research. We identified two validity types:

- Internal Validity.
- External validity.

7.1.1 Internal Validity

Internal validity refers to the degree to which the conduct of a study eliminates the systematic error [57]. Multiple factors could cause systematic errors in research, and this includes the choice of population recruited for the study, the factors involved in the measurement of study variables etc [58]. While conducting the questionnaire section for data collection, we wanted to elicit as much of information possible from the participants. So we chose open-ended questions for the questionnaire part and thus avoiding bias that might have occurred as a result of giving suggestions to participants [56]. In addition to this, we also made the data collection through questionnaire anonymous so that participants don't adjust their answers in fear of being judged. Moreover, when conducting the task phase before the data collection, we gave the participants with documentation techniques such as code cities, documentation from 'Arts of Illusion' etc. This way we prevented the participants from solving the tasks in favour of the technique proposed in this study and also helped in eliminating the risk that participants had to attribute positive feedback alone to us. The thematic analysis performed on the collected qualitative data was done inductively. This way of doing inductive analysis links the generated themes emerge directly from the data and thus being the reflection of the data itself.

7.1.2 External Validity

External Validity refers to the degree to which the findings of the study could be generalized to the population [57]. So, the characteristics of the sample obtained

using convenience sampling are analyzed to find the how they represent the population. Though we used convenience sampling for choosing participants, the common characteristic of the selected sample is their connection to software engineering. Two of them in the sample are teaching assistants of the course and rest all are final year Master's students studying Software engineering. The collective knowledge they all have concerning software engineering is about program comprehension, software maintenance, software quality etc. gives us confidence that they know to answer the questionnaire. It is possible that the opinion of more experienced developers differ from the ones that the students expressed. Since the students are already in their final year and have previous experience using software comprehension techniques on an unknown system, we argue that they are qualified to judge the approach.

7.2 Limitations

We already discussed the limitations observed by the participants who took part in the study. However, in this section, we will emphasize and discuss more in detail related to the limitations we observed during the whole research.

The UI tracer tool developed is only intended as a prototype, and so it has few limitations. The main issue with the current version of the prototype is its performance. We observed that the UI-Tracer tool generates results with low performance. Currently, during our test runs, we observed that it takes up to 28 hours to complete its processing on the source code of 'Arts of Illusion' application and generate results. The tool was run on 5348 methods of the 'Arts of Illusion' source code and these many times the tool has to run. So for each method, we observed on an average it takes up to 15 seconds to complete its processing on a method. So, when the source code is large, it heavily impacts the performance of the result generation.

The other issue concerned with the prototype is its support for projects other than Java. Though this issue is mentioned by the participants who took part in the study, we wanted to add this topic here to discuss few things that were not noticed by the participants. We have designed UI-Tracer tool to be adapted to most of the situations in theory. We believe in the generalizability of each component that performs the prime tasks of this research. However, it would help through future researches to discover the challenges incurred in generalizing the tool.

The quality of the results produced by the tool has its limitations as well. This is evident from the number of images obtained from the tool. From the 503 images obtained as a result, we have 393 images that are regarded as invalid. The ratio of valid images to invalid images could be rounded to 1:4. This is a significant difference, and we find this as another limitation that should be addressed with future researches.

7.3 Discussion

RQ1. How can we design and implement the technique to visually observe the impact a method has on the UI?

As shown in the chapter implementation 4, we created a tool that could automate and generate results that could be used in the creation of documentation as proposed. All the main components are flexible to be supported on different platforms like Windows, MacOS etc because of having built the tool on Java. Moreover it is an added advantage that Java with its vast various types of libraries [59] would make it easier while extending the tool with additional features. Currently the tool can only process UI applications based on Java that uses Ant as the build system. As mentioned before the Java library Java Parser is used to retrieve information about the classes and the methods contained in the application that has to be studied. This library is used by the ‘Explorer’ component of the UI-Tracer to parse the contents of the Java file. This ‘Explorer’ component should be adapted for processing UI applications that are built on programming languages other than Java in future researches.

The ‘Comparator’ component performs image processing on what appears on the screen and it uses Sikuli API which could also be used to simulate user actions. Sikuli being a GUI testing tool gives this advantage and we believe that we could use and extend this feature to the tool to simulate basic user actions. Especially in UI systems with multiple screens this feature of the Sikuli could be used to simulate user actions (e.g. clicking a button). For example, in Eclipse IDE, on starting it prompts the user for choosing the workspace.

One of the main components, is the local database added to the tool. This database is used to store all the information related to the mapping of the source code and the images recorded as differences. Simple queries could be used to retrieve data and this could be presented to the user with GUI in the future as a separate tool to add convenience for developers while comprehending. The database could be extended to store additional information about methods and used for different forms of analysis if required.

The tool developed was aimed for research purposes and is still in a prototype version. Therefore, there are few limitations to the tool concerning its performance and usability in general. Theoretically, we observed that the performance concern with the tool could be addressed to some extent with the Sikuli API. Sikuli has a matching algorithm which it uses while comparing images. This matching algorithm in Sikuli has a variable called similarity parameter [60, 61]. The higher the value of this parameter the more robust the algorithm would be, and Sikuli will notice even minute changes between images. The downside associated with this way of using a higher value of similarity parameter is its performance in matching. On the other hand, the lower the value of this parameter would result in improved performance, but the quality of the matching would be compromised. This way of low robust

matching could have been used in our research but it might have an impact on the results of RQ2. So, in this research, we have used a maximum value for the matching algorithm giving importance to the quality of differences generated for RQ2. However, there is no proof recorded as a difference in performance while altering the value of similarity parameter. This should be studied in future research works along with other technical challenges involved. Moreover, in ‘Arts of Illusion’ application, the default build is not associated with any test suite. This saved us a significant amount of time. So, for other applications running the build without running the test suite will save a significant amount of time and prevent build failures associated with test case failures.

As already mentioned the number of invalid recordings observed is a concern related to the prototype. These are caused by run time exceptions that occurred while the application is started which led to the failure in loading UI. As these recordings are related to the failure in loading UI, it would be interesting to see if we can explore the possibility of adding a listener to the screen through Sikuli or other libraries. This way of adding listener will help prevent the tool to record images before the UI has loaded. From the usability point of view, not many efforts were taken during the research as the intention was to create a prototype. Right now, the UI associated with the tool was designed and developed just for study purpose. These limitations have to be addressed in the future researchers by enhancing the quality of the tool which is in its prototype version. Overall, we answer RQ1 with the design and implementation of the tool UI-Tracer. Through the development of UI-Tracer, we prove that the creation of a tool that could help the developers to visually observe the impact a method has on the UI. This way it can help the developers who use UI as a starting point [23].

RQ2. Can we map the UI components of the view that appears after the application is loaded (the “starting screen”) to its source code through this approach?

The prototype recorded 503 images in total as differences. In these 503 images, there are 110 valid images and 393 invalid images. On analysis of the valid images, we found that each component in the starting screen of the UI has gone through a change at least once in the entire processing of the prototype. For example, the UI component ‘Camera object’ recorded the highest number of differences with 25 images. The table 5.3 in results chapter also shows that there are few methods that have created multiple changes by affecting various components of the UI. These changes linked to the UI were examined to see if we can map the components back to the source code. On inspecting the code, we noted that all components of the UI elements could be traced directly to its source code. This is also evident to an extent through the evaluation performed where the participants successfully identified the starting point for comprehension using the new technique.

It is clear from the results that few components are impacted by multiple methods of the source code. Not all methods from the map are responsible for the creation of the UI elements. Some UI component changes in the images are not directly

traced to the methods that created it. For example, we obtained certain methods as results from the UI map that is responsible for fetching the icons of the component. Although the changes are not directly mapped to the source of the UI component, the mapping led us to the part of the source code that is responsible for loading its icons. This way we argue that the results obtained are more than just a UI-map but also could help the developers with the possibility to explore and understand more areas of the source code.

As discussed in section 5.5, we show that all the UI components listed in the ‘Art of illusion’ website [54] has gone through a change at least once and was recorded by the UI-Tracer tool. There are other UI elements which are only visible on a click of a button. We did not focus on these UI elements that do not appear on the starting screen of the application. This is because in this research we focused only on creating a mapping between the UI of the starting screen of the application and its source code. However, from the results observed we could still trace the source code of the UI which is not directly visible on the starting screen. For example, the menu button ‘File; gives a drop-down list on click. Now from the results we have, we could find the source code of the methods responsible for the creation of the ‘File’ button. We believe this could be a good starting point for developers to comprehend even for the UI elements without a direct mapping of its method.

However, it has to be noted that the limitation the tool has concerning the generation of these the valid set of images is significant. It is therefore rather important to understand the challenges and technical difficulties involved in the generation of low number of results. Overall from the results observed and analyzed, we imply that this approach would contribute to the developers who need to comprehend software system through its UI. Through these discussions, we prove that the generated set of results map the UI components to its source code and thereby answering RQ2.

RQ3. How do developers or maintainers perceive this technique?

From the analysis of data collected during the evaluation phase, we obtained answers for RQ3. All 24 participants who took part in the study managed to solve the given tasks using the here presented documentations within the 30 minutes. This time includes the introduction of this new technique to the participants, who were exposed to it for the first time. Thus, the participants managed to use the documentation to successfully find methods that were entry points to solve the tasks within such a short time (the documented example system consists of over 90 000 lines of code and more than 500 classes). After collecting this experience in working with the new documentation technique 20 of the 24 participants felt confident enough to make statements about the advantages and limitations they perceive the technique. All 24 participants provided answers on potential usages they identified. Therefore, we think that participants indeed do understand the concept behind this form of documentation and are quickly able to use it.

During evaluation, we observed that participants begin with identifying code lo-

cations as starting points when they are presented with a task. They looked for options to know the starting point for comprehending. This backs up the findings of the study [23] that identified developers follow a “problem – solution – test” pattern i.e. developers start looking for identifying the code location when presented with a problem”. From what we observed during the evaluation and from the data we collected, it is evident that participants did the same. On analysis of the data we collected from 24 participants, it was found that 12 participants reported that they find locating the code as an advantage that UI tracer provides. In addition to this, 8 participants even mentioned that they would use this technique while locating the code. These numbers show that this technique has the potential to help the developers when they follow the “problem-solution-test” pattern. This gives us the hope that developers would find this tool and its results useful with its some key advantages.

Study [55] shows the importance of understandability and how long developers spent on comprehending software systems during maintenance. Also from [4] we could see the result of issues caused due to lack of understandability of software systems. Interestingly we have found themes from the collected data that concerns these issues. We identified that almost half of the participants in our study mentioned that they would use this approach for understanding an unfamiliar system and on the other hand three participants mentioned maintenance as a usage scenario of this technique. Theoretically, we could establish a relation that through this approach one could locate code or get familiar with an unknown system. The type of usage might differ but they cover important parts of the maintenance challenges, other comprehension challenges such as understanding a complex algorithm, a systems architecture, or dependencies. Thereby we believe that we could address the common maintenance challenge and reduce the time spent on comprehending software systems through this approach.

In this research, no evidence is recorded on the time difference taken by the participant to solve the task with other documentation techniques. However, it is fair to say through our results that this technique has the scope to be used with other techniques that can reduce the common issues with maintenance such as understandability of the system. Thus, we see a demarcation line between program comprehension challenges that are not in range of this proposal, but often in the center of traditional approaches, and challenges that can be addressed with the here presented idea (approaching new systems and locating code). This suggests that traditional comprehension techniques and our idea might function as complements in future.

Are limitations inherent to the idea?

The two top limitations mentioned by the participants are “different projects” (5 participants) and “larger systems” (7 participants). Both limitations are not actually concerned with the presented documentation, but the tool used to generate it. This is interesting since the participants only got a short explanation of what the

tool is doing, but neither saw the tool nor knew about the time it took to generate the documentation. Nonetheless, the participants showed a good feeling about general challenges of such a tool. Indeed, it is necessary to provide some configurations to address other programs, e.g. providing an ant script to compile another system. Even more, there is a language specific aspect to compilation and the manipulation of the source code. However, it is theoretically possible to extend the tool in future to also support other languages.

Likewise, participants doubt that the tool can scale to generate documentation for larger systems within a reasonable time. This is a reasonable concern that we share, although the run-time of around 28 hours for Art of Illusion can be considered somewhat acceptable for a non-optimized prototype that is run over the weekend. Future studies will need to be concerned with optimizing the tool, such that larger systems can be covered as well. The other two limitations concern the documentation itself. Three participants mentioned worries that the documentation gets too big to be helpful, e.g. when locating the right method to implement a change. This is a problem that concerns documentations in general, however, an important one. To eliminate this downside, there is a need for providing navigation and search support in future. Some students perceived the documentation as misleading. This was often due to the fact that different methods can have similar impacts on the user interface, creating confusion about the right entry point. This limitation is to some degree inherent to the presented idea. What the user can see is an interaction of multiple classes and methods in the system, and can be affected by all of them. However, further support might help developers to be more aware of that limitation or even provide options to recommend the method that is the best entry point.

We conclude that most of the limitations mentioned by the participants are connected to the downside of the current state of the prototype that we used for generating the documentation. They do not necessarily hold in general for the underlying idea of enriching documentation with a user perspective.

8

Conclusion

Through our research, we have presented a new technique for program comprehension enrich documentation of source code with aspects of the user perspective of a system. We also came out successful in developing a simple prototype that could generate the documentation for the proposed technique. While our initial prototype is limited to connecting only user interface elements and no behavioral aspects yet, we already gained promising results. The prototype is developed in Java and could be easily modified for an upgrade. We have strong hopes that in future, the idea will help to complement classical approaches that support understanding of architecture or algorithms with support for locating functionality in the source code.

From the already discussed limitations, we can directly retrieve some directions for future research. The most prominent is one is to show that the approach can be extended to systems developed in languages other than Java. Particularly, the adaption to other programming concepts, e.g. functional languages, will be interesting to experiment. Another direction of interest are systems running in a browser, which are more complicated to deploy automatically. Also, performance optimization needs to be addressed in future. Possible directions are incremental methods for creating the documentation as the system is developed.

Another future research direction concerns the presentation and usability of the documentation in general. Few participants noted that it would be a problem with large number of images in the documentation. Finding a better way to navigate in the documentation is an example of the future research direction. A possibility would be to use image processing libraries to automate the finding of changes per UI components of an application. This could be used along with the data in the database and could be presented to the user with an interactive GUI. This way the user can quickly navigate to the code that is responsible for the change. Directions to address the confusion issues as mentioned by the participants can be to provide strategies that developers can use to quickly check what method is the most promising entry point to affect aspects related to a user interface element. Moreover, the tool can be further enhanced by using additional information from the database to provide

An even more important direction for future research becomes obvious with the observation that eight participants stated that they would use this approach only for specific purposes. Part of these comments is the observation that the current version does merely give an insight into user interface elements. Thus, the question

is whether this approach will stay limited to this part of the user perspective or whether future extensions could help to reveal more about the functionality of a system from the user perspective. We see this potential, since many rich options of tools like Sikuli, e.g. filling in forms and processing data automatically, have not been explored yet. This feature of simulating user actions with Sikuli could also be used in the future to discover if we can generate the mapping of UI components that only change when interacting with them. Also, the current version is designed to generate documentation for the UI elements from the starting screen. It would be interesting to see how we can extend the current tool to support multiple screens as Sikuli can be used to simulate user actions. We see this as one of the most important directions for future.

Bibliography

- [1] McKee, J. R. (1984, July). Maintenance as a function of design. In Proceedings of the July 9-12, 1984, national computer conference and exposition (pp. 187-193). ACM.
- [2] Postema, M., Miller, J., & Dick, M. (2001). Including practical software evolution in software engineering education. In Software Engineering Education and Training, 2001. Proceedings. 14th Conference on (pp. 127-135). IEEE.
- [3] Kaur, U., & Singh, G. (2015). A Review on Software Maintenance Issues and How to Reduce Maintenance Efforts. International Journal of Computer Applications, 118(1).
- [4] Uchida, S., & Shima, K. (2005). An experiment of evaluating software understandability. Journal of Systemic, Cybernetics and Informatics, 2, 7-11.
- [5] Souza, S. C. B. D., Anquetil, N., & Oliveira, K. M. D. (2006). Which documentation for software maintenance? Journal of the Brazilian Computer Society, 12(3), 31-44.
- [6] Poulin, J. S. (1994, November). Measuring software reusability. In Software Reuse: Advances in Software Reusability, 1994. Proceedings., Third International Conference on (pp. 126-138). IEEE.
- [7] Storey, M. A. (2005, May). Theories, methods and tools in program comprehension: Past, present and future. In Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on (pp. 181-191). IEEE.
- [8] Śliwerski, J., Zimmermann, T., & Zeller, A. (2005, May). When do changes induce fixes?. In ACM sigsoft software engineering notes (Vol. 30, No. 4, pp. 1-5). ACM.
- [9] Śliwerski, J., Zimmermann, T., & Zeller, A. (2005, May). When do changes induce fixes?. In ACM sigsoft software engineering notes (Vol. 30, No. 4, pp. 1-5). ACM.
- [10] Borjesson, E., & Feldt, R. (2012, April). Automated system testing using visual gui testing tools: A comparative study in industry. In Software Testing,

- Verification and Validation (ICST), 2012 IEEE Fifth International Conference on (pp. 350-359). IEEE.
- [11] Yeh, T., Chang, T. H., & Miller, R. C. (2009, October). Sikuli: using GUI screenshots for search and automation. In Proceedings of the 22nd annual ACM symposium on User interface software and technology (pp. 183-192). ACM.
- [12] Sikuli (Version X-1.0rc3) [Computer software]. Retrieved from <http://www.sikuli.org>
- [13] Java Annotation, Retrieved from <https://docs.oracle.com/javase/tutorial/java/annotations/>
- [14] Javadoc Tool, Retrieved from <http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html>
- [15] Doxygen [Computer Software]. Retrieved from <http://www.stack.nl/~dimitri/doxygen/>
- [16] Graphviz [Computer Software]. Retrieved from <http://www.graphviz.org/>
- [17] Urquiza-Fuentes, J., & Velázquez-Iturbide, J. A. (2004, July). A survey of program visualizations for the functional paradigm. In Proc. 3rd Program Visualization Workshop (pp. 2-9).
- [18] H. A. Müller, S. R. Tilley, and K. Wong, "Understanding software systems using reverse engineering technology perspectives from the rigi project," in Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, October 24-28, 1993, Toronto, Ontario, Canada, 2 Volumes, 1993, pp. 217-226.
- [19] Rajlich, V., & Wilde, N. (2002). The role of concepts in program comprehension. In Program Comprehension, 2002. Proceedings. 10th International Workshop on (pp. 271-278). IEEE.
- [20] Herbsleb, J.D. & Moitra, D. Global Software Development. IEEE Software, March/April, 2001, pp. 16-20.
- [21] Gračanin, D., Matković, K., & Eltoweissy, M. (2005). Software visualization. Innovations in Systems and Software Engineering, 1(2), 221-230.
- [22] Harman, M., & Hierons, R. (2001). An overview of program slicing. Software Focus, 2(3), 85-92.
- [23] Roehm, T., Tiarks, R., Koschke, R., & Maalej, W. (2012, June). How do professional developers comprehend software?. In Proceedings of the 34th

- International Conference on Software Engineering (pp. 255-265). IEEE Press.
- [24] Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (1993). How to design and evaluate research in education (Vol. 7). New York: McGraw-Hill.
 - [25] Buse, R. P., & Weimer, W. R. (2010). Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4), 546-558.
 - [26] Alégroth, E., Gao, Z., Oliveira, R., & Memon, A. (2015, April). Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study. In *Software Testing, Verification and Validation (ICST)*, 2015 IEEE 8th International Conference on (pp. 1-10). IEEE.
 - [27] Vaishnavi, V., & Kuechler, W. (2004). Design research in information systems.
 - [28] Boehm, B. Software engineering. *IEEE Transactions on Computers*, C-25(12):1226-1241, 1976.
 - [29] Python Software Foundation. Python Language Reference, version 2.7. Available at <http://www.python.org>
 - [30] Java. Java Language Reference, version 1.8. Available at <http://www.java.com>
 - [31] Ant. Apache (Version 1.9.9) [Software Management & Comprehension tool]. Available from <https://ant.apache.org/>
 - [32] Maven. Apache (Version 3.5.0) [Software Management & Comprehension tool]. Available from <https://maven.apache.org/>
 - [33] SQLite. (Version 3.19.3) [Database]. Available from <https://www.sqlite.org/>
 - [34] Drever, E. (1995). Using Semi-Structured Interviews in Small-Scale Research. A Teacher's Guide.
 - [35] Jorgensen, D. L. (1989). Participant observation. John Wiley & Sons, Inc..
 - [36] Hevner, A., & Chatterjee, S. (2010). Design science research frameworks. *Design research in information systems*, 23-31.
 - [37] JUnit. (Version: 4.12) [Unit Testing framework]. Available from www.junit.org
 - [38] Bruneau, M., Durupt, A., Roucoules, L., Pernot, J. P., & Rowson, H. (2014). METHODOLOGY OF REVERSE ENGINEERING FOR LARGE ASSEMBLIES PRODUCTS FROM HETEROGENEOUS DATA.

- [39] Sphinx. (Version: 1.6.3) [Documentation Generator]. Available from <http://www.sphinx-doc.org>
- [40] JArchitect. (Version: 2017.1) [Computer Software]. Available from <http://www.jarchitect.com/>
- [41] SourceTrail. (Version: 2017.2) [Computer Software]. Available from <http://www.sourcetrail.com>
- [42] JIVE. [Computer Software]. Available from <http://www.cse.buffalo.edu/jive>
- [43] Alégroth, E. (2015). Visual GUI Testing: Automating High-level Software Testing in Industrial Practice. Chalmers University of Technology.
- [44] Alegroth, E., Nass, M., & Olsson, H. H. (2013, March). JAutomate: A tool for system-and acceptance-test automation. In Software testing, verification and validation (icst), 2013 ieee sixth international conference on (pp. 439-446). IEEE.
- [45] Kaur, H., & Gupta, G. (2013). Comparative study of automated testing tools: Selenium, quick test professional and testcomplete. International Journal of Engineering Research and Applications, 3(5), 1739-43.
- [46] Etikan, I., Musa, S. A., & Alkassim, R. S. (2016). Comparison of convenience sampling and purposive sampling. American Journal of Theoretical and Applied Statistics, 5(1), 1-4.
- [47] Bricki, N., & Green, J. (2007). A guide to using qualitative research methodology.
- [48] Leech, B. L. (2002). Asking questions: techniques for semistructured interviews. PS: Political Science & Politics, 35(4), 665-668.
- [49] Eclipse, I. D. E. (2007). Eclipse Foundation.
- [50] IntelliJ, I. D. E. A. (2011). the most intelligent Java IDE. JetBrains[online].[cit. 2016-02-23]. Dostupné z: <https://www.jetbrains.com/idea/chooseYourEdition>.
- [51] Boyatzis, R. E. (1998). Transforming qualitative information: Thematic analysis and code development. sage.
- [52] Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. Qualitative research in psychology, 3(2), 77-101.

- [53] Garrett, J. J. (2010). Elements of user experience, the: user-centered design for the web and beyond. Pearson Education.
- [54] Eastman, P. (2013). Arts Of Illusion documentation. Retrieved from <http://www.artofillusion.org/docs/AoI%20Manual/contents.html>
- [55] Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., & Li, S. (2017). Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*.
- [56] Slovenija, R., & v Sloveniji, T. (2003). DEVELOPMENTS IN APPLIED STATISTICS.
- [57] Kitchenham, B. (2004). Procedures for performing systematic reviews. Keele, UK, Keele University, 33(2004), 1-26.
- [58] Alexander, L., Lopes, B., Ricchetti-Masterson, K., & Yeatts, K. (2015). Sources of systematic error or bias: information bias. *Epidemiol Res Inf Center (ERIC) Notebook*, 2, 1-5.
- [59] Ye, Y., Yamamoto, Y., Nakakoji, K., Nishinaka, Y., & Asada, M. (2007, September). Searching the library and asking the peers: learning to use Java APIs on demand. In *Proceedings of the 5th international symposium on Principles and practice of programming in Java* (pp. 41-50). ACM.
- [60] Sikuli Documentation, Global Functions[MinTargeSize], Retrieved from :<http://doc.sikuli.org/globals.html#min-target-size>
- [61] Sikuli Java API Documentation, Retrieved from :<http://doc.sikuli.org/pattern.html>
- [62] Rubin, J., & Chisnell, D. (2008). Handbook of usability testing: how to plan, design, and conduct effective tests. John Wiley & Sons.
- [63] McDaniel, S., & Snyder, L. (2004). Planning Usability Tests For Maximum Impact. In *ANNUAL CONFERENCE-SOCIETY FOR TECHNICAL COMMUNICATION* (Vol. 51, pp. 345-349). UNKNOWN.
- [64] Stetler, C. B., Legro, M. W., Wallace, C. M., Bowman, C., Guihan, M., Hagedorn, H., ... & Smith, J. L. (2006). The role of formative evaluation in implementation research and the QUERI experience. *Journal of General Internal Medicine*, 21(S2).
- [65] Techsmith, M. (2011). Usability Testing for Software and Websites.

- [66] Downey, L. L. (2007). Group usability testing: Evolution in usability techniques. *Journal of Usability Studies*, 2(3), 133-144.
- [67] Persson, S. (2004). *Qualitative Methods in Software Engineering* (Unpublished master's thesis). Lund University. Retrieved from http://fileadmin.cs.lth.se/serg/old-serg-dok/docs-masterthesis/59_Rep.5520.Persson.pdf
- [68] Hulley, S. B., Cummings, S. R., Browner, W. S., Grady, D. G., & Newman, T. B. (2013). *Designing clinical research*. Lippincott Williams & Wilkins.
- [69] Thoma, A., McKnight, L., McKay, P., & Haines, T. (2008). Forming the research question. *Clinics in plastic surgery*, 35(2), 189-193.
- [70] Patton, M. Q. (1990). *Qualitative evaluation and research methods*. SAGE Publications, inc.
- [71] Developers, J. (2008). Jython implementation of the high-level, dynamic, object-oriented language Python written in 100 pure Java. Technical report, www.jython.org, accessed 2008.