UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group Software Engineering

Master Thesis Proposal

Submitted to the Software Engineering Research Group
in Partial Fullfilment of the Requirements for the Degree of

Master of Science

Integration of Multiple Static Analysis Tools in a Single Interface

by
Sandeep Varma Ganaraju

Thesis Supervisors:
Prof. Dr. Eric Bodden
Dr.-Ing. Ben Hermann

Paderborn, April 20, 2019

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift

Abstract. Static analysis plays a major role in software development to find bugs and any vulnerabilities in code. There are different analysis tools available in the market. However, it is found out in different surveys about why the tools are not as efficient as expected by software developers. In recent research, it is found out that in a typical software development organisation, they use multiple tools including legacy tools used in nightly builds as an example. This Thesis aims to address the scenario where a developer works with different tools and how adaptive could be the user interface. The novel ideas including approaches adapted from different software engineering disciplines are evaluated through usability design cycle. The usability aspect of the proposed ideas is considered during the evaluation phase. The target users for this evaluation are experienced software developers which ensures the applicability of this Thesis work.

Keywords: Static Analysis, Usability, Wireframe, User Experience Design

Contents

1 Introduction

The effectiveness of Software Development relies on bug free coding. In our day to day progress in coding leads to complexity of software which brings a broader scope for bugs and vulnerabilities that could be introduced easily. The presence of bugs impacts major loss to an extent of $1.1 Trillion in 2016. [36] There are many static analysis tools available in market to address these primary issues. However in latest surveys by Maria et al. [8] and by Johnson et al. [23] it is noticed that software developers are not quite happy with effectiveness and usability of static analysis tools.

In general, a software development organisation used to use a single tool in the beginning in their SDLC (Software Development Life Cycle) process. Later on, when different static analysis tools came into market having reputation for different capabilities on findings of bugs, as an example are emerged then organisations considered to add multiple tools into their development cycle. The other reason could also be some tools are free and open source which made management team to simply add for greater advantage. The advantages could be reducing false positives by recognising a bug reported by different tools, maximise the possibility of detection of bugs etc. This lead to scenario of using multiple static analysis tools for a single software project.

In scenario where an organisation decides to use different tools it leads to disruptive workflow of development process. This brings new challenge on how to make theses tools integrate to the existing SDLC in less disruptive way by improving the respective user interface in terms of usability. This opens a new opportunity / challenge which requires Research and thereby this Thesis aims to address it.

1.1 Problem Statement

1.1.1 How to Integrate the Results of Multiple Static Analysis Tools in a Unified User Interface?

The overall main aim of the Thesis is about, " *How to integrate the results of multiple static analysis tools in a unified user interface?* ". This question was broken down into different Research Questions during Literature Review / Preparatory phase. The three important Research Questions are selected with respect to scope and time limits of the Thesis work.

Research Question 1:
How to display results of the same codebase from different analysis tools?
Research Question 2: What feedback works to know that the bug fixing is on-going?
Research Question 3: How to carry traceability of bug fixing?

The research questions are explained in detail at Motivation chapter 4. To answer each research question, the user interface is designed with novel ideas and also by researching into the different software engineering disciplines tackling a similar issue. The developed prototype with the ideas brainstormed during research is evaluated with software developers. As part of the evaluation, the usability [42] of the user interface is assessed and therefore new usability problems could be noticed which requires to be addressed in the next following iteration of the User Experience Design cycle [22] which is the essence of Human Centered Design [31]. The problems gathered in an evaluation are considered as requirements for new design and the process repeats. This leads to multiple iterations of the User Experience Design cycle. This approach is followed for all the three research questions. The primary contribution of the Thesis is to make sure the ideas evaluated are valid.

1.2 Outline

This Introduction chapter mentioned what the Thesis is all about. The remaining part of the Proposal is structured as follows:

Chapter 2 explains the key concepts such as 'Static Analysis', 'User Experience Design' and 'Wireframe' which are necessary to understand the work of this Thesis.
Chapter 4 discusses what the current research findings and the need of doing the Thesis.
Chapter 5 overviews the goals of the Thesis work.
Chapter 6 explains the way the challenges addressed or attempts to answer the Research Questions using User Experience Design cycle.
Chapter 7 shows how the user interface designs are evaluated with the solution ideas for the challenges or answers to Research Questions.
Chapter 8 shows the time plan of accomplishing the Thesis work.

2 Background

This chapter discusses the key concepts that are required to understand the Thesis work.

2.1 Static Analysis

In the last few years, we see the enormous increase in the usage of Software. We see the presence of Software everywhere in the walk of our life with the Internet of Things, for example. As the usage increases, quality is stressed as to be more secure and it does not break. Earlier, Software developers used to do manual auditing of the code but sooner they realised it is time-consuming and so planned to automate the process. Therefore, different testing mechanisms evolved like black-box testing, white-box testing etc.

Static Analysis falls under the category White-box testing. It tests the Software without executing the code and therefore the name means it. On the other side, there are tools testing with a mechanism by executing the code which is called Dynamic testing. Static Analysis is also called as Source Code Analysis. The uses of static analysis tools are compiler optimization, coding support and detection of security vulnerabilities and bugs etc. [11]. It reports bugs such as Injections, Cross Site Scripting (XSS), Buffer Overflow, and Dead Code etc. [34] There are different techniques followed for analysing Source Code. One example as Data Flow Analysis, it tests the source code by dividing into basic blocks [1]. Here is an example; a php program as seen below is divided into blocks and each block is considered as one node.

```
$a = 0;
$b = 1;

if ($a == $b)
{ # start of block
echo "a and b are the same";
} # end of block
else
{ # start of block
echo "a and b are different";
} # end of block
```

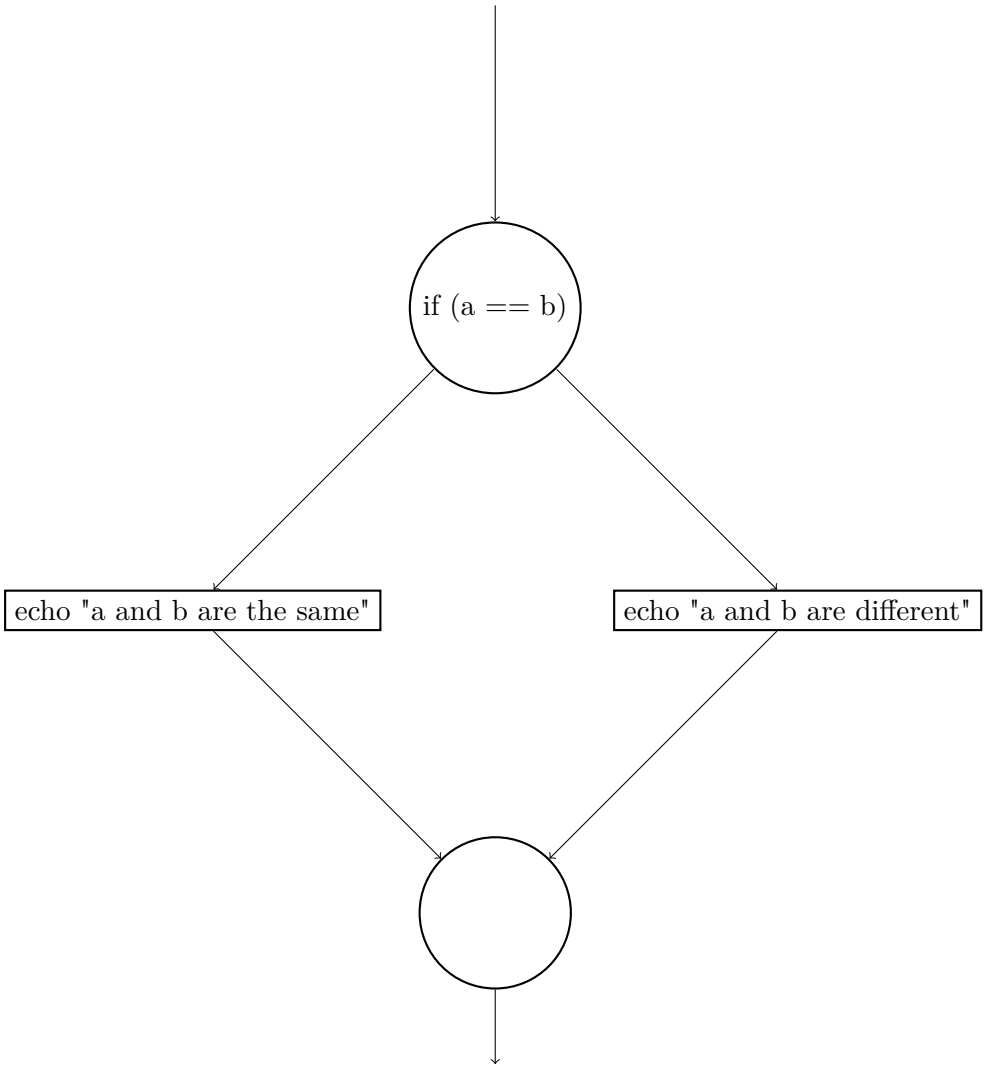

Figure 2.1: Control Flow Graph

Thereafter, a path is formed by connecting the nodes along the control flow as seen in the graph. 2.1

There are different kinds of tools available for doing static code analysis such as IDE Notifications, IDE tools, Dedicated tools, Linters and CLI tools. Out of which, the Dedicated tools and CLI tools are more likely to report security vulnerabilities, whereas others are more likely to report coding style issues. In this thesis, we will look into the usability aspect of such tools.

2.2 Deduction Research Methodology

In scientific research, there are fundamentally two kinds of methodologies followed which are called Inductive inference and Deductive inference. In Deduction research methodology, by first a hypothesis is made based on the existing knowledge or domain experience of the person who made a certain hypothesis. Then observations are made in order to validate the hypothesis. This can be illustrated as seen in the figure 2.2.

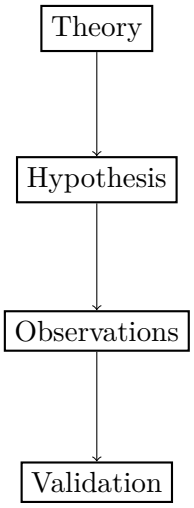

Figure 2.2: Deduction Approach

Example: A person who has knowledge of the working of torchlight knew a battery is needed for its working. When a situation comes, where the torchlight is not working. The person could immediately assume a hypothesis i.e., the batteries are discharged. Then the old batteries are replaced with newly charged batteries, by which observations are made. Finally, the hypothesis is validated based on the observations. If the torchlight works then the assumed hypothesis is true or else invalid and thereby theory/knowledge improved, next change of hypothesis is needed.

In this thesis, deductive research methodology is primarily used as framework and adapts as per 'User Experience Design' cycle which is introduced in 2.5.

2.3 Usability

The Human Computer Interaction researcher, Dr. Jakob Nielsen defines Usability as "a quality attribute that assesses how easy user interfaces are to use" [41]. It is characterised by five quality elements:

1. **Learnability:** It says whether the design is simple for the person seeing the design for the first time and he can understand it.
2. **Efficiency:** After getting familiar with design, how simple it is to do the tasks in a shorter time as possible?
3. **Memorability:** Once the person knew the design and worked for some time. Later, when he comes back after some significant time then is the design helpful enough to memorise?
4. **Errors:** It signifies the possibility of a person encountering the design do some errors and then whether he could recover from it easily or not.
5. **Satisfaction:** It determines the delightfulness of the person using the design.

2.4 Wireframe

A wireframe is a methodology of showing a blueprint of a certain product. For example, if we consider to Wireframe a User Interface of an Application. It means to show a blueprint design of how the Application UI look like with different elements like buttons, text boxes etc placed on it and also how they interact and navigate to other User Interfaces. The blueprint design could vary from low fidelity i.e., rough sketches to high fidelity i.e., a more closer look to the desired final UI. There are several tools available in the market to do Wireframe and Balsamiq [2] is one such tool. It is a kind of prototyping the actual product with certain simulation in functionality and visualization of the actual product. The advantage of this is the easiness of finding the mistakes in design earlier and so the cost of fixing them would be less. Here is an example of Wireframe done to a website idea about how it should look and the elements I would like to see as on a end product i.e., final front-end design of the website coded is shown in the following figure 2.3 of what one could design using Balsamiq tool and discuss with peers how they feel or to oneself to draw the ideas of what they want to achieve before actually coding to get the web page with the desired UI.

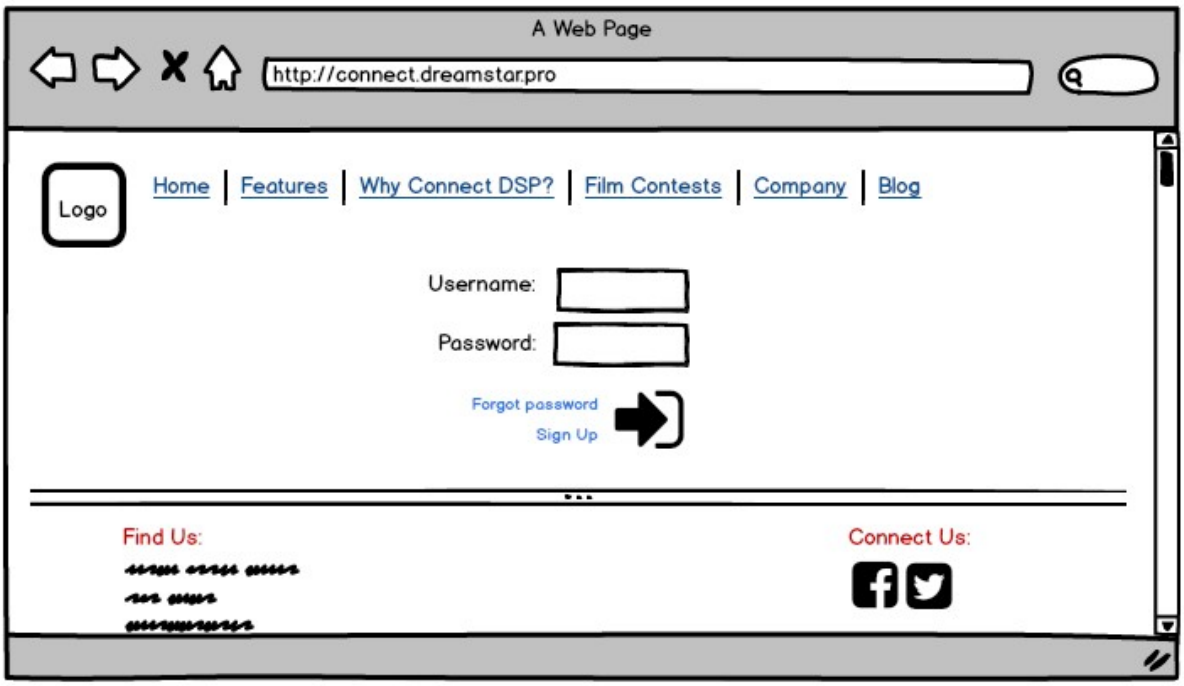

Figure 2.3: An Example to Wireframe a Website.[2]

2.5 User Experience Design

User Experience (UX) Design plays a vital role in the success of a product but often underestimated. In a typical Software Industry, they say reasons to like it might lead to over budget or no time in order to skip it and get right into the development phase. Well, a User Experience Design process is far beyond what we know as User Interface Design or Usability. User Interface Design is what typically done by Wireframe tool and Usability is a way of testing whether the

designed product is usable enough or not. A User Experience (UX) Design is a user-centered process which emphasis on the context of the user and his needs than rather focusing solely on interface design. [39] For example, let us say you designed a navigation application where the user says I want to reach from point A to point B and it displays the route. Now, what if the user wants to mention points as a market name instead of street name and your underlying database consists only street names, that is a bad user experience (UX) even though the application is designed with better UI and also Usable. Its Design [22] cycle as seen in the figure 2.4 illustrates an iterative cycle where the requirements are gathered, then a prototype is made on it and evaluated, then again new requirements are gathered based on the evaluation.

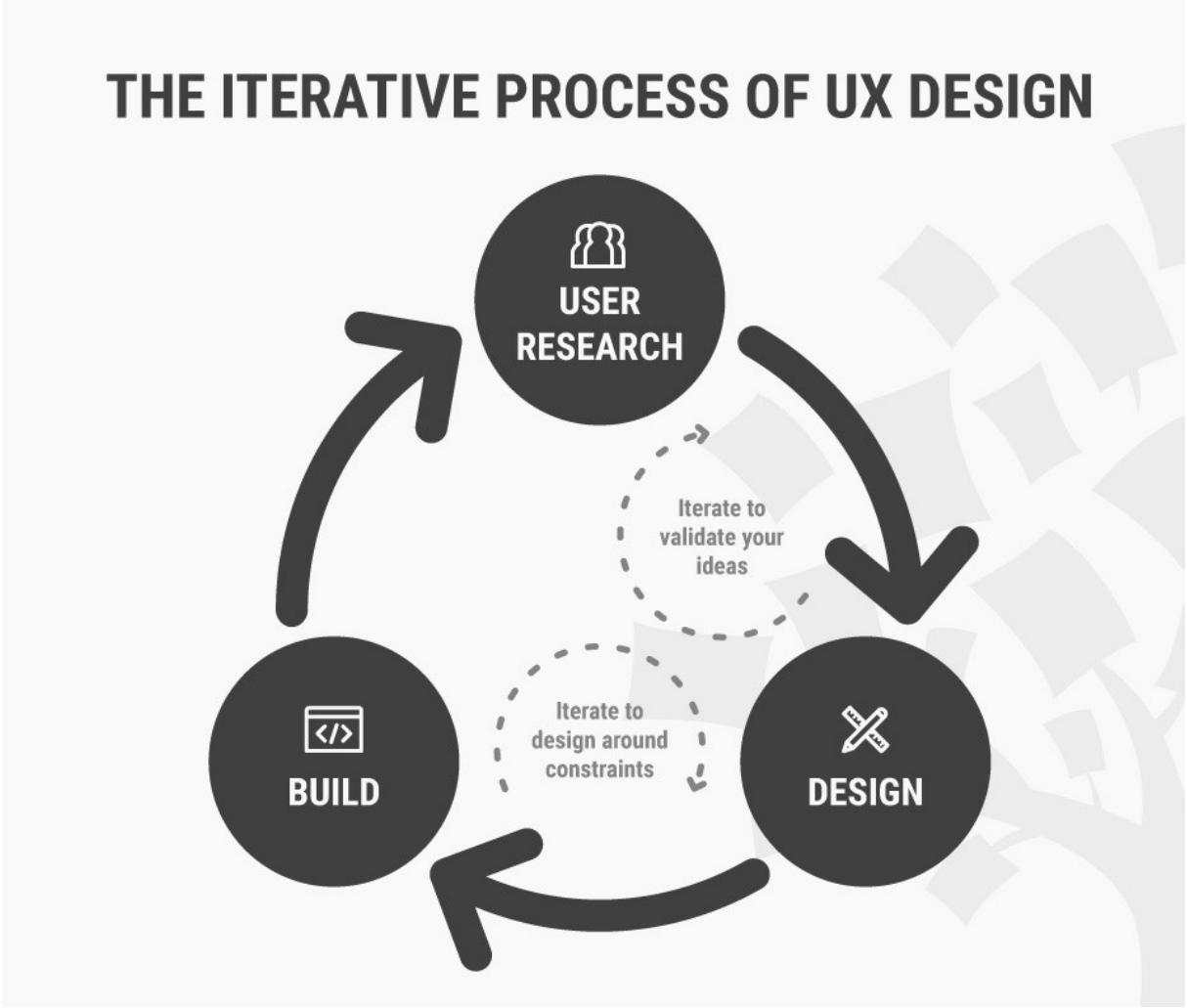

Figure 2.4: UX-Design.[22]

The applicability of the User Experience Design can be seen in detail at Chapter 6.

3

Related Work

Static Analysis plays a prominent role in releasing bug-free Software. In spite of that, important tools suffer from well-documented usability issues [8, 23]. Maria et. al. [8] did an empirical study on what developers want and need from Program Analysis. They noticed that there are some obstacles which hinder the usage of a Static Analysis tool by a developer such as 'Wrong checks are on by default', 'Too many false positives', 'Too slow', 'Complex user interface' etc. Being a user interface an obstacle for a developer is noteworthy. Johnson et. al. [23] also found design flaws in current Static Analysis tools and the need for an interactive mechanism in assisting developers in fixing bugs. The interesting findings are like if the output of static analysis tool is user friendly and intuitive then false positives and high number of warnings could be less problematic for a developer, showing call hierarchies with which parts of code are affected by a bug, be able to share settings with predefined coding standards among the team, need of a web browser for reacting on the analysis output for instance adding comment to a bug which goes out of context to the developer. The key takes away from the above-mentioned papers is the importance of Usability in the ongoing adaption of Static Analysis tools.

In general, the setup of most of the recent research [8] [23] done in the area of Static Code Analysis is like assuming a single project in an organisation. Further, they assume there is a single person working on a single project with a single tool tackling a single type of problems. Somehow, the assumptions are made so singular to address a specific issue in their research. However, in practice i.e., in the real world of software engineering, there are numerous people working in teams for multiple projects at a time. Each project uses multiple tools in their software development. Even in the case of Static Code Analysis, multiple tools are used which are each capable of addressing several types of issues.

4 Motivation

4.1 Current Research

In the current research, we could see the usage of multiple tools in the Software Industry where each tool is different in computation approaches as one could be standard run on Nightly builds for example with Checkmarx [7] tool or could be following Incremental Analysis, which means only testing the changed version of code instead of running analysis on complete codebase. The reasons of using multiple tools could be as each tool is capable of detecting bugs with different coverage [4] [10], it could help in finding more bugs easily which are not found by tool but the other [32]. There is also research [16] going in the direction of using multiple static analysis tools in order to prioritise the bug warning alerts. There is a paper [25] published in 2008 which uses results of three different static analysis tools for a programming language, Java and merges them together in order to show warnings to the developer. Also, recently a team from OASIS [28] in the OASIS SARIF Technical Committee [29] introduced a Static Analysis Results Interchange Format (SARIF) of representing bug warnings in a JSON format. Currently [30], the specification is being standardised. This shows there is potential research going in direction of using multiple static analysis tools at once. This shows the potential of using multiple tools in near future once the standardisation is adapted across static analysis tools developers or at least with the availability of adapters which are capable of converting the output of one static analysis tool to a SARIF format. Here's an example [35] of how SARIF might look. SARIF is been used by SWAMP [37], a continuous software assurance which is automated.

```
1
2  "results": [
3  {
4  "ruleId": "GSV001",
5  "message": {
6  "text": "Variable \"i\" was used without being initialized.",
7  "richText": "Variable ‘i‘ was used without being initialized."
8  },
9  "locations": [
10 {
11 "physicalLocation": {
```

```
12 "uri": "file://build.example.com/work/src/collections/list.cpp",
      "region": {
13 "startLine": 16
14 }
15 },
16 "fullyQualifiedLogicalName": "collections::list:add"
17 }
18 ]
```

Although we have seen the current research trends and their direction, the usability aspect is not addressed so far in the context of using multiple tools. There are no research papers found addressing the usability issue with multiple tools in single interface. The multiplicity of tools used for a single project and that too with different computation capabilities brings a new challenge. One tool could produce results in no time and others could take more time in comparison. This could lead to a situation of breaking the usability of the user interface. On that challenge, this thesis aims to address such a scenario with a problem statement as " How to integrate the results of multiple static analysis tools in a unified user interface? ".

The thesis work begins by brainstorming on the problem statement and as a result, different research questions are brought up and of which three research questions are selected with respect to available sources, scope and limitations of Thesis time frame.

4.2 What Current Tools do?

Let us now examine each research question and see what the current tools behave like.

1. How to display results for the same codebase from different analysis tools?

This question needs to address the scalability aspect of displaying results as usage of multiple tools results in redundancy and a high number of warnings with more coverage. The following figure 4.1 shows how a FindBugs [14] static analysis tool shows results for a project [15]. This could be the same with other tool and so when you use multiple tools, there is a need for a better user interface.

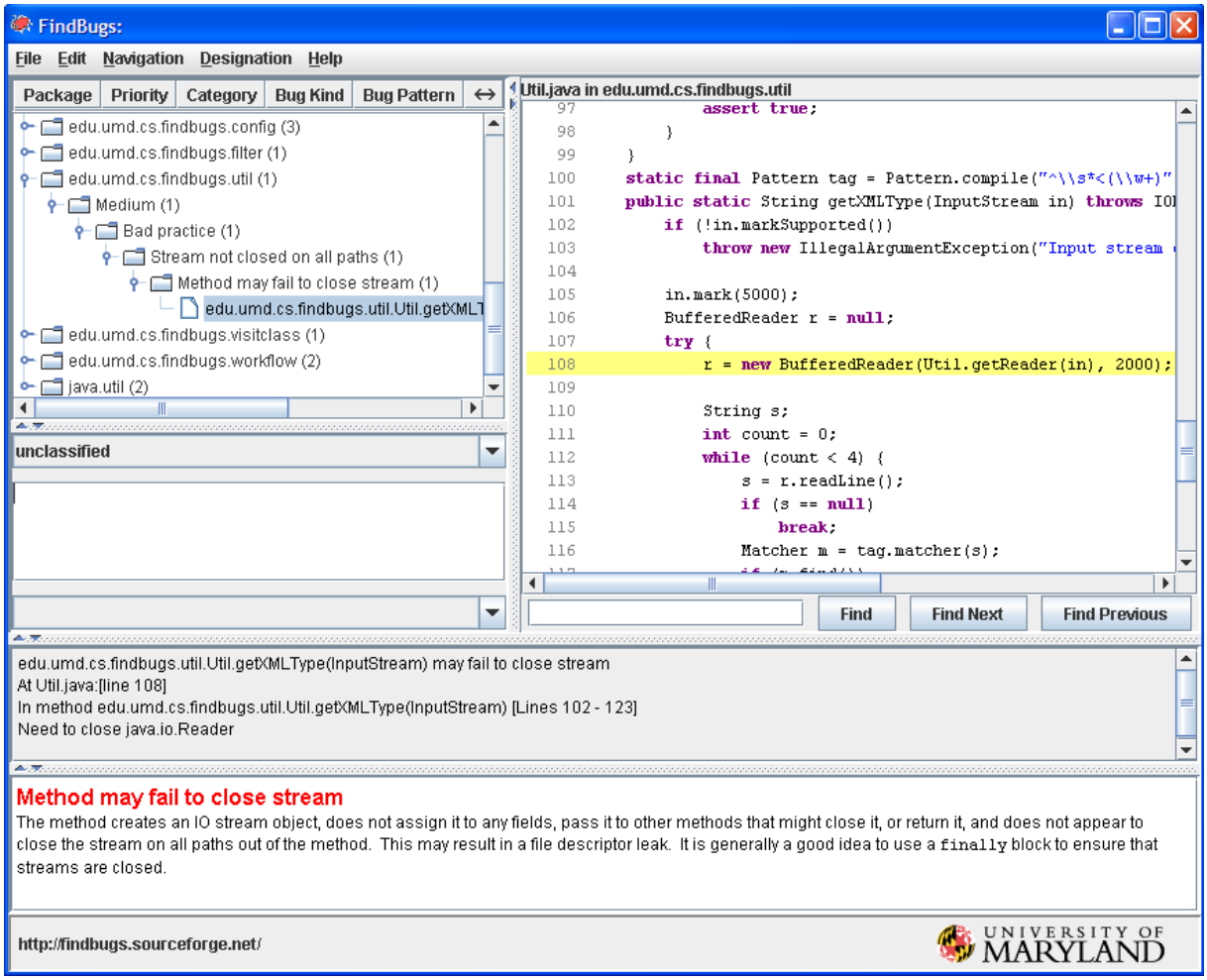

Figure 4.1: A preview of FindBugs scan results.

2. What feedback works to know that the bug fixing is on-going?

This question needs to address the scenario where one tool could give an instant update on the bug fixing process and others might take more time to analyse and report the update on it. In the design aspect, especially the User Interface needs to be adaptive [26] enough as static analysis tools sometimes take a long time to stop and there is no intuitive feedback provided. Also, One interesting aspect of Johnson et. al. [23] study is about the importance of feedback from tools without disrupting the developer workflow. Traditionally, the project files are added to FindBugs [14] tool as seen in the figure 4.2 in order to start the scanning and the user has to wait for some minutes to see the results. There was no feedback in the process of scanning.

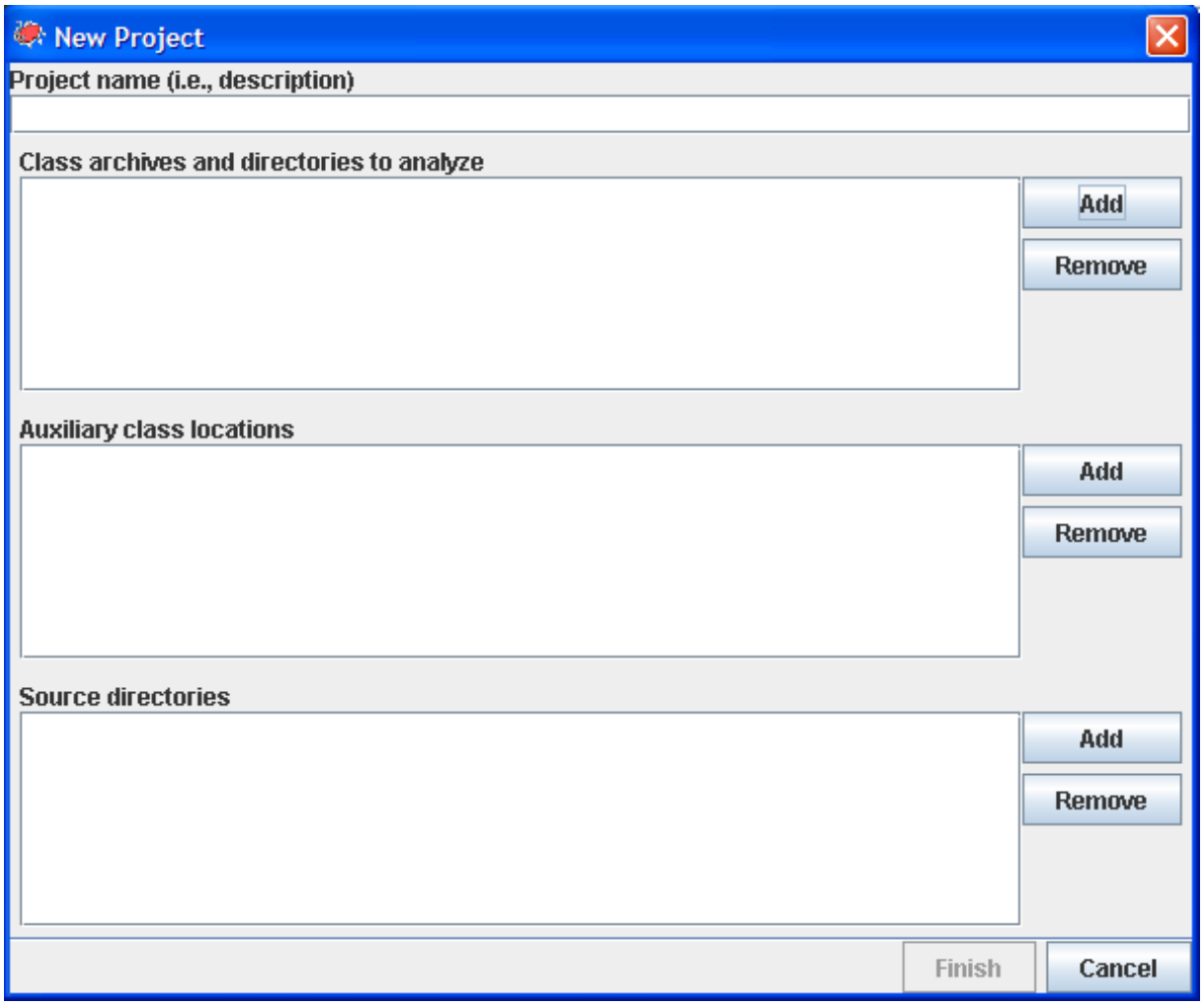

Figure 4.2: A preview of initiating FindBugs to scan a project.

Back in the history of Computer Science, there is an observation made when the user interface is non-responsive, the user shuts down the system. Thereafter, comes to the implementation of user interface called Ghost screen by Colleran et al. [9] who patented the method which manages application programs with non-responsive user interfaces in the year 2005. About the response times, NN Group [33] states that if the execution of a certain task takes 0.1 up to 1 second then there is no need for feedback, just show the result. If it takes 10 seconds, there

should be feedback and if it is variable every time then there is the importance of per cent bar [6] but sometimes it could be overkill to use as it cause stress to the user by the principle of display inertia. If the time taken by a task is unknown then there has to be feedback like a spinning ball or in an example of a task being to scan databases then it has to report user what database is being scanned currently. Overall, there has to be feedback stating that the system is working, if not indicating what is actually doing. This motivates to know how responsiveness in terms of usability is vital to consider in the development of modern tools. Further, this needs to be addressed in the context of using multiple tools.

3. How to carry traceability of bug fixing?

In the scenario, where the user has picked a bug to fix and worked on it and later he submitted for analysis. Then the bug could either get fixed or new bugs could have been introduced or different bugs got resolved by fixing the one bug. All these might have taken place and this is uncertain. Thereby it would be better to have traceability in order to somehow safeguard the code repository from future bugs or monitor the changes happening in the context of bugs.

As per the related work in research [20], a tool named "Teamscale" [38] shows the influence on the quality status for each change in code backed by version control commits. For a change, it shows how many quality problems are added or removed. The illustration could be observed in the following figure 4.3.

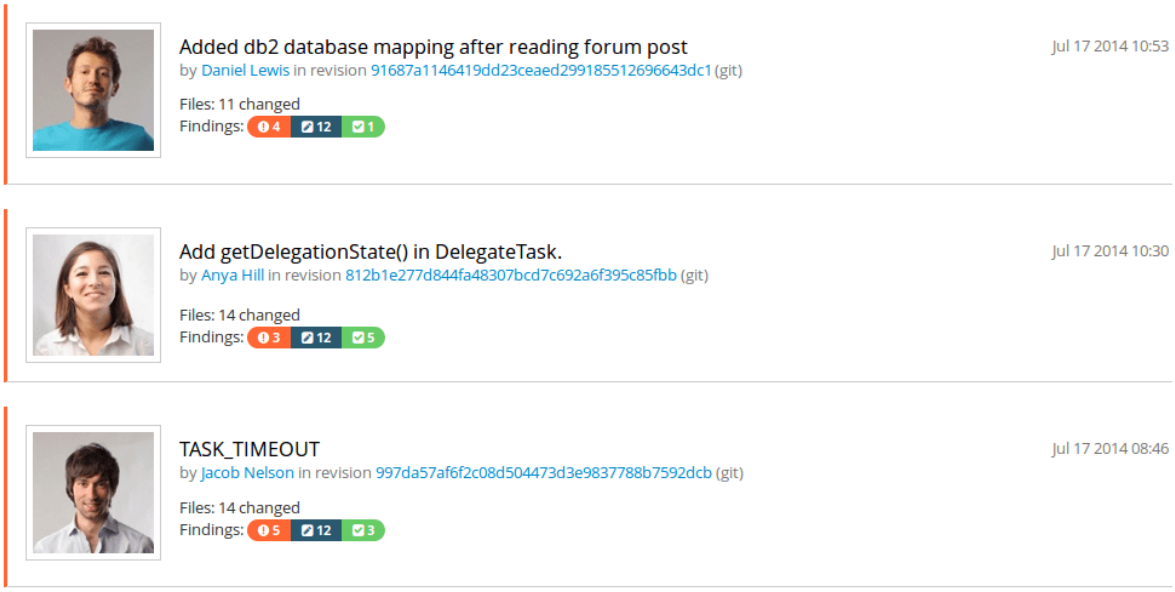

Figure 4.3: A Teamscale feature of showing quality status with code changes.

5 Objectives

Static analysis tools suffer from well-documented usability issues [8, 23]. In the current research, it is found out that a typical software organisation uses multiple tools. This leads to the issue where the user interface has to be adaptive in integrating such different tools.

In this thesis, it is researched on different user interface designs that allow code developers to navigate around this issue in a non-disruptive way.

1. Research different techniques that tackle the respective research question in other domains of software engineering.

2. Adapt those techniques and design own techniques for the domain of static analysis.

3. Design prototypes with a wireframe tool [2] of those techniques to improve the usability of integrating analysis tools.

4. Design user studies that evaluate the efficiency of those techniques, with professional code developers.

5. Run the user studies and report on their results.

6. Loop 2 $\rightarrow$ 5

6 Approaches

The Thesis work follows the Deduction research approach as mentioned in chapter 2 where the theory is established based on own ideas in addition to literature review findings. In order to tackle the research questions, different disciplines of software engineering such as Complex datasets, Compiler reporting, Continuous integration, Refactoring tools, Issue tracker, Stack Overflow, Gamification, Usability Engineering etc. are looked into and studied what ideas can be adapted into our scenario along with own novel solution ideas.

After the literature review in above-mentioned disciplines, here are a few important take-aways in the scope of this Thesis. In the area of 'Complex datasets', current research where Dix et. al. [12] talks about more complex grouping and linking of datasets in the context of a user interface of Spreadsheets application. There could be two datasets with fields having similar meaning and fields which are completely different. So, the key takeaway is about design lessons of extensibility of columns for example, 'venues were geocoded to allow spatial graphs' could be related as an example dates in bug reports to some standard format for all tools used and shown on a unified interface. Next, Gaur et. al. [18] speaks about the linear search problem in indexing as it takes more time for large volumes of data. So, different parameters are introduced to decrease computation time. Example: a database with toys is searched linearly for a given query it takes more time than a modified query let's say a toy in red colour and type horse then the search is simplified by looking at two parameters i.e, colour and type. This sparks the idea of grouping bugs as per module, bug type which could ease user in finding a certain bug on an interface.

In the area of 'Compiler reporting', Horning et. al. [21] mentions the importance of error logging with statistics as to what the compilers are expected to tell the user. It also mentions, the importance of stating what kind of bugs are not found along with bugs found but in reality this questions the scalability. So, the key takeaway is it is ideal to show the number of certain bugs founds in an analysis. Next in the area of 'Refactoring tools', Dustinca [13] talks about how the Refactoring tools are to be built and in user context, it has to overcome the barrier of discoverability which means the difficulty of use. To assist the developer on this issue, they introduced a smart tag in the context of user editor and notifies which parts of the code can be refactored. This emphasizes the importance of 'on-board' phase which plays a key role in Gamification [17] discipline. Hayashi et. al. [19] illustrates the importance of task level commits

in order to maintain edit history of refactorings. This gives an idea of which a user does a bug-fix level commit to addressing the traceability scenario. Mealy et. al. [24] mentions about the importance of usability for software refactoring tools and this could perhaps give some basic guidelines similar to knowing Usability Engineering [42] discipline.

In the area of 'Issue tracker', Baysal et. al. [3] mentions reducing the information overload for a developer in using the issue tracker. It is found out in their there is a too much of information they receive which in fact confuses the developer in how to react, example: the developer receive a high number of bugs reported via email and this leads to a situation where the developer ignore the email. They found out some interesting solution ideas such as having a private dashboard for each developer as it becomes easy to react to issues correspond to them. Expressiveness is one other mentioned in their paper which says an example, severity or priority are vague terms to describe a bug. Perhaps, it is ideal to describe the priory as per team decision instead of personal choice. This signifies in categorising the results as per categories in our unified interface. Next in 'Stack Overflow', in a research paper by Wang et. al. [43] it is found there are 10934198 questions on a 'User Interface' topic for example. It is quite challenging to go through such a high volume database but nevertheless, the Stack Overflow team has a friendly user interface as shown in the following figure 6.1. It uses some clean filter techniques like tags for each topic, priority and trending etc. A research by Treude et. al. [40] found out that most of the questions (72.30%) in Stack Overflow have between 2 and 4 tags. This could perhaps ease in filtering/indexing issues.

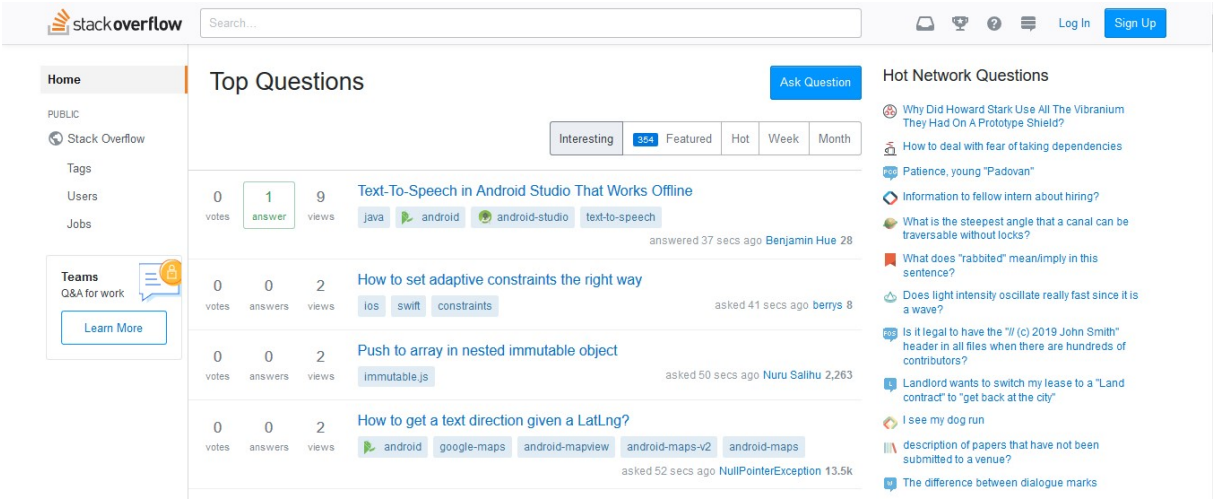

Figure 6.1: An user interface of Stack Overflow Website.

Overall, by analysing the software engineering disciplines as mentioned above, a theory is established. The prototypes are created based on the established theory and user study mentioned in Evaluation chapter 7 is performed which follows Qualitative and Quantitative approaches in assimilating the results. It follows a well established Iterative process of UX Design [39] cycle as seen in the figure 2.4.

Our approach leads an iterative process where initially prototypes with our novel ideas are

evaluated by the target users. Next, the evaluation results lead to the requirements gathering phase. Then again, the prototypes are developed based on user feedback and also including any new findings in the literature, and so on the cycle repeats until the desired satisfaction of target users is achieved.

In our Qualitative research methodology, the feedback of users is concerned as an example of emotional feeling on the usability of the designed prototype. Example: user behaviour, quotes etc. On the other hand, in our Quantitative research methodology, we analyse some metrics on the results gathered during the evaluation phase. Example: time taken to perform the task, performance etc.

For every Research Question, the user scenario is formulated and what usual Static Code Analysis tool does. Next, what can be done better considering other solution ideas from different Software Engineering domains in addition to our own ideas is analysed through the UX Design process. In this process, the metrics mentioned above which are Qualitative and Quantitative are observed.

Here is an example of how the implementation approach could get started.

6.1 Research Question 1: How to Display Results from the Same Codebase from Different Analysis Tools?

Solution ideas:

1. Display results separately for each tool
2. Combine the results and mark its respective icon in a column to indicate which tool identified the certain bug.

With above-mentioned possible solution ideas, two different prototypes are designed using a wireframe tool called Balsamiq [2]. Assume the name *toolShort* mean a Static Analysis tool capable of giving results in a short time and toolLong mean a Static Analysis tool gives results after a long time.

Prototype 1:

The prototype for solution idea i.e., displaying the results separately is shown in following figure 6.2.

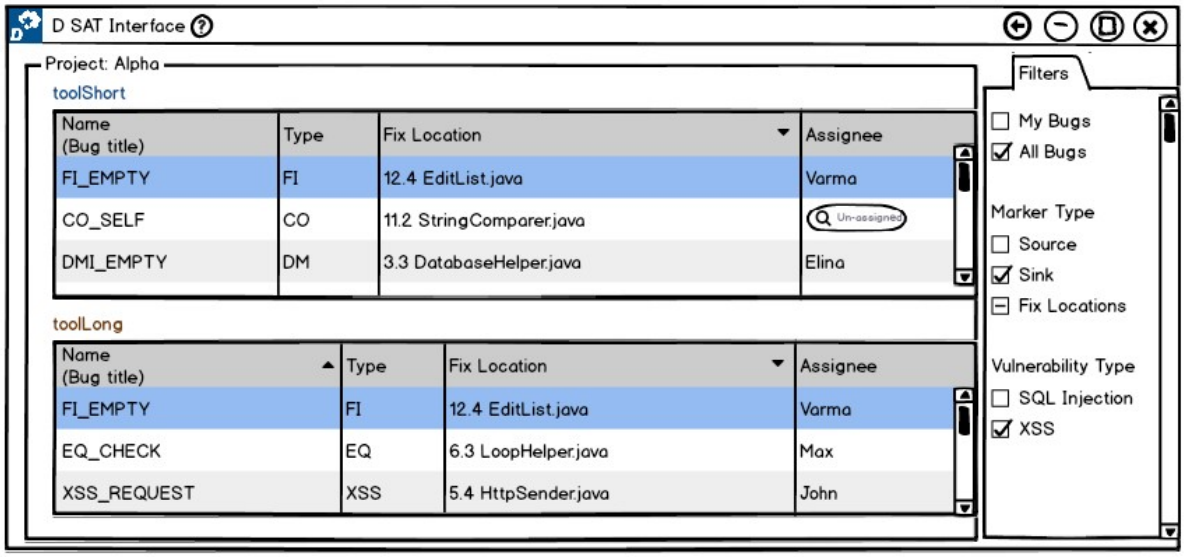

Figure 6.2: An interface prototype with tools displaying results separately.

Prototype 2:

The prototype for solution idea i.e., combining the results is shown in following figure 6.3.

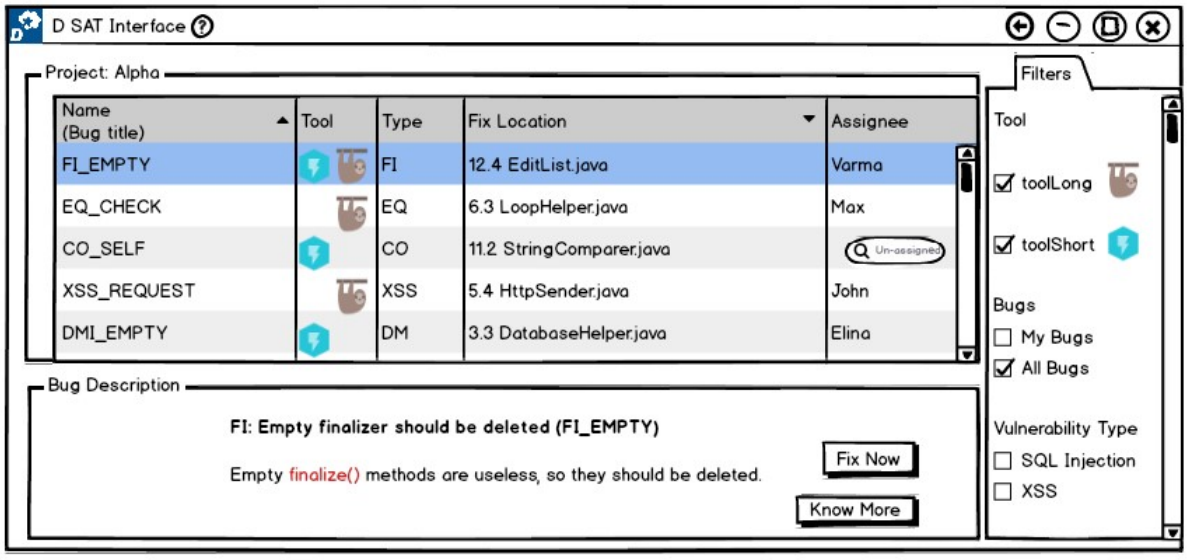

Figure 6.3: An interface prototype with tools displaying results combined.

Next, once the prototypes are designed, they are evaluated as mentioned in chapter 7. After evaluation, requirements are gathered based on user feedback and also new findings from the literature, if any. New designs are prototyped as mentioned above and repeats as per User Experience Design cycle.

6.2 Research Question 2: What Feedback Works to Know that the Bug Fixing is On-going?

Solution ideas:

1. Once the user attempts to fix a bug and submit for analysis, then the bug is shown as pending in status.
2. If the bug is fixed then it is shown as 'fixed' in status, or else, 'try again'.

With the above mentioned possible solution ideas, the prototypes are designed. The first prototype 6.4 illustrates a bug being displayed as 'pending' in the status column of bug listing. This happens when a user selects a bug and attempts to fix it then submit for analysis tools. Perhaps the shorter tool i.e., the tool capable of analysing in less computation time would report back whether the bug is fixed or not. Thereby, the Prototype 2 6.5 illustrates the bug is not fixed and shown as 'try again' in status column as the user attempted to fix earlier.

Also, in Prototype 1 it is observed that 'Fix Now' button in 'Bug Description' window is disabled which also depicts the bug is being analysed in the background.

Prototype 1

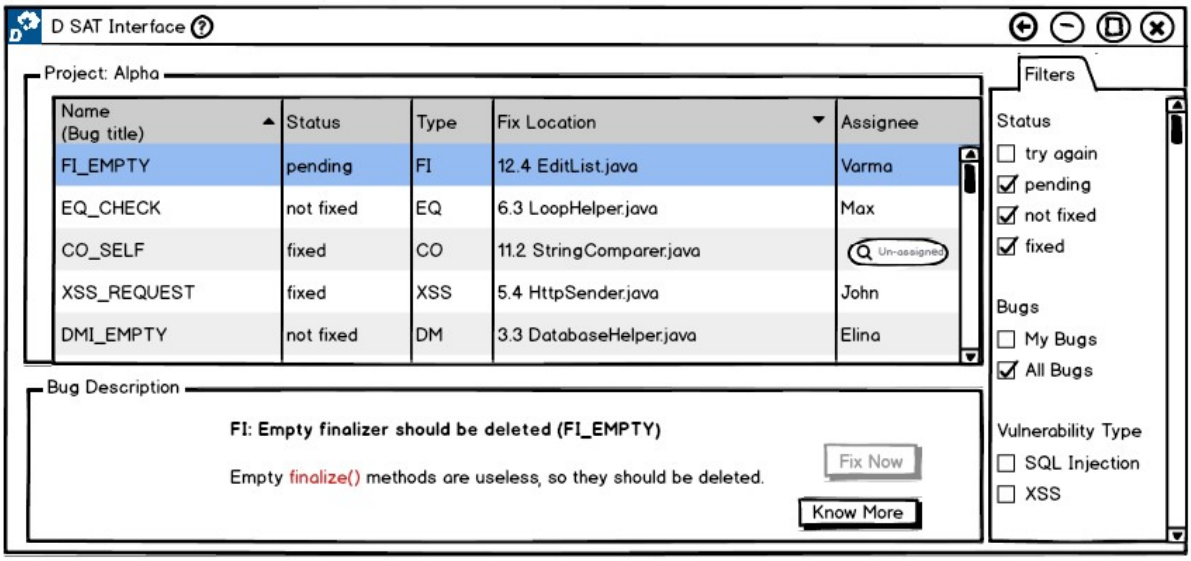

Figure 6.4: An interface prototype showing pending status.

Prototype 2

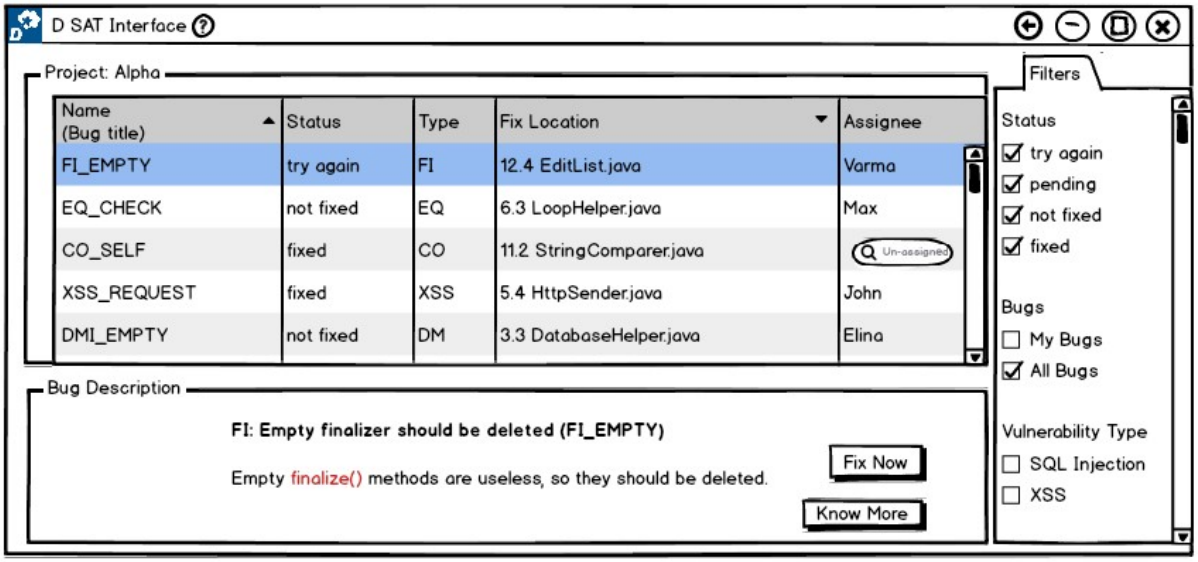

Figure 6.5: An interface prototype showing 'try again' status.

Next, once the prototypes are designed, they are evaluated as mentioned in chapter 7. After evaluation, requirements are gathered based on user feedback and also new findings from the literature, if any. New designs are prototyped as mentioned above and repeats as per User Experience Design cycle.

6.3 Research Question 3: How to Carry Traceability of Bug Fixing?

Solution ideas:

1. a time stamp for each bug fixing attempt with changes button
2. a Revert button

With the above mentioned possible ideas, the prototypes are designed. The first prototype i.e., Prototype 1 6.6 illustrates there is a time stamp for each bug fixing attempt which might help in the context of traceability as to know when someone trying to mitigate a bug. Also, a button 'Changes' could perhaps show the code difference to the previous state of the codebase. The other prototype i.e., Prototype 2 6.7 illustrates in a situation where the user wants to revert back to the previous situation of the codebase. This could perhaps help in a situation when new bugs are introduced with an attempt to fix a certain bug.

Next, once the prototypes are designed, they are evaluated as mentioned in chapter 7. After evaluation, requirements are gathered based on user feedback and also new findings from the literature, if any. New designs are prototyped as mentioned above and repeats as per User

Experience Design cycle.

Prototype 1

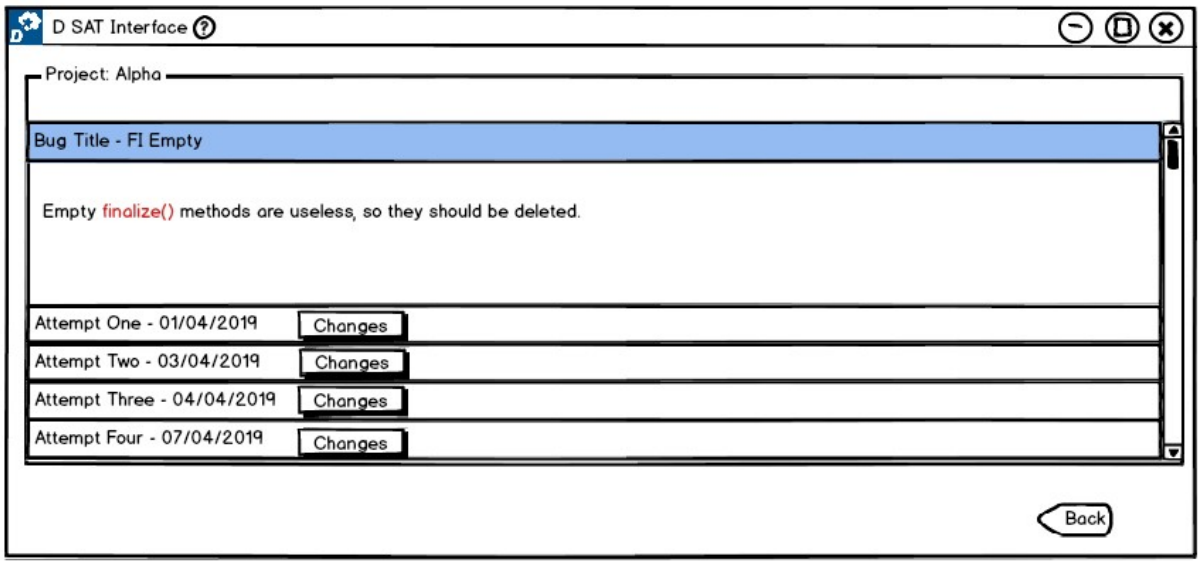

Figure 6.6: An interface prototype showing time stamp and changes button.

Prototype 2

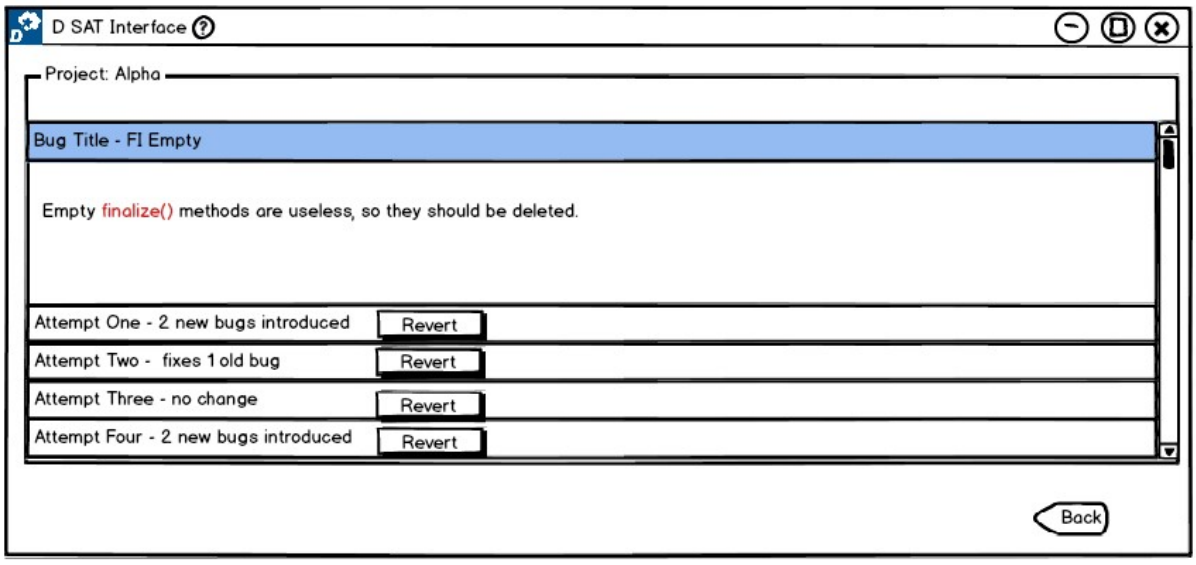

Figure 6.7: An interface prototype showing revert button.

7 Evaluation Plan

The prototypes designed with approaches mentioned in the earlier chapter are evaluated as per the below experiment design guidelines.

7.1 Experiment Design

7.1.1 Number of Test Users

The target users for evaluation are experienced software developers who have good knowledge of software development in general and used one of static analysis tool in their development process. So, if not the professional software developers at least the students pursuing a Masters degree in Computer Science. This ensures the evaluation process to be valid and authentic. There will be five users selected for evaluating the prototypes. One might be surprised about why only five users required, the reason behind that is well explained by Human-Computer Interaction researcher Dr. Jakob Nielsen with a simple formula[44].

$$N(1-(1-L)^n)$$

Where **N** is the total number of usability problems exist in the given design, **L** is the proportion of usability problems discovered while testing a single user which is typically 31% as found in his research. The below plot 7.1 further illustrates that after the number of users is five then the usability problems discovered does not increase much further as there would be high overlap with already found usability problems by previous users. Thereby, five users are selected for each iteration of the User Experience Design cycle to test the prototype designs.

7.1.2 Order of Evaluation

As the order of prototypes with different solution ideas presented in evaluation could influence the user as they tend to learn. Therefore, the order is changed for different segments of users

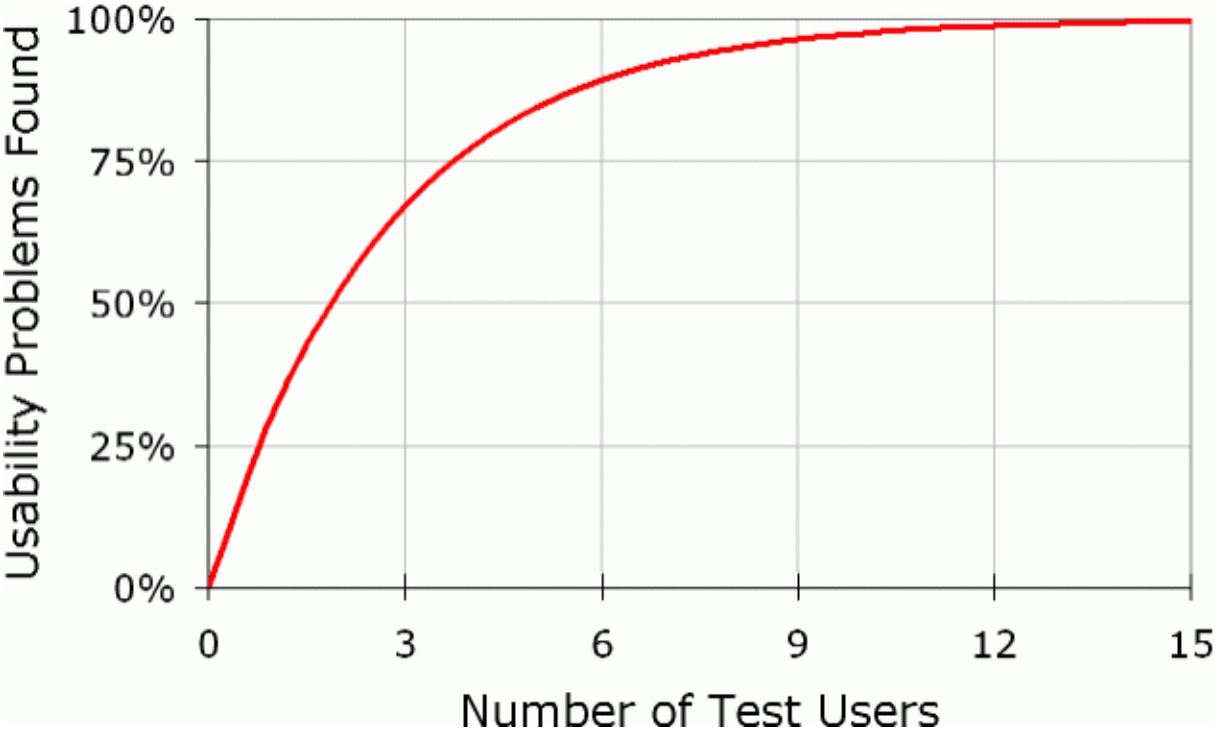

Figure 7.1: A plot illustrating the usability problems found with the users.

which could lead to a qualitative result. For instance, let us say there are two prototypes named A and B. Half of users i..e, three in our case will have prototype A evaluated first and another half i.e., two users will evaluate B first.

7.2 Usability Inspection Methods

There are many Usability inspection methods [27] like Heuristic evaluation, Heuristic estimation, Cognitive walkthrough, Pluralistic walkthrough, Feature inspection, Consistency inspection, Standards inspection and Formal usability inspection. Out of which, the Usability aspect of the prototypes are evaluated with 'Cognitive walkthrough' and 'Heuristic evaluation'.

7.2.1 Cognitive Walkthrough

In Cognitive walkthrough, users are asked to perform tasks with predefined steps. For each step, there are questions examined to determine usability. Blackmon, Polson, et al. in their paper [5] mentions four questions which are significant to analyse while performing Cognitive Walkthrough for the Web. They are;

1. Will the user try and achieve the right outcome?
2. Will the user notice that the correct action is available to them?

3. Will the user associate the correct action with the outcome they expect to achieve?
4. If the correct action is performed; will the user see that progress is being made towards their intended outcome?

These questions are also quite applicable in our context. Thereby, these questions are assessed for each step which is predetermined with designed elements on the user interface in order to solve the Research question been tackled. So, the steps vary for each design. This approach gives qualitative feedback from a user as they are a mostly open-ended scenario to discuss especially, for questions which are answered as 'No' would lead to having their suggestions/feedback.

7.2.2 Heuristic Evaluation

In a Heuristic evaluation, the overall design is evaluated after the user has gone through the interface once or twice understanding the workflow/architecture. Then the design is evaluated with Heuristics which are called rules of thumb in the examination of usability problems. Nielsen proposed ten heuristics as standard such as, Visibility of system status, Match between system and the real world, User control and freedom, Consistency and standards, Error prevention, Recognition rather than recall, Flexibility and efficiency of use, Aesthetic and minimalist design, Help users recognize, diagnose, and recover from errors and Help and documentation.

Once the evaluator went through the design twice, he/she mentions the problems in the context of certain heuristic. They could go through each problem mentioned with severity rating ranging from 0 to 4, where 0 - do not agree this is a usability problem, 1 - cosmetic problem, 2 - minor usability problem, 3 - major usability problem (important to fix), 4 - usability catastrophe (imperative to fix). Finally, the list of problems with severity is gathered from all evaluators. Then, the accumulated results help in re-designing the prototypes taking care that the current issues are addressed for the next iteration of the User Experience Design cycle. This evaluation process also gives valuable qualitative feedback.

Overall, both Cognitive walkthrough and Heuristic evaluation helps to identify the usability problems in detail as possible which is qualitative analysis. Further, when more than two best solution ideas are needed to evaluated against each other then a Heuristic estimate method could be considered. It is simply to estimate which is more accepted by users with a parameter of the majority, where more number of users/evaluators could determine the stronger validity of voting which is quantitative analysis.

8 Time Plan

The Thesis work assumed to follow the below time plan 8.1 as a standard framework. It is primarily focussed on three iterations of the UX Design process with a proper user study as a minimum with the limitation of Thesis time i.e., five months. However, with available resources will strive to perform possible UX Design cycle iterations and reach quality output with this Thesis work.

Each UX Design cycle iteration is planned for one month. It comprises of four steps such as, developing prototypes, plan user study, hold user study and finally assimilating the results of that iteration. Each step is planned to execute in a week time. Next, after an iteration, again develop prototypes with improvements based on previous iteration results. Thereby, three iterations are planned as seen in Figure 8.1.

Milestone 1

As the first milestone, by the end of May month, the first iteration of the UX Design Cycle has to be completed.

Milestone 2

As the second milestone, by the end of June month, the second iteration of the UX Design Cycle has to be completed.

Milestone 3

As the third milestone, by the end of July month, the third iteration of yhe UX Design Cycle has to be completed.

Milestone 4

As the fourth and the final milestone, by the end of September month, the Thesis documentation including presentation has to be completed.

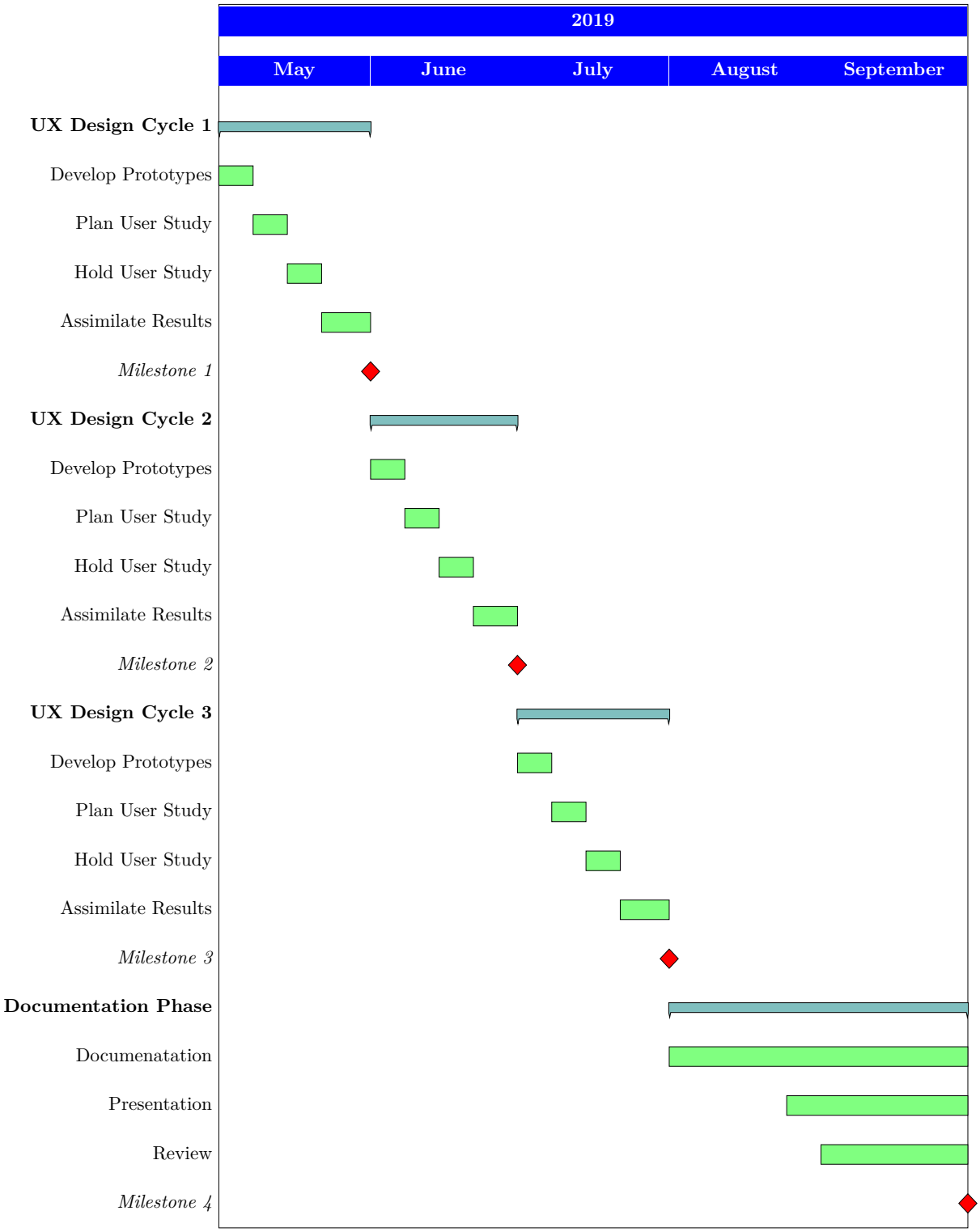

Figure 8.1: Thesis Official Time Plan

9 Conclusion

As software became a part of a day to day life, bug-free software is quite necessary. For which, static code analysis tools are used to identify and mitigate bugs during software development life cycle. Although there are essential tools, in terms of usability they lack to have the attention of developers. Further, the use of multiple static code analysis tools is increasing for obvious reasons like to ensure the bug priority, increase coverage area etc. On the other hand, there is a standardisation process going on with SARIF format and this promises to have generated results in the same format from different analysis tools.

Thereby, this shows the future scope of ongoing research with multiple tools usage. So, when the integration of results from different tools happens, it is necessary to be usable for better usage by developers. This thesis aims to address this issue with three different research questions i.e.,

1. How to display results of the same codebase from different analysis tools?
2. What feedback works to know that the bug fixing is on-going?
3. How to carry traceability of bug fixing?

The research questions are answered by studying different software engineering disciplines and adapt the possible techniques if any. Also, the developers' feedback is analysed through user experience design cycle to make sure the designed prototypes are usable enough to overcome the issues. Hence, the thesis work ensures the applicability of results examined.

Bibliography

[1] *A Survey of Static Program Analysis Techniques*. URL: https://www.ics.uci.edu/~lopes/teaching/inf212W12/readings/Woegerer-progr-analysis.pdf (visited on).

[2] *Balsamiq. Rapid, effective and fun wireframing software. | Balsamiq*. URL: https://balsamiq.com/.

[3] Olga Baysal, Reid Holmes, and Michael W. Godfrey. "No issue left behind: reducing information overload in issue tracking". In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*. Ed. by Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne Storey. New York, New York, USA: ACM Press, 2014, pp. 666–677. ISBN: 9781450330565. DOI: 10.1145/2635868.2635887.

[4] Al Bessey et al. "A few billion lines of code later: using static analysis to find bugs in the real world". In: *Communications of the ACM* 53.2 (2010), pp. 66–75.

[5] Marilyn Hughes Blackmon et al. "Cognitive walkthrough for the web". In: *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM. 2002, pp. 463–470.

[6] Lorraine Borman. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY: ACM, 1985. ISBN: 0897911490. URL: http://dl.acm.org/citation.cfm?id=317456.

[7] *Checkmarx – Application Security Testing and Static Code Analysis*. URL: https://www.checkmarx.com/.

[8] Maria Christakis and Christian Bird. "What developers want and need from program analysis: an empirical study". In: *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference*. IEEE. 2016, pp. 332–343.

[9] John David Colleran, Gerardo Bermudez, and Vadim Gorokhovky. *Responsive user interface to manage a non-responsive application*. US Patent 6,850,257. Feb. 2005.

[10] Aurelien Delaitre et al. "Evaluating Bug Finders–Test and Measurement of Static Code Analyzers". In: *2015 IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS)*. IEEE. 2015, pp. 14–20.

[11] *Designing code analyses for Large Software Systems (DECA)*. URL: https://www.hni.uni-paderborn.de/swt/lehre/deca/.

[12] Alan Dix et al. "Spreadsheets as User Interfaces". In: *Proceedings of the International Working Conference on Advanced Visual Interfaces - AVI '16*. Ed. by Maria Francesca Costabile et al. New York, New York, USA: ACM Press, 2016, pp. 192–195. ISBN: 9781450341318. DOI: 10.1145/2909132.2909271.

[13] dustinca. *Proceedings of the 2nd Workshop on Refactoring Tools*. New York, NY: ACM, 2008. ISBN: 9781605583396. URL: http://dl.acm.org/citation.cfm?id=1636642.

[14] *FindBugs™ - Find Bugs in Java Programs*. URL: http://findbugs.sourceforge.net/.

[15] *FindBugs™ - GUI Scan Results*. URL: http://findbugs.sourceforge.net/manual/gui.html.

[16] Lori Flynn et al. "Prioritizing alerts from multiple static analysis tools, using classification models". In: *Proceedings of the 1st international workshop on software qualities and their dependencies*. ACM. 2018, pp. 13–20.

[17] *Gamification | Coursera*. URL: https://www.coursera.org/learn/gamification.

[18] Garima Gaur, Sumit Kalra, and Arnab Bhattacharya. "Patterns for Indexing Large Datasets". In: *Proceedings of the 23rd European Conference on Pattern Languages of Programs - EuroPLoP '18*. Ed. by Unknown. New York, New York, USA: ACM Press, 2018, pp. 1–6. ISBN: 9781450363877. DOI: 10.1145/3282308.3282314.

[19] Shinpei Hayashi et al. "Historef: A tool for edit history refactoring". In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2/03/2015 - 06/03/2015, pp. 469–473. ISBN: 978-1-4799-8469-5. DOI: 10.1109/SANER.2015.7081858.

[20] Lars Heinemann, Benjamin Hummel, and Daniela Steidl. "Teamscale: Software quality control in real-time". In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 592–595.

[21] James J Horning. "What the compiler should tell the user". In: *Compiler Construction*. Springer. 1974, pp. 525–548.

[22] *How to Change Your Career from Graphic Design to UX Design*. URL: https://www.interaction-design.org/literature/article/how-to-change-your-career-from-graphic-design-to-ux-design.

[23] Brittany Johnson et al. "Why don't software developers use static analysis tools to find bugs?" In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 672–681.

[24] Erica Mealy et al. "Improving Usability of Software Refactoring Tools". In: *2007 Australian Software Engineering Conference (ASWEC'07)*. IEEE, 10/04/2007 - 13/04/2007, pp. 307–318. ISBN: 0-7695-2778-7. DOI: 10.1109/ASWEC.2007.24.

[25] Na Meng et al. "An approach to merge results of multiple static analysis tools (short paper)". In: *2008 The eighth international conference on quality software*. IEEE. 2008, pp. 169–174.

[26] Lisa Nguyen Quang Do and Eric Bodden. "Gamifying Static Analysis". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018. Lake Buena Vista, FL, USA: ACM, 2018, pp. 714–718. ISBN: 978-1-4503-5573-5. DOI: 10.1145/3236024.3264830.

[27] Jakob Nielsen. "Usability inspection methods". In: *Conference companion on Human factors in computing systems*. ACM. 1994, pp. 413–414.

[28] *OASIS*. URL: https://www.oasis-open.org/.

[29] *OASIS SARIF TC*. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sarif.

[30] *OASIS SARIF TC: Repository for development of the draft standard*. URL: https://github.com/oasis-tcs/sarif-spec.

[31] *Observe, Test, Iterate, and Learn (Don Norman) (Video).* URL: https://www.nngroup.com/videos/observe-test-iterate-and-learn-don-norman/.

[32] Daniel Plakosh et al. *Improving the Automated Detection and Analysis of Secure Coding Violations.* Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2014.

[33] *Response Time Limits: Article by Jakob Nielsen.* URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (visited on).

[34] *Sample Of Covered Software Vulnerabilities (OWASP Top 10 and more).* URL: https://www.checkmarx.com/technology/vulnerability-coverage/.

[35] *SARIF Example.* URL: https://blogs.grammatech.com/static-analysis-results-a-format-and-a-protocol-sarif-sasp.

[36] *Software Fail Watch.* URL: https://www.tricentis.com/news/software-fail-watch-says-1-1-trillion-in-assets-affected-by-software-bugs-in-2016/.

[37] *SWAMP SCARF to SARIF.* URL: https://github.com/mirswamp/swamp-scarf-sarif.

[38] *Teamscale.* URL: https://www.cqse.eu/en/products/teamscale/features/.

[39] *The Definition of User Experience (UX).* URL: https://www.nngroup.com/articles/definition-user-experience/.

[40] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. "How do programmers ask and answer questions on the web?" In: *Proceeding of the 33rd international conference on Software engineering - ICSE '11.* Ed. by Richard N. Taylor, Harald Gall, and Nenad Medvidović. New York, New York, USA: ACM Press, 2011, p. 804. ISBN: 9781450304450. DOI: 10.1145/1985793.1985907.

[41] *Usability 101: Introduction to Usability.* URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

[42] *Usability Engineering : Book by Jakob Nielsen.* URL: https://www.nngroup.com/books/usability-engineering/.

[43] Shaowei Wang, David Lo, and Lingxiao Jiang. "An empirical study on developer interactions in StackOverflow". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing.* ACM. 2013, pp. 1019–1024.

[44] *Why You Only Need to Test with 5 Users.* URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.