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

Responsiveness in Static Analysis Tools

by
Sandeep Varma Ganaraju

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

Paderborn, April 1, 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. The Static Analysis plays a major role in Software Development to find bugs and any vulnerabilities in code. There are different Analysis tools available in market. However, it is found out in different surveys about why the tools are not efficient as expected by Software Developers. With the advent of new Static Analysis approaches called as Incremental Analysis, Layered Analysis or Just-in-Time Analysis which aims to compute the analysis in short time. This challenges even further to the developer as to how to integrate such usage of tools in real time scenario. Also, in a typical Software Development Organisations they use multiple tools including legacy tools used in Nightly builds as an example. This Thesis aims to address that scenario where developer works with different tools and how responsive 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 proposed ideas is considered during evaluation phase. The target users for this evaluation are experienced Software Developers which ensures the applicability of this Thesis work.
Keywords: Static Analysis, Incremental Analysis, Layered Analysis, Responsiveness, 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. [31] There are many Static Analysis tools available in market to address these primary issues. However in latest surveys by Maria et al. [7] and by Johnson et al. [21] it is noticed that Software Developers are not quite happy with effectiveness and usability of Static Analysis tools. This brings the scope for improvement of static analysis tools and the paper by Nguyen Quang Do et al. [25] introduces how Gamifying the bug fixing process could enhance the usability of Static Analysis tool. Making User Interface Responsive is presented as one of the challenge in the paper. This Thesis work aims to address that challenge as an overview.

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.

The static analysis tools are evolving from time to time in a direction of reducing computation time. To name a few, they are called Incremental Analysis, Layered Analysis and Just-in-Time Analysis etc. For example, a tool named Cheetah which is introduced as a Just-in-Time Taint Analysis for Android Applications scans a project in less than a second. An other example, a tool with incremental Static Analysis using Path Abstraction named iSATURN proved in the evaluation phase as improvement in scan time by 32 percent over previous SATURN algorithm.

In scenario where an Organisation decides to use different tools as such and especially a legacy tool in combination with newly evolved tool, it leads to disruptive workflow of development process. This brings new challenge on how to make theses tools integrate to the existing

Software Development Life Cycle in less disruptive way by improving the respective User Interface in terms of responsiveness. This opens a new opportunity / challenge which requires Research and thereby this Thesis aims to address it.

1.1 Problem Statement

The overall main aim of the Thesis is about, " *How to integrate the results of multiple static analysis tools?* ". This question was broken down into different Research Questions during Literature Review / Preparatory phase. The 3 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 3. These are evaluated by developing prototypes with the ideas brainstormed during Research and assessed with Software Developers, the responsiveness of User Interface. So, as part of primary contribution of the Thesis is to make sure the ideas evaluated are valid.

1.2 Outline

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

Chapter 2 explains key concepts necessary to understand the work of this Thesis.
Chapter 3 discusses the need of doing the Thesis.
Chapter 4 overviews the goals of the Thesis work.
Chapter 5 explains the way the challenges addressed or attempts to answer the Research Questions.
Chapter 6 evaluates the solution ideas for the challenges or answers to research to Research Questions.
Chapter 7 shows the time plan of doing the Thesis work.

2 Background

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

2.1 Static Analysis

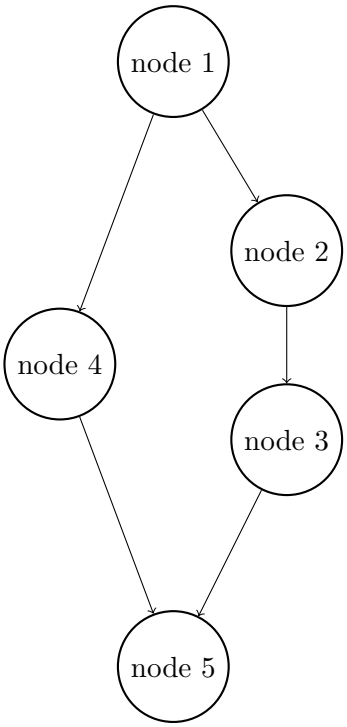
In the last few years, we see the enormous increase in the usage of Software. We see the presence of Software everywhere in the walk of our life with the Internet of Things, for example. As the usage increases, quality is stressed as to be more secure and it does not break. Earlier, Software developers 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. [10]. It reports bugs such as Injections, Cross Site Scripting (XSS), Buffer Overflow, and Dead Code etc. [30] 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
```

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



Another example, it tests whether the variables are tainted along the path from Source where the input is provided by the user or untrustworthy and reaches Sink i.e., end of the path without sanitizing the variable. This is called Taint Analysis. It is also called Information Flow Analysis where Information Flow is defined by Dorothy Denning [9] as "Information flows from object x to object y, denoted x $\rightarrow$ y , whenever information stored in x is transferred to, object y." Thereby, the tainted objects are recognised along the flow and depict which are vulnerable. So, overall Static Analysis has great importance in maintaining the quality of code and bug-free before the code went into the production phase.

The Static Analysis tools using incremental approach are based on the idea that recalculates only the part of the solution that has been modified by the change in program instead of applying complete original algorithm again. Barbara et. al. introduces ACINCF which is incremental update algorithm for forward data flow analysis and ACINCB for backward data flow analysis. Incremental Static Analysis method was patented by Kalman et al. in the year 2012.

The other advanced approach is Layered Static Analysis patented by Cifuentes et al. in the following year 2013, after Incremental Static Analysis is patented. It is a method which helps in reducing the part of source code having potential bugs. It selects part of Static program analysis in an order of less time required to more for each iteration and slowly eradicating the bug free code.

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.

2.2 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. [34] 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 [19] cycle as seen in the figure 2.1 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.

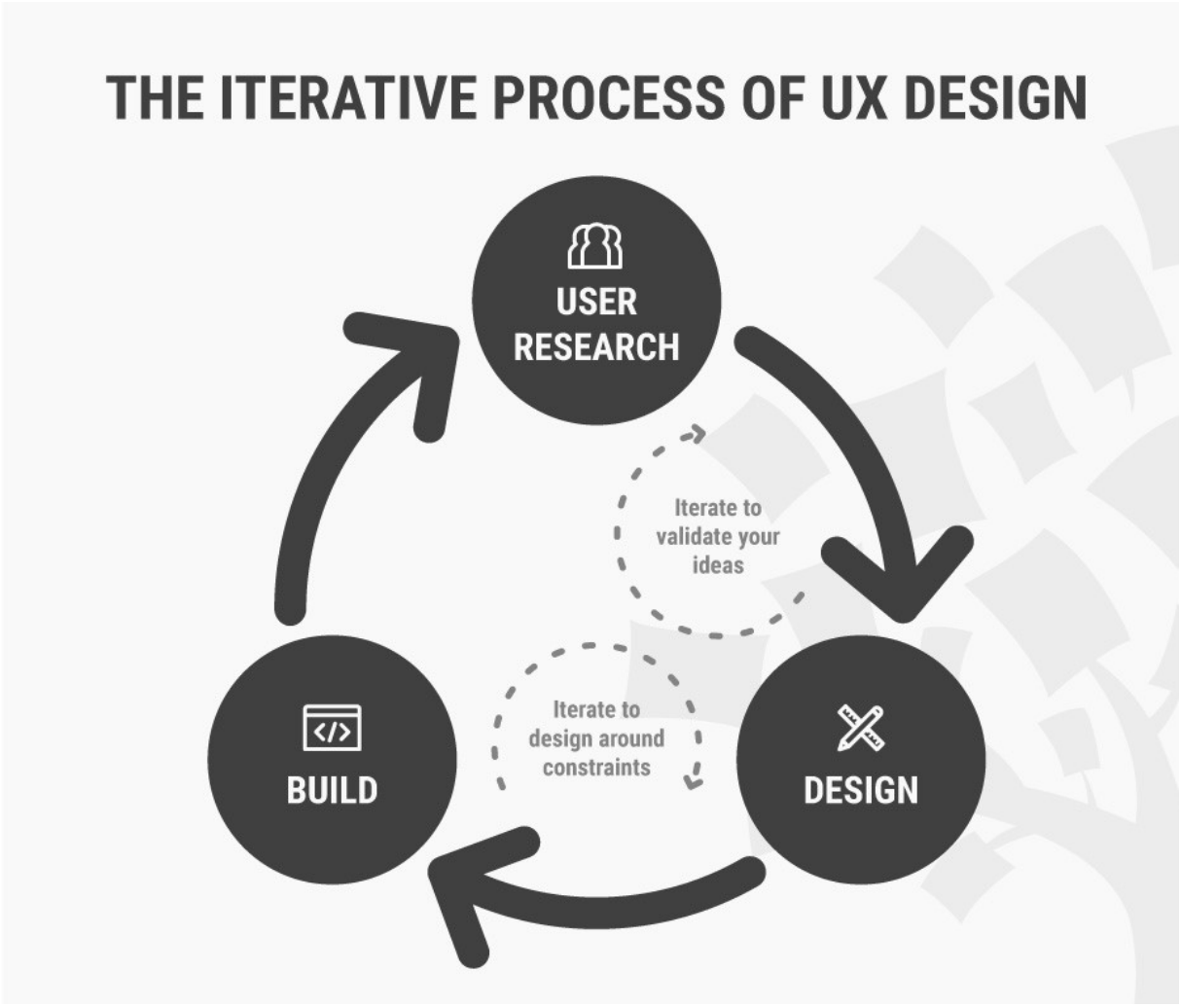


Figure 2.1: UX-Design.[19]

2.3 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 Website Wireframe is shown in the following figure 2.2 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.

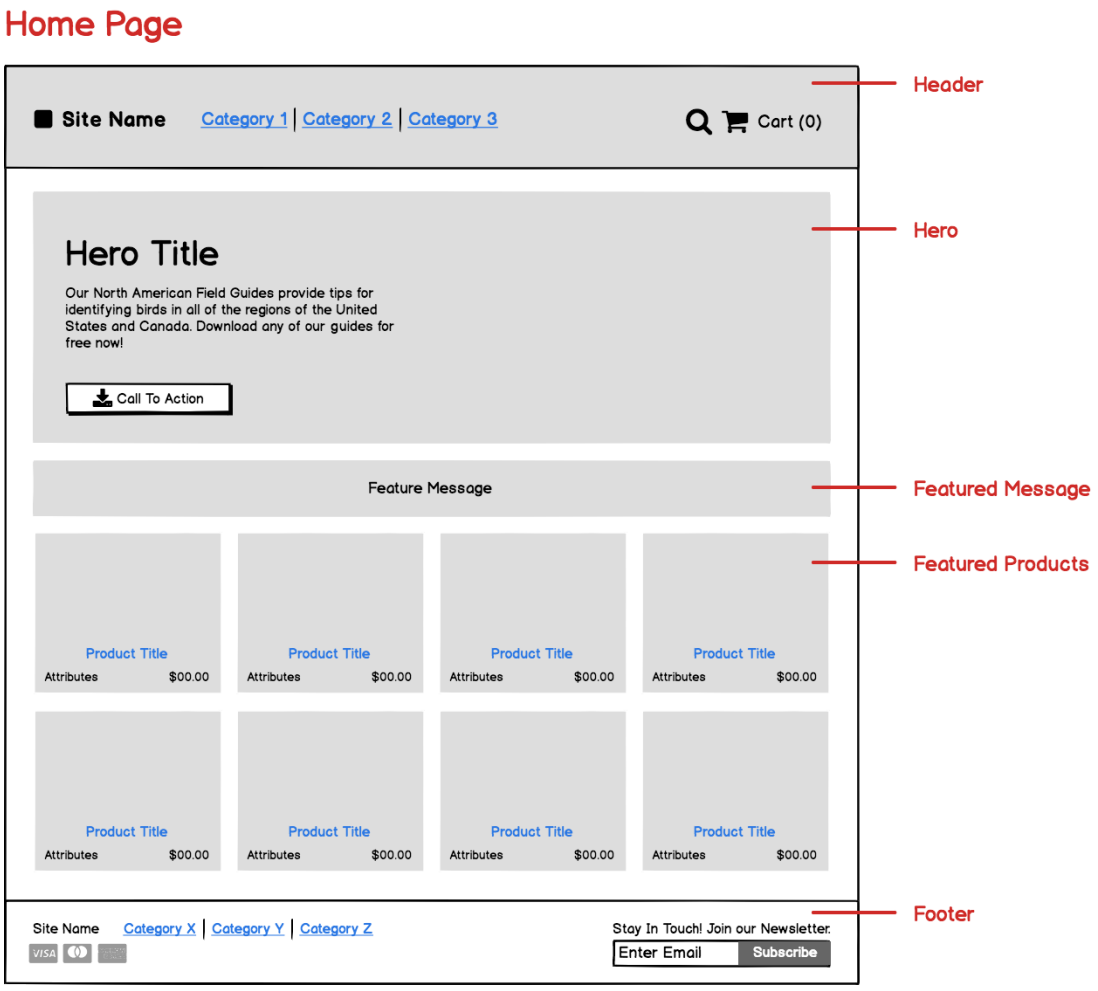


Figure 2.2: Website Wireframe.[2]

3 Motivation

This chapter discusses the need of doing the Thesis.

Static Analysis plays a prominent role in releasing bug-free Software. In spite of their important tools suffer from well-documented usability issues [7, 21]. Maria et. al. [7] 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 UI an obstacle for a developer is noteworthy. Johnson et. al. [21] also found design flaws in current Static Analysis tools and the need for an interactive mechanism in assisting developers in fixing bugs. One interesting aspect of their study is about the importance of feedback from tools without disrupting the developer workflow.

Overall, let us focus on the two aspects mentioned in the above-cited papers i.e., the more time is taken by Analysis tool and Design issues. Regarding the aspect of time, the research is going at a fast pace to make modern Analysis tool capable of producing the results in less time. The Static Analysis mechanisms are evolved from standard mechanisms towards incremental analysis such as ACINF, ACINB Data Flow Analysis tools then to layered analysis, promising the reduction in time for computation. For example, an incremental Static Analysis tool named iSATURN [24] which is built on leveraging the path abstraction properties of SATURN [39] Analysis tool promised to show a 32% reduction in analysis time. Another example, a layered analysis tool called Cheetah [12] which is a Just-in-Time Taint Analysis for Android applications produce analysis results in less than a second. This shows modern tools are capable of giving results in no time. In this way, it is evident the computation time mentioned as a hurdle will be overcome in the near future when these tools start coming into the market. Next, about design, there is a need for better user interfaces of these tools which go in parallel to the development made in Static Analysis mechanisms to really put into practice by end users.

In the design aspect, especially the User Interface needs to be responsive [25] enough as static analysis tools sometimes take a long time to stop and there is no intuitive feedback provided. 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. [8] who patented the method which manages application programs with non-responsive user interfaces in the year 2005. About the response

times, NN Group [28] 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 percent bar [5] 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 is vital to consider in the development of modern tools.

In general, the setup of most of the research 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.

In the current scenario, we could see that 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 [6] tool or could be following Incremental Analysis. The multiplicity of tools usage 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. On that challenge, this Thesis aims to address such a scenario with an overall Research Question as " How to integrate the results of multiple static analysis tools? ".

The main research question is dissected into 3 research questions to make it more precise. Firstly, How to display results from the same codebase from different analysis tools? This question needs to address the scalability aspect and what could be the impact of one fix be on other warnings. Secondly, 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. Finally, 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.

In order to tackle these research questions, different disciplines of software engineering such as Complex datasets [11], [29], [16], Compiler reporting [18], [33], [22], Continuous integration [32] (example Travis [4], [14], [38]), Refactoring tools [13], [17], [23], [26], Issue tracker [3], [20], [27], StackOverflow [37] [35], Gamification [15] , Usability Engineering [36] etc. are looked into and studied what ideas can be adapted into our scenario along with own novel solution ideas.

4 Objectives

This chapter overviews the main goals of this Thesis work.

Static analysis tools suffer from well-documented usability issues [7, 21]. Some analysis tools can report results in milliseconds, while others can take up to hours or days. This discrepancy in waiting times can be confusing to code developers, and in some poorly implemented tools, can lead to them thinking that the tooling froze when it is simply computing results.

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 issue of responsiveness 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 mocking tool [2] of those techniques to improve the usability of analysis tools - with respect to responsiveness- (e.g., FindBugs, Soot, Checkmarx, etc.).

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

5 Approaches

This chapter explains the process considered to answer the Research Questions.

The Thesis work follows the Deduction research approach where the theory is established based on own ideas in addition to literature review findings.

The prototypes are created based on the established theory and user study mentioned in Evaluation chapter 6 is performed which follows Qualitative and Quantitative approaches in assimilating the results. It follows a well established Iterative process of UX Design [34] cycle as seen in the figure 2.1.

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 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.

5.1 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 short time and toolLong mean a Static Analysis tool gives results after long time.

Prototype 1:

The prototype for solution idea i.e., displaying the results separately is shown in following figure 5.1.

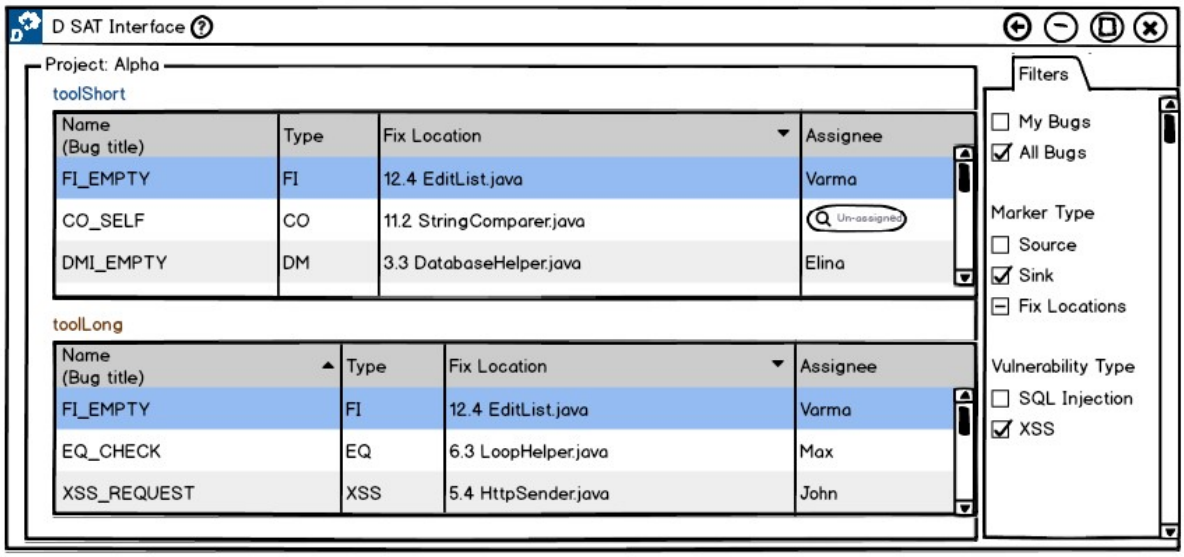


Figure 5.1: An interface prototype with tools displaying results separately.

Prototype 2:

The prototype for solution idea i.e., combining the results is shown in following figure 5.2.

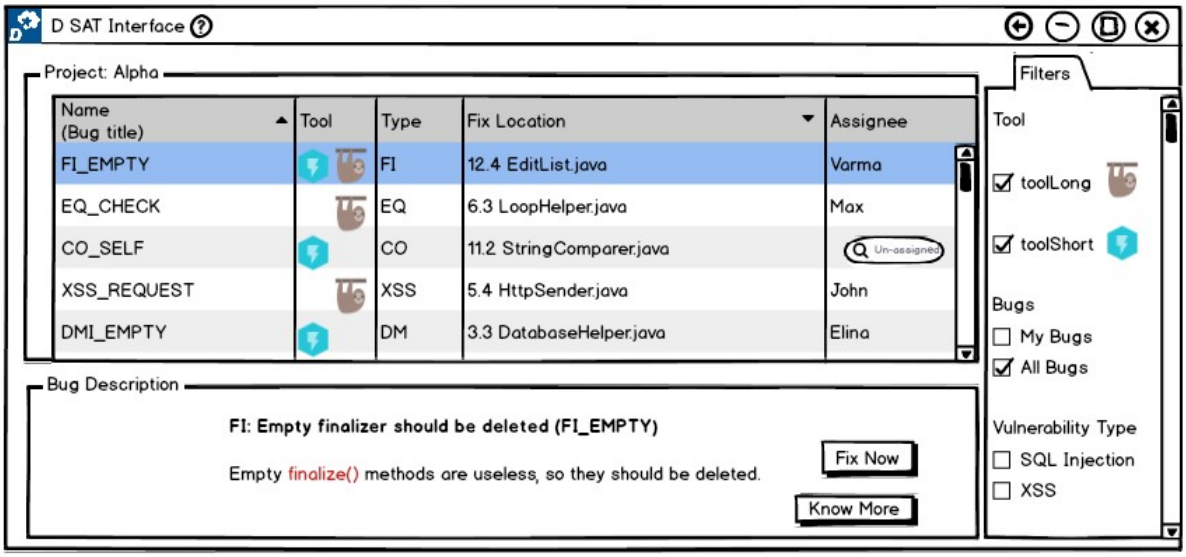


Figure 5.2: An interface prototype with tools displaying results combined.

5.2 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 5.3 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 5.4 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

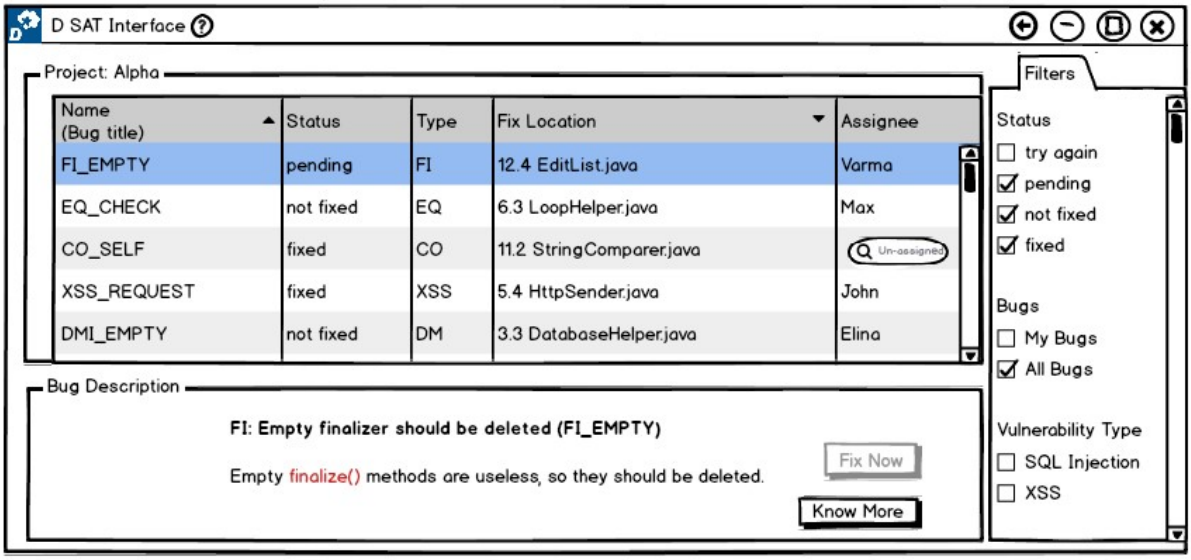


Figure 5.3: An interface prototype showing pending status.

Prototype 2

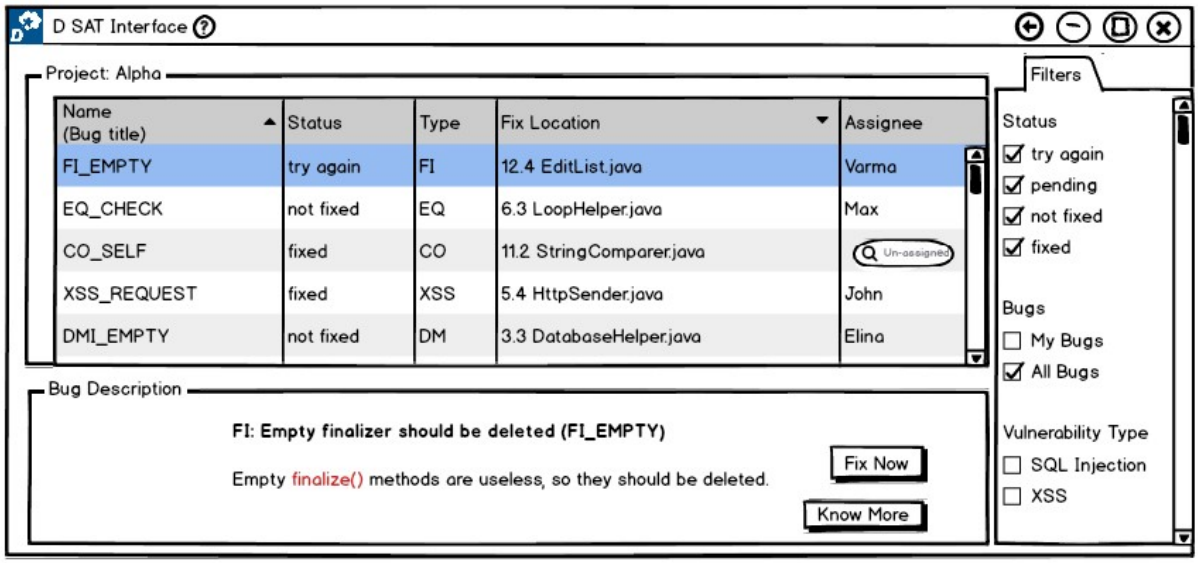


Figure 5.4: An interface prototype showing 'try again' status.

5.3 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 5.5 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 5.6 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.

Prototype 1

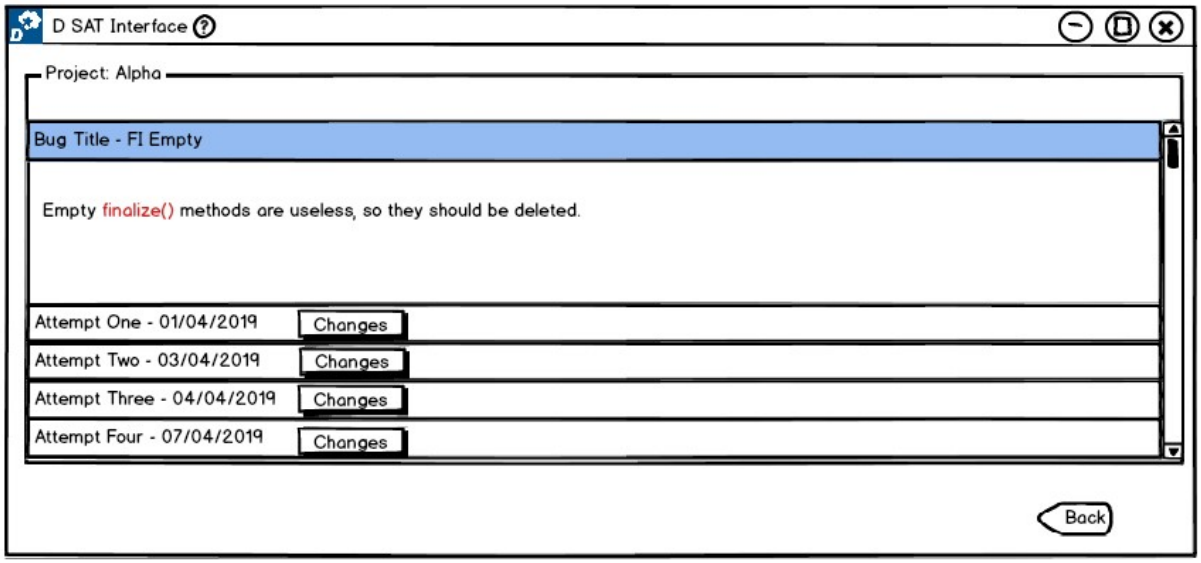


Figure 5.5: An interface prototype showing time stamp and changes button.

Prototype 2

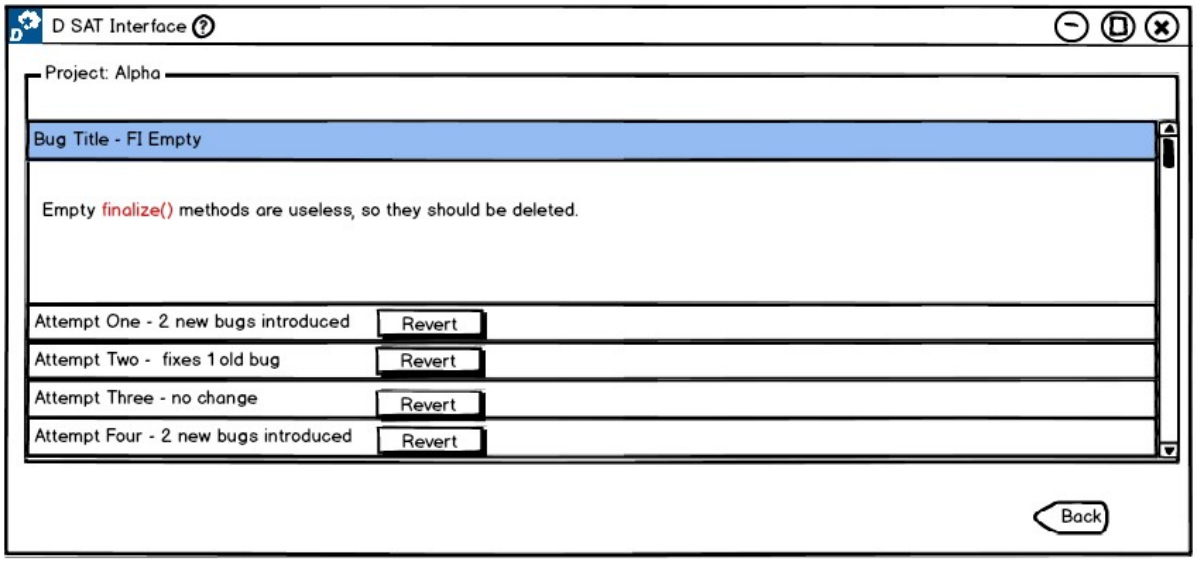


Figure 5.6: An interface prototype showing revert button.

6 Evaluation Plan

This chapter explains how the evaluation of the solution ideas to Research Questions are done.

The primary thing to be done is User Study as per the protocol mentioned below and then assimilate the results gathered during User Study into well-formed answered to our research questions.

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 which could lead to a qualitative result.

Here is the draft for User Study protocol.

The User Study is performed following this protocol as guidelines.

Phase 1: Set up study environment

The solution ideas for the Research Questions are assimilated into best 3 prototypes prepared using Mockup tool i.e., Balsamiq. These prototypes as exported as pdf and saved at Online drive which can be given access to the user once he agrees to participate in the study.

Phase 2: Organize appointments with users

An invitation will be sent to users typically software developers and preferably who use Static Analysis tools or at least having minimum experience. This ensures the results to be qualitative. A Consent form is signed to make sure the collected user data can be used for this user study safely without violating user privacy.

Phase 3: Introduction

The User Study is introduced once the user shows up at the appointment. Once again, what this user study about and what data is collected during the process.

Phase 4: General Questions

Video recording is started. The user is asked with questions related to his software development experience and with Static Analysis tools.

Phase 5: Performance of tasks / Observing the usage

The user is given with the task to fix a bug in a way that one could do on an interface. The task would be the following as an example;

- Select the project
- Observe the bugs notification
- Select a bug
- After observing the code editor,
click on an option on screen to indicate the user has fixed the bug

During this process, the user gives feedback on how he/she feels by the responsive nature of the Analysis tool. This could be analysed with predetermined questions to make results quantitative.

This process is repeated for the three prototypes.

Phase 6: Wrap up

The user is thanked for participation and recorded video is saved for data extraction.

Phase 7: Data Extraction

The questionnaire asked during the user study session is filled out in form for analysis in comparison to other users' responses.

This helps to conclude whether the researched Static Analysis Tools User Interfaces are responsive as per the user expectations and usage.

7 Time Plan

The Thesis work assumed to follow the below time plan 7.1 as a standard framework. It is primarily focussed on one iteration of the UX Design process with a proper user study as a minimum with limitation of Thesis time. However, with available resources will strive to perform possible UX Design iterations and reach quality output with this Thesis work.

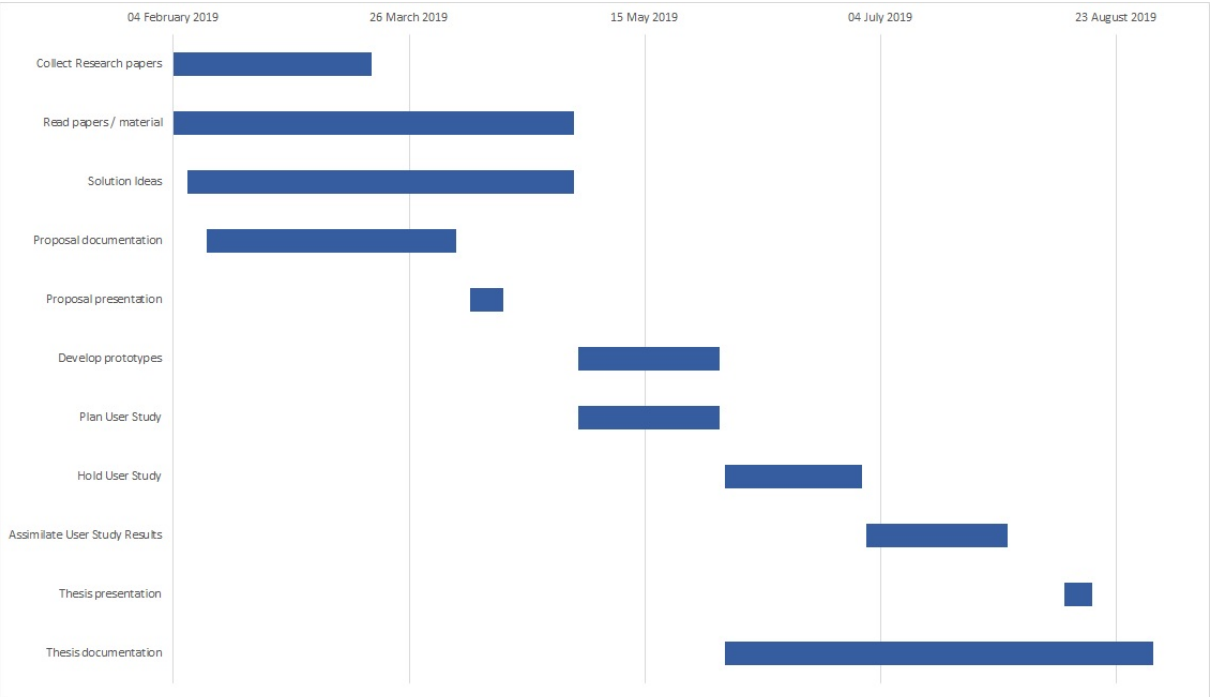


Figure 7.1: Time Plan

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] Moritz Beller, Georgios Gousios, and Andy Zaidman. "Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 20/05/2017 - 21/05/2017, pp. 356–367. ISBN: 978-1-5386-1544-7. DOI: 10.1109/MSR.2017.62.

[5] 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.

[6] *Checkmarx – Application Security Testing and Static Code Analysis*. URL: https://www.checkmarx.com/ (visited on).

[7] 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.

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

[9] Dorothy E. Denning. "A Lattice Model of Secure Information Flow". In: *Commun. ACM* 19.5 (May 1976), pp. 236–243. ISSN: 0001-0782. DOI: 10.1145/360051.360056. URL: http://doi.acm.org/10.1145/360051.360056.

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

[11] 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.

[12] Lisa Nguyen Quang Do et al. "Just-in-time Static Analysis". In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2017. Santa Barbara, CA, USA: ACM, 2017, pp. 307–317. ISBN: 978-1-4503-5076-1. DOI: 10.1145/3092703.3092705. URL: http://doi.acm.org/10.1145/3092703.3092705.

[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] Keheliya Gallaba et al. "Noise and heterogeneity in historical build data: an empirical study of Travis CI". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018*. Ed. by Marianne Huchard, Christian Kästner, and Gordon Fraser. New York, New York, USA: ACM Press, 2018, pp. 87–97. ISBN: 9781450359375. DOI: 10.1145/3238147.3238171.

[15] *Gamification | Coursera*. URL: https://www.coursera.org/learn/gamification (visited on).

[16] 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.

[17] 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.

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

[19] *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.

[20] Kasthuri Jayarajah, Meera Radhakrishnan, and Camellia Zakaria. "Duplicate issue detection for the Android open source project". In: *Proceedings of the 5th International Workshop on Software Mining - SoftwareMining 2016*. Ed. by Ming Li, Xiaoyin Wang, and Lucia. New York, New York, USA: ACM Press, 2016, pp. 24–31. ISBN: 9781450345118. DOI: 10.1145/2975961.2975965.

[21] 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.

[22] Yannis Lilis and Anthony Savidis. "LNCS 7304 - Supporting Compile-Time Debugging and Precise Error Reporting in Meta-programs". In: ().

[23] 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.

[24] Murali Krishna Mudduluru Rashmi and Ramanathan. "Efficient Incremental Static Analysis Using Path Abstraction". In: *Fundamental Approaches to Software Engineering*. Ed. by Stefania Gnesi and Arend Rensink. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 125–139. ISBN: 978-3-642-54804-8.

[25] 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.

[26] Gustavo H. Pinto and Fernando Kamei. "What programmers say about refactoring tools?" In: *Proceedings of the 2013 ACM workshop on Workshop on refactoring tools - WRT '13*. Ed. by Emerson Murphy-Hill and Max Schaefer. New York, New York, USA: ACM Press, 2013, pp. 33–36. ISBN: 9781450326049. DOI: 10.1145/2541348.2541357.

[27] Arif Raza, Luiz Fernando Capretz, and Faheem Ahmed. "Maintenance support in open source software projects". In: *Eighth International Conference on Digital Information Management (ICDIM 2013)*. IEEE, 10/09/2013 - 12/09/2013, pp. 391–395. ISBN: 978-1-4799-0615-4. DOI: 10.1109/ICDIM.2013.6694005.

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

[29] Shanshan Ruan et al. "Density Peaks Clustering for Complex Datasets". In: *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*. IEEE, 20/10/2016 - 21/10/2016, pp. 87–92. ISBN: 978-1-5090-5952-2. DOI: 10.1109/IIKI.2016.20.

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

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

[32] Daniel Stahl, Kristofer Hallen, and Jan Bosch. "Continuous Integration and Delivery Traceability in Industry: Needs and Practices". In: *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 31/08/2016 - 02/09/2016, pp. 68–72. ISBN: 978-1-5090-2820-7. DOI: 10.1109/SEAA.2016.12.

[33] Chengnian Sun, Vu Le, and Zhendong Su. "Finding and analyzing compiler warning defects". In: *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. Ed. by Laura Dillon, Willem Visser, and Laurie Williams. New York, New York, USA: ACM Press, 2016, pp. 203–213. ISBN: 9781450339001. DOI: 10.1145/2884781.2884879.

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

[35] 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.

[36] *Usability Engineering : Book by Jakob Nielsen*. URL: https://www.nngroup.com/books/usability-engineering/ (visited on).

[37] 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.

[38] David Gray Widder et al. “I’m leaving you, Travis”. In: *Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18*. Ed. by Andy Zaidman, Yasutaka Kamei, and Emily Hill. New York, New York, USA: ACM Press, 2018, pp. 165–169. ISBN: 9781450357166. DOI: 10.1145/3196398.3196422.

[39] Yichen Xie and Alex Aiken. “Saturn: A Scalable Framework for Error Detection Using Boolean Satisfiability”. In: *ACM Trans. Program. Lang. Syst.* 29.3 (May 2007). ISSN: 0164-0925. DOI: 10.1145/1232420.1232423. URL: http://doi.acm.org/10.1145/1232420.1232423.